

BeagleBoard Docs

Release 1.0.20231205-wip



BeagleBoard.org Foundation

Dec 05, 2023

Table of contents

1 Introduction **3**
- 1.1 An Introduction to Beagles . . . 3
  - 1.1.1 Blink LED . . . 3
  - 1.1.2 An Introduction to Linux . . . 16
  - 1.1.3 QWIIC, STEMMA and Grove Add-ons in Linux . . . 17
  - 1.1.4 Linux-enabled boards . . . 19
  - 1.1.5 Zephyr-enabled boards . . . 20
- 1.2 Support . . . 20
  - 1.2.1 Getting Started Guide . . . 20
  - 1.2.2 First, read the manual . . . 27
  - 1.2.3 Getting support . . . 27
  - 1.2.4 Understanding Your Beagle . . . 29
- 1.3 Contribution . . . 29
  - 1.3.1 Code of Conduct . . . 29
  - 1.3.2 Frequently Asked Questions . . . 30
  - 1.3.3 What should I know before I get started? . . . 30
  - 1.3.4 How can I contribute? . . . 30
  - 1.3.5 Articles on contribution . . . 40

2 BeaglePlay **57**
- 2.1 Introduction . . . 58
  - 2.1.1 Detailed overview . . . 59
- 2.2 Quick Start Guide . . . 63
  - 2.2.1 What’s included in the box? . . . 63
  - 2.2.2 Attaching antennas . . . 64
  - 2.2.3 Tethering to PC . . . 65
  - 2.2.4 Access VSCode . . . 65
  - 2.2.5 Demos and Tutorials . . . 65
- 2.3 Design and specifications . . . 67
  - 2.3.1 Block diagram . . . 67
  - 2.3.2 System on Chip (SoC) . . . 68
  - 2.3.3 Power management . . . 68
  - 2.3.4 General Connectivity and Expansion . . . 71
  - 2.3.5 Buttons and LEDs . . . 73
  - 2.3.6 Wired and wireless connectivity . . . 74
  - 2.3.7 Memory, Media and Data storage . . . 76
  - 2.3.8 Multimedia I/O . . . 76
  - 2.3.9 RTC & Debug . . . 76
  - 2.3.10 Mechanical Specifications . . . 79
- 2.4 Expansion . . . 81
  - 2.4.1 mikroBUS . . . 81
  - 2.4.2 Grove . . . 81
  - 2.4.3 QWIIC . . . 81
  - 2.4.4 CSI . . . 81
  - 2.4.5 OLDI . . . 82
- 2.5 Demos and tutorials . . . 83
  - 2.5.1 Using Serial Console . . . 83

2.5.2 Connect WiFi . . . 83
2.5.3 Using Grove . . . 93
2.5.4 Using mikroBUS . . . 93
2.5.5 Using QWIIC . . . 98
2.5.6 Using OLDI Displays . . . 99
2.5.7 Using CSI Cameras . . . 99
2.5.8 Wireless MCU Zephyr Development . . . 99
2.5.9 Introduction . . . 105
2.5.10 BeagleConnect™ Greybus demo using BeagleConnect™ Freedom and BeaglePlay . . . 106
2.6 Support . . . 111
2.6.1 Certifications and export control . . . 111
2.6.2 Additional documentation . . . 111
2.6.3 Change History . . . 112

3 BeagleBone AI-64 113
3.1 Introduction . . . 114
3.1.1 BeagleBone Compatibility . . . 114
3.1.2 BeagleBone AI-64 Features and Specification . . . 115
3.1.3 Board Component Locations . . . 115
3.1.4 Board components . . . 115
3.2 Quick Start Guide . . . 116
3.2.1 What's In the Box . . . 116
3.2.2 Methods of operation . . . 117
3.2.3 Update software . . . 124
3.2.4 Next steps . . . 125
3.3 Design and Specifications . . . 125
3.3.1 Block Diagram and Overview . . . 125
3.4 Expansion . . . 132
3.4.1 Cape Header Connectors . . . 132
3.4.2 RANDOM PRU STUFF THAT MIGHT NEED A HOME . . . 156
3.5 Demos and Tutorials . . . 157
3.5.1 Edge AI . . . 157
3.6 Additional Support Information . . . 199
3.6.1 Hardware Design . . . 199
3.6.2 Software Updates . . . 199
3.6.3 RMA Support . . . 200
3.6.4 Troubleshooting video output issues . . . 200
3.6.5 Getting Help . . . 200
3.6.6 Change History . . . 200
3.6.7 Mechanical Details . . . 200
3.6.8 Pictures . . . 201

4 BeagleBone AI 205
4.1 Introduction . . . 206
4.2 Connecting Up Your BeagleBone AI . . . 207
4.2.1 What's In the Box . . . 207
4.2.2 What's Not in the Box . . . 207
4.2.3 Fans . . . 207
4.2.4 Main Connection Scenarios . . . 208
4.2.5 Tethered to a PC . . . 208
4.2.6 Standalone w/Display and Keyboard/Mouse . . . 212
4.2.7 Wireless Connection . . . 213
4.2.8 Connecting a 3 PIN Serial Debug Cable . . . 213
4.3 BeagleBone AI Overview . . . 214
4.3.1 BeagleBone® AI Features . . . 214
4.3.2 Board Component Locations . . . 216
4.4 BeagleBone AI High Level Specification . . . 216
4.4.1 Block Diagram . . . 216
4.4.2 AM572x Sitara™ Processor . . . 217

4.4.3 Memory . . . 221
4.4.4 Boot Modes . . . 221
4.4.5 Power Management . . . 222
4.4.6 Connectivity . . . 222
4.5 Detailed Hardware Design . . . 222
4.5.1 Power Section . . . 223
4.5.2 eMMC Flash Memory (16GB) . . . 225
4.5.3 Wireless Communication: 802.11 ac & Bluetooth: AzureWave AW-CM256SM . . . 225
4.5.4 HDMI . . . 226
4.5.5 PRU-ICSS . . . 226
4.5.6 PRU-ICSS Resources and FAQ's . . . 227
4.5.7 User LEDs . . . 238
4.6 Connectors . . . 238
4.6.1 Expansion Connectors . . . 239
4.6.2 Serial Debug . . . 254
4.6.3 USB 3 Type-C . . . 255
4.6.4 USB 2 Type-A . . . 255
4.6.5 Gigabit Ethernet . . . 255
4.6.6 Coaxial . . . 255
4.6.7 microSD Memory . . . 255
4.6.8 microHDMI . . . 255
4.7 Cape Board Support . . . 255
4.7.1 BeagleBone® Black Cape Compatibility . . . 255
4.7.2 EEPROM . . . 256
4.7.3 Pin Usage Consideration . . . 256
4.7.4 GPIO . . . 256
4.7.5 I2C . . . 256
4.7.6 UART or PRU UART . . . 256
4.7.7 SPI . . . 257
4.7.8 Analog . . . 257
4.7.9 PWM, TIMER, eCAP or PRU PWM/eCAP . . . 257
4.7.10 eQEP . . . 257
4.7.11 CAN . . . 257
4.7.12 McASP (audio serial like I2S and AC97) . . . 257
4.7.13 MMC . . . 257
4.7.14 LCD . . . 257
4.7.15 PRU GPIO . . . 257
4.7.16 CLKOUT . . . 257
4.7.17 Expansion Connector Headers . . . 257
4.7.18 Signal Usage . . . 257
4.7.19 Cape Power . . . 258
4.7.20 Mechanical . . . 258
4.8 Additional Support Information . . . 258
4.8.1 REGULATORY, COMPLIANCE, AND EXPORT INFORMATION . . . 258
4.8.2 Mechanical Information . . . 258
4.8.3 Change History . . . 258
4.8.4 Pictures . . . 260

5 BeagleBone Black 263
5.1 Introduction . . . 264
5.2 Change History . . . 264
5.2.1 Document Change History . . . 264
5.2.2 Board Changes . . . 266
5.3 Connecting Up Your BeagleBone Black . . . 268
5.3.1 What's In the Box . . . 268
5.3.2 Main Connection Scenarios . . . 268
5.3.3 Tethered To A PC . . . 269
5.3.4 Standalone w/Display and Keyboard/Mouse . . . 271

5.4 BeagleBone Black Overview . . . 276
5.4.1 BeagleBone Compatibility . . . 276
5.4.2 BeagleBone Black Features and Specification . . . 277
5.4.3 Board Component Locations . . . 278
5.5 BeagleBone Black High Level Specification . . . 279
5.5.1 Block Diagram . . . 279
5.5.2 Processor . . . 279
5.5.3 Memory . . . 279
5.5.4 Power Management . . . 282
5.5.5 PC USB Interface . . . 282
5.5.6 Serial Debug Port . . . 283
5.5.7 USB1 Host Port . . . 283
5.5.8 Power Sources . . . 283
5.5.9 Reset Button . . . 283
5.5.10 Power Button . . . 283
5.5.11 Indicators . . . 284
5.5.12 CTI JTAG Header . . . 284
5.5.13 HDMI Interface . . . 284
5.5.14 Cape Board Support . . . 284
5.6 Detailed Hardware Design . . . 285
5.6.1 Power Section . . . 285
5.6.2 Sitara AM3358BZCZ100 Processor . . . 295
5.6.3 4GB eMMC Memory . . . 300
5.6.4 Board ID EEPROM . . . 302
5.6.5 Micro Secure Digital . . . 302
5.6.6 6.6 User LEDs . . . 303
5.6.7 Boot Configuration . . . 304
5.6.8 Default Boot Options . . . 305
5.6.9 10/100 Ethernet . . . 305
5.6.10 LAN8710A Mode Pins . . . 307
5.6.11 HDMI Interface . . . 307
5.6.12 USB Host . . . 311
5.6.13 PRU-ICSS . . . 313
5.7 Connectors . . . 315
5.7.1 Expansion Connectors . . . 315
5.7.2 Power Jack . . . 321
5.7.3 USB Client . . . 321
5.7.4 USB Host . . . 321
5.7.5 Serial Header . . . 321
5.7.6 HDMI . . . 324
5.7.7 microSD . . . 324
5.7.8 Ethernet . . . 327
5.7.9 JTAG Connector . . . 327
5.8 Cape Board Support . . . 327
5.8.1 BeagleBone Black Cape Compatibility . . . 328
5.8.2 EEPROM . . . 330
5.8.3 Pin Usage Consideration . . . 335
5.8.4 Expansion Connectors . . . 336
5.8.5 8.5 Signal Usage . . . 338
5.8.6 8.6 Cape Power . . . 339
5.8.7 8.7 Mechanical . . . 340
5.9 BeagleBone Black Mechanical . . . 341
5.9.1 Dimensions and Weight . . . 341
5.9.2 Silkscreen and Component Locations . . . 341
5.10 Pictures . . . 341
5.11 Support Information . . . 341
5.11.1 Hardware Design . . . 341
5.11.2 Software Updates . . . 347

5.11.3 RMA Support . . . 347
5.11.4 Trouble Shooting HDMI Issues . . . 347

6 BeagleBone Blue 351
6.1 BeagleBone Blue Pinouts . . . 352
6.1.1 UT1 . . . 353
6.1.2 GPS . . . 353
6.2 SSH . . . 353
6.3 WiFi Setup . . . 354
6.4 IP settings . . . 354
6.5 Flashing Firmware . . . 355
6.5.1 Overview . . . 355
6.5.2 Required Items . . . 355
6.5.3 Steps Overview . . . 355
6.5.4 Windows PCs . . . 355
6.6 Play with the code . . . 356
6.7 BeagleBone Blue tests . . . 357
6.7.1 ADC . . . 357
6.7.2 GP0 . . . 357
6.7.3 GP1 . . . 357
6.7.4 UT1 . . . 357
6.7.5 GPS . . . 357
6.7.6 I2C . . . 358
6.7.7 Motors . . . 358
6.8 Accessories . . . 358
6.8.1 Chassis and kits . . . 358
6.8.2 Cases . . . 358
6.8.3 Cable assemblies and sub-assemblies . . . 358
6.8.4 UART, I2C, CAN, Quadrature encoders, PWR . . . 359
6.8.5 SPI, GPIO, ADC . . . 359
6.8.6 Motors . . . 359
6.8.7 DSM . . . 359
6.8.8 Power supplies . . . 360
6.8.9 Motors . . . 360
6.8.10 Radio remotes . . . 360
6.8.11 GPS . . . 360
6.8.12 Replacement antennas . . . 360
6.8.13 USB devices . . . 361
6.8.14 SPI devices . . . 361
6.8.15 I2C devices . . . 361
6.8.16 UART devices . . . 361
6.8.17 Bluetooth devices . . . 361
6.9 Frequently Asked Questions (FAQs) . . . 361
6.9.1 Are there any books to help me get started? . . . 361
6.9.2 What system firmware should I use for starting to explore my BeagleBone Blue? . . . 361
6.9.3 What is the name of the access point SSID and password default on BeagleBone Blue? . . . 362
6.9.4 I've connected to BeagleBone Blue's access point. How do I get logged into the board? . . . 362
6.9.5 How do I connect BeagleBone Blue to my own WiFi network? . . . 362
6.9.6 Where can I find examples and APIs for programming BeagleBone Blue? . . . 362
6.9.7 My BeagleBone Blue fails to run successful tests . . . 362
6.9.8 I'm running an image off of a microSD card. How do I write it to the on-board eMMC flash? 362
6.9.9 I've written the latest image to a uSD card, but some features aren't working. How do I make it run properly? . . . 363
6.9.10 I've got my on-board eMMC flash configured in a nice way. How do I copy that to other BeagleBone Blue boards? . . . 363
6.9.11 I have some low-latency I/O tasks. How do I get started programming the BeagleBone PRUs? 363
6.9.12 Are there available mechanical models? . . . 363
6.9.13 What is the operating temperature range? . . . 363

6.9.14 What is the DC motor drive strength? . . . 364

7 BeagleBone (all) 365

8 BeagleV-Ahead 367
8.1 Introduction . . . 368
8.1.1 Pinout Diagrams . . . 368
8.1.2 Detailed overview . . . 371
8.2 Quick Start . . . 373
8.2.1 What's included in the box? . . . 373
8.2.2 Unboxing . . . 374
8.2.3 Antenna guide . . . 375
8.2.4 Tethering to PC . . . 375
8.2.5 Flashing eMMC . . . 376
8.2.6 Access UART debug console . . . 380
8.2.7 Connect USB gadgets . . . 382
8.2.8 Connect to WiFi . . . 382
8.2.9 Demos and Tutorials . . . 384
8.3 Design & specifications . . . 384
8.3.1 Block diagram . . . 384
8.3.2 System on Chip (SoC) . . . 385
8.3.3 Power management . . . 385
8.3.4 General Connectivity and Expansion . . . 389
8.3.5 Buttons and LEDs . . . 389
8.3.6 Wired and wireless connectivity . . . 389
8.3.7 Memory, Media and Data storage . . . 389
8.3.8 Multimedia I/O . . . 395
8.3.9 Debug . . . 395
8.3.10 Mechanical Specifications . . . 401
8.4 Expansion . . . 403
8.4.1 Cape Headers . . . 403
8.5 Demos . . . 410
8.5.1 Using CSI Cameras . . . 410
8.6 Support . . . 411
8.6.1 Certifications and export control . . . 411
8.6.2 Additional documentation . . . 411
8.6.3 Change History . . . 412

9 BeagleV-Fire 413
9.1 Introduction . . . 414
9.1.1 Pinout Diagrams . . . 414
9.1.2 Detailed overview . . . 415
9.2 Quick Start . . . 417
9.2.1 What's included in the box? . . . 418
9.2.2 Unboxing . . . 418
9.2.3 Tethering to PC . . . 419
9.2.4 Flashing eMMC . . . 419
9.2.5 Access UART debug console . . . 419
9.2.6 Demos and Tutorials . . . 420
9.3 Design & specifications . . . 420
9.3.1 Block diagram . . . 420
9.3.2 System on Chip (SoC) . . . 420
9.3.3 Power management . . . 420
9.3.4 General Connectivity and Expansion . . . 428
9.3.5 Buttons and LEDs . . . 428
9.3.6 Connectivity . . . 428
9.3.7 Memory, Media and Data storage . . . 428
9.3.8 Multimedia I/O . . . 434
9.3.9 Debug . . . 434

9.3.10 Mechanical Specifications . . . 434
9.4 Expansion . . . 436
9.4.1 Cape Headers . . . 436
9.5 Demos . . . 442
9.5.1 Flashing gateware and Linux image . . . 442
9.5.2 Microchip FPGA Tools Installation Guide . . . 447
9.5.3 Gateware Design Introduction . . . 451
9.5.4 How to retrieve BeagleV-Fire's gateware version . . . 453
9.6 Support . . . 454
9.6.1 Certifications and export control . . . 454
9.6.2 Additional documentation . . . 455
9.6.3 Change History . . . 455

10 Capes **457**
10.1 BeagleBone cape interface spec . . . 457
10.1.1 Background and overview . . . 458
10.1.2 Digital GPIO . . . 460
10.1.3 I²C . . . 462
10.1.4 SPI . . . 464
10.1.5 UART . . . 465
10.1.6 CAN . . . 465
10.1.7 ADC . . . 466
10.1.8 PWM . . . 467
10.1.9 TIMER PWM . . . 468
10.1.10 Counter . . . 469
10.1.11 eCAP . . . 470
10.1.12 MMC/SDIO . . . 471
10.1.13 LCD . . . 472
10.1.14 I²S . . . 472
10.1.15 PRU . . . 473
10.1.16 Dynamic overlays . . . 477
10.1.17 Dynamic pinmux control . . . 477
10.1.18 Methodology . . . 477
10.1.19 References . . . 478
10.2 BeagleBoard.org BeagleBone Relay Cape . . . 478
10.2.1 Installation . . . 479
10.2.2 Usage . . . 479
10.2.3 Code to Get Started . . . 480
10.2.4 C Source with File Descriptors . . . 480
10.2.5 C Source with LibGPIOd-dev and File Descriptors . . . 481

11 PocketBeagle **485**
11.1 Introduction . . . 486
11.2 Change History . . . 486
11.2.1 Document Change History . . . 487
11.2.2 Board Changes . . . 487
11.3 Connecting Up PocketBeagle . . . 488
11.3.1 What's In the Package . . . 488
11.3.2 Connecting the board . . . 488
11.3.3 Tethered to a PC using Debian Images . . . 488
11.3.4 Other ways to Connect up to your PocketBeagle . . . 502
11.4 PocketBeagle Overview . . . 502
11.4.1 PocketBeagle Features and Specification . . . 502
11.4.2 Board Component Locations . . . 503
11.5 PocketBeagle High Level Specification . . . 504
11.5.1 Block Diagram . . . 504
11.5.2 System in Package (SiP) . . . 504
11.5.3 Connectivity . . . 504
11.5.4 Power . . . 507

11.5.5 JTAG Pads . . . 508
11.5.6 Serial Debug Port . . . 508
11.6 Detailed Hardware Design . . . 509
11.6.1 OSD3358-SM SiP Design . . . 509
11.6.2 MicroSD Connection . . . 510
11.6.3 USB Connector . . . 510
11.6.4 Power Button Design . . . 510
11.6.5 User LEDs . . . 516
11.6.6 JTAG Pads . . . 516
11.6.7 PRU-ICSS . . . 516
11.7 Connectors . . . 520
11.7.1 Expansion Header Connectors . . . 520
11.7.2 P1 Header . . . 520
11.7.3 P2 Header . . . 524
11.7.4 mikroBUS socket connections . . . 527
11.7.5 Setting up an additional USB Connection . . . 527
11.8 PocketBeagle Cape Support . . . 528
11.9 PocketBeagle Mechanical . . . 528
11.9.1 9.1 Dimensions and Weight . . . 528
11.10 Additional Pictures . . . 528
11.11 Support Information . . . 528
11.11.1 Hardware Design . . . 530
11.11.2 Software Updates . . . 530
11.11.3 Export Information . . . 530
11.11.4 RMA Support . . . 530
11.11.5 Getting Help . . . 530

12 BeagleConnect Freedom **531**
12.1 Introduction . . . 532
12.1.1 What is BeagleConnect™ Freedom? . . . 532
12.1.2 What makes BeagleConnect™ new and different? . . . 533
12.2 Quick Start Guide . . . 534
12.2.1 What's included in the box? . . . 534
12.2.2 Attaching antenna . . . 535
12.2.3 Tethering to PC . . . 536
12.2.4 Wireless Connection . . . 536
12.2.5 Access Micropython . . . 536
12.2.6 Demos and Tutorials . . . 536
12.3 Design . . . 536
12.3.1 Detailed overview . . . 536
12.3.2 Detailed hardware design . . . 536
12.3.3 Mechanical . . . 541
12.4 Connectors . . . 541
12.5 Demos & tutorials . . . 541
12.5.1 Using Micropython . . . 543
12.5.2 Using Zephyr . . . 546
12.5.3 Using BeagleConnect Greybus . . . 547
12.6 Support . . . 551
12.6.1 Certifications and export control . . . 551
12.6.2 Additional documentation . . . 551
12.6.3 Change History . . . 551
12.6.4 Document Changes . . . 551

13 BeagleBoard (all) **553**

14 Projects **555**
14.1 simpPRU . . . 555
14.1.1 simpPRU Basics . . . 555
14.1.2 Build from source . . . 556

14.1.3 Install . . . 556
14.1.4 Language Syntax . . . 557
14.1.5 IO Functions . . . 566
14.1.6 Usage(simppru) . . . 571
14.1.7 Usage(simppru-console) . . . 571
14.1.8 simpPRU Examples . . . 575
14.2 BB-Config . . . 589
14.2.1 BB-Config Detail . . . 589
14.2.2 Build from Source . . . 591
14.2.3 Features . . . 591
14.2.4 Version . . . 601
14.3 Robotics Control Library . . . 602
14.4 BeagleConnect Technology . . . 602
14.4.1 Background . . . 603
14.4.2 High-level . . . 603
14.4.3 Software architecture . . . 603
14.4.4 TODO items . . . 603
14.4.5 Associated pre-work . . . 605
14.4.6 User experience concerns . . . 605
14.4.7 BeagleConnect™ Greybus demo using BeagleConnect™ Freedom . . . 605

15 Books **621**
15.1 BeagleBone Cookbook . . . 621
15.1.1 Basics . . . 621
15.1.2 Sensors . . . 631
15.1.3 Displays and Other Outputs . . . 658
15.1.4 Motors . . . 671
15.1.5 Beyond the Basics . . . 685
15.1.6 Internet of Things . . . 710
15.1.7 The Kernel . . . 749
15.1.8 Real-Time I/O . . . 761
15.1.9 Capes . . . 773
15.1.10 Parts and Suppliers . . . 802
15.1.11 Misc . . . 804
15.2 PRU Cookbook . . . 823
15.2.1 Case Studies - Introduction . . . 823
15.2.2 Getting Started . . . 847
15.2.3 Running a Program; Configuring Pins . . . 858
15.2.4 Debugging and Benchmarking . . . 868
15.2.5 Building Blocks - Applications . . . 885
15.2.6 Accessing More I/O . . . 955
15.2.7 More Performance . . . 961
15.2.8 Moving to the BeagleBone AI . . . 971
15.2.9 PRU Projects . . . 978

16 Accessories **985**
16.1 Power supplies . . . 987
16.2 Displays . . . 988
16.2.1 Monitors and Resolutions . . . 988
16.3 Peripherals . . . 989
16.3.1 Keyboard & Mouse Combo . . . 989
16.3.2 Keyboards . . . 990
16.3.3 Mice . . . 990
16.3.4 USB HUBS . . . 990
16.4 Cables . . . 990
16.4.1 USB Data/Power Cables . . . 990
16.4.2 Serial Debug Cables . . . 991
16.4.3 JTAG debug Cables . . . 992
16.4.4 HDMI Cables . . . 992

16.4.5 miniDP to HDMI . . . 993
16.5 Cameras . . . 993
16.5.1 USB Cameras . . . 993
16.5.2 CSI Cameras . . . 993

17 Terms & Conditions **995**
17.1 Design . . . 995
17.2 Additional terms . . . 995
17.3 United States FCC and Canada IC regulatory compliance information . . . 996
17.4 Board warnings, restrictions and disclaimers . . . 996

This adds content to the print version that is in /index.rst, but skipped in /index-tex.rst

Chapter 1

Introduction

Welcome to the BeagleBoard project documentation. If you are looking for help with your Beagle open-hardware development platform, you've found the right place!

Important: This documentation is a work in progress. For the latest versions of this documentation, be sure to check the official release sites:

- https://docs.beagle.cc (cached with local proxies)
- https://docs.beagleboard.org (non-cached, without proxies)

For bleeding edge (development-stage) documentation:

- https://docs.beagleboard.io (straight from docs repo)

Get started quickly on our Linux-enabled boards with *Blink LED*, follow-up with articles in *An Introduction to Beagles*, and reach out via resources on our *Support* page as needed to resolve issues and engage with the developer community. Don't forget that this is an open-source project! Your contributions are welcome. Learn about how to contribute to the BeagleBoard documentation project and any of the many open-source Beagle projects ongoing on our *Contribution* page.

Warning: Make sure you thoroughly read and agree with our *Terms & Conditions* which covers warnings, restrictions, disclaimers, and warranty for all of our boards. Use of either the boards or the design materials constitutes agreement to the T&C including any modifications done to the hardware or software solutions provided by the BeagleBoard.org Foundation.

1.1 An Introduction to Beagles

1.1.1 Blink LED

The "Hello World!" of the embedded world is to blink an LED. Here we'll show you how to do just that in three simple steps.

1. Plug in the Beagle
2. Log into the Beagle
3. Blink the LED

These steps will work for any of the Beagles.

Plug in the Beagle

For this step you need to get a USB cable and attach your Beagle to your host computer with it. Where you attached the cable depends on which Beagle you have. Click on the tab for your board.

Black

Blue

AI-64

Play

Pocket

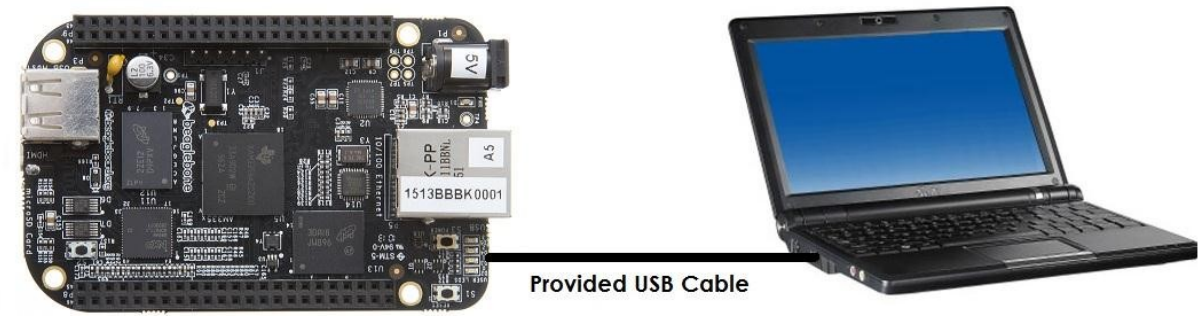


Fig. 1.1: Tethered Configuration

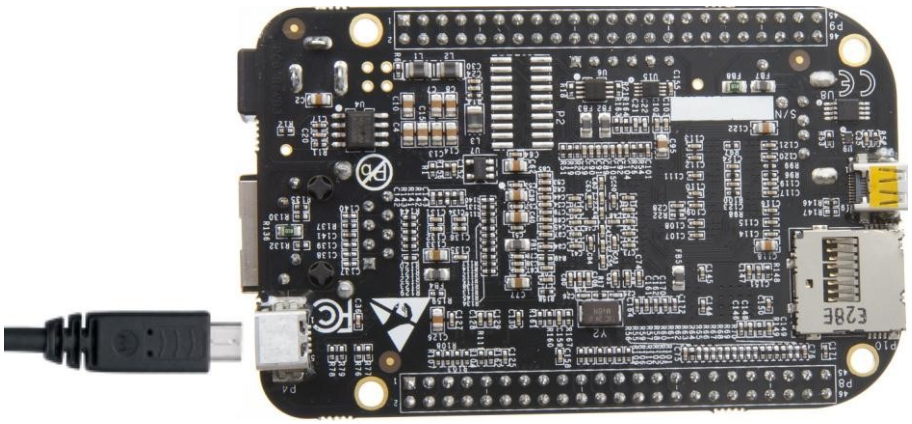


Fig. 1.2: Mini USB Connection to the Board as seen from the bottom.

For more details see: *Connecting Up Your BeagleBone Black*

For more details see: *Quick Start Guide*

For more details see: *Quick Start Guide*

For more details see: *Connecting Up PocketBeagle*

Once attached you will see some LEDs blinking. Wait a bit and the blinking will settle down to a steady heart beat.

The Beagle is now up and running, but you didn't have to load up Linux. This is because all Beagles (except PocketBeagle, see *Update board with latest software* to install an image on the Pocket) have built-in flash memory that has the Debian distribution of Linux preinstalled.

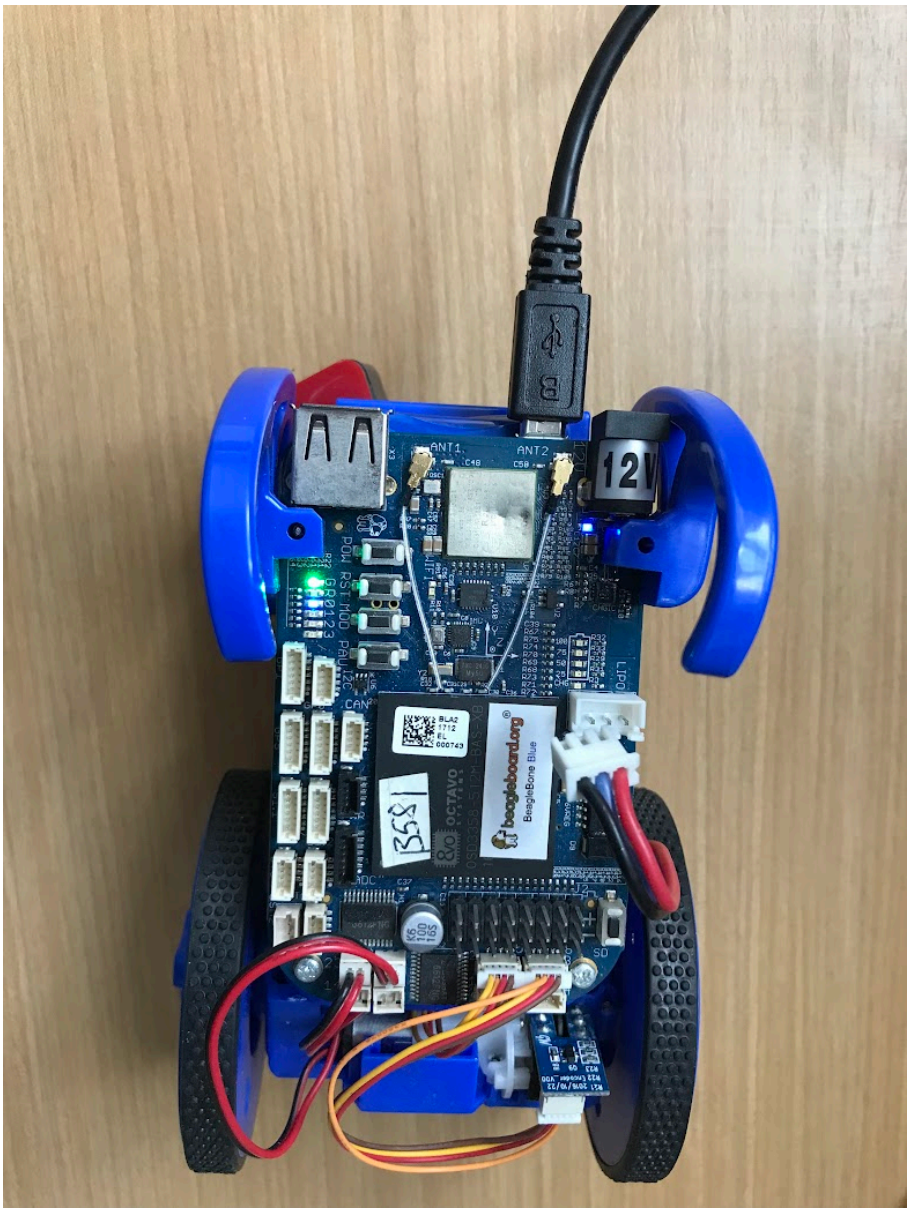


Fig. 1.3: Micro USB Connection to the Blue

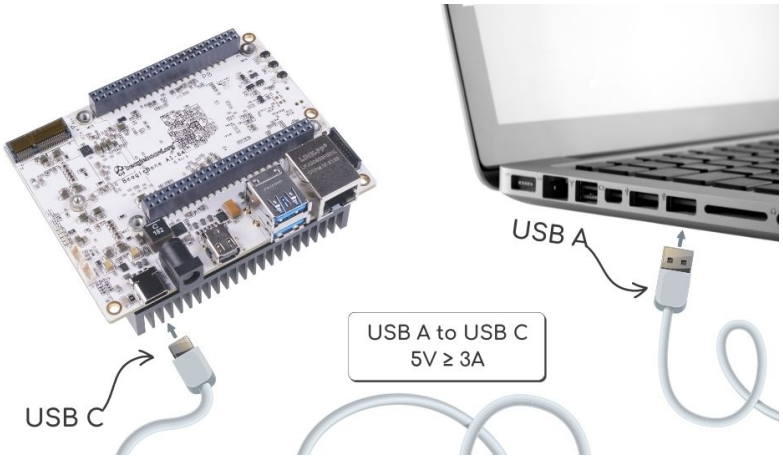


Fig. 1.4: Tethered Configuration

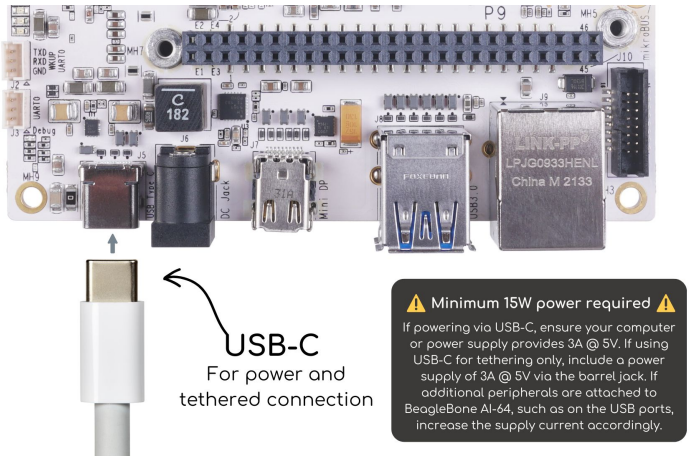


Fig. 1.5: USB-c Connection to the Board

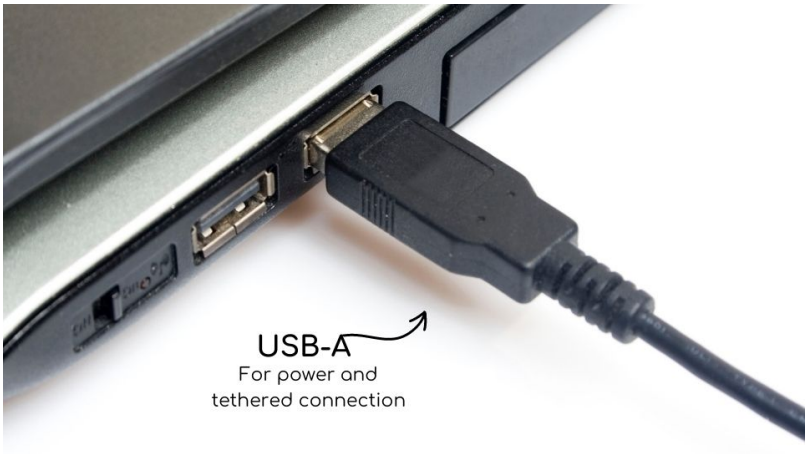


Fig. 1.6: USB Connection to the PC/Laptop

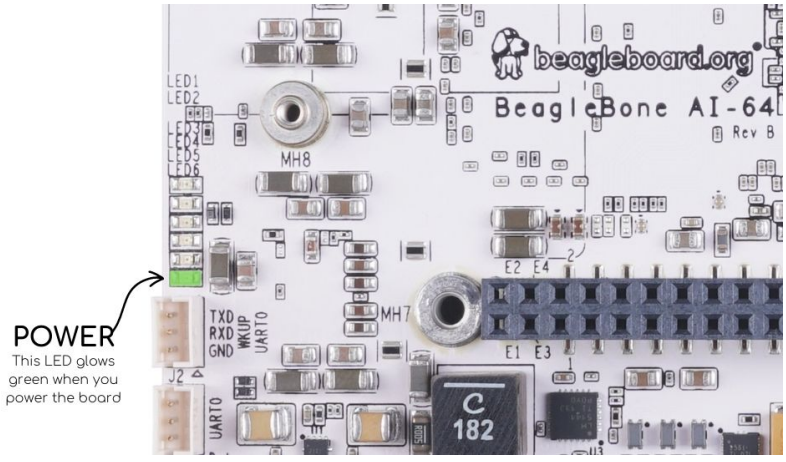


Fig. 1.7: Board Power LED

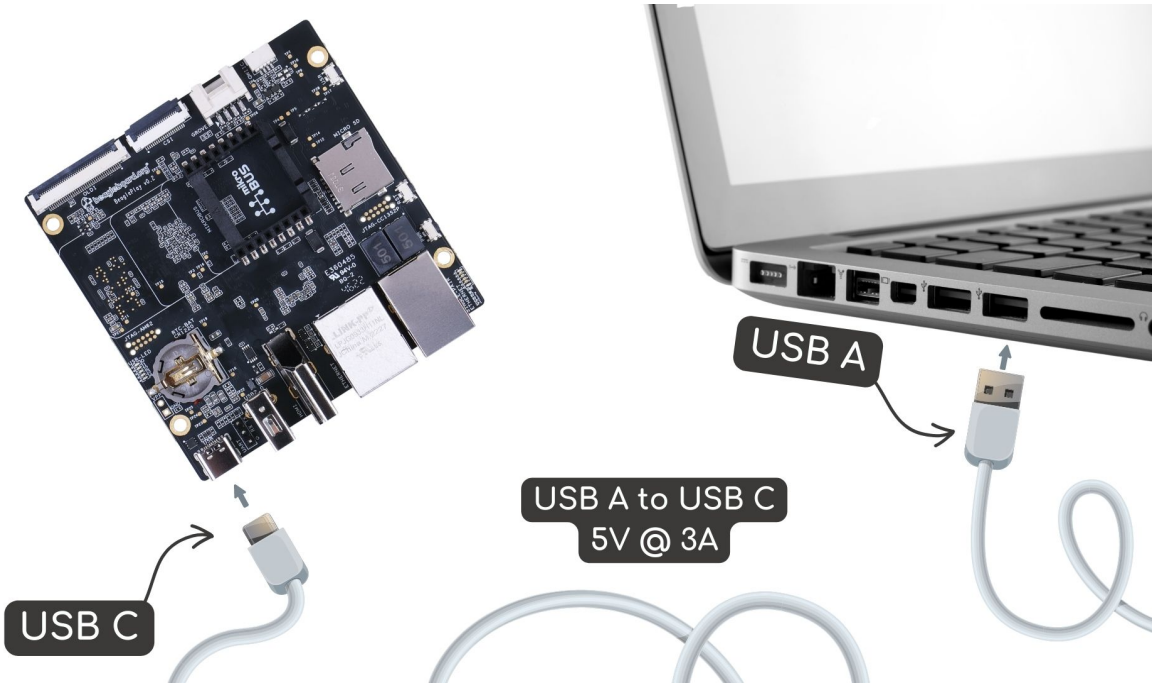


Fig. 1.8: Tethering BeaglePlay to PC

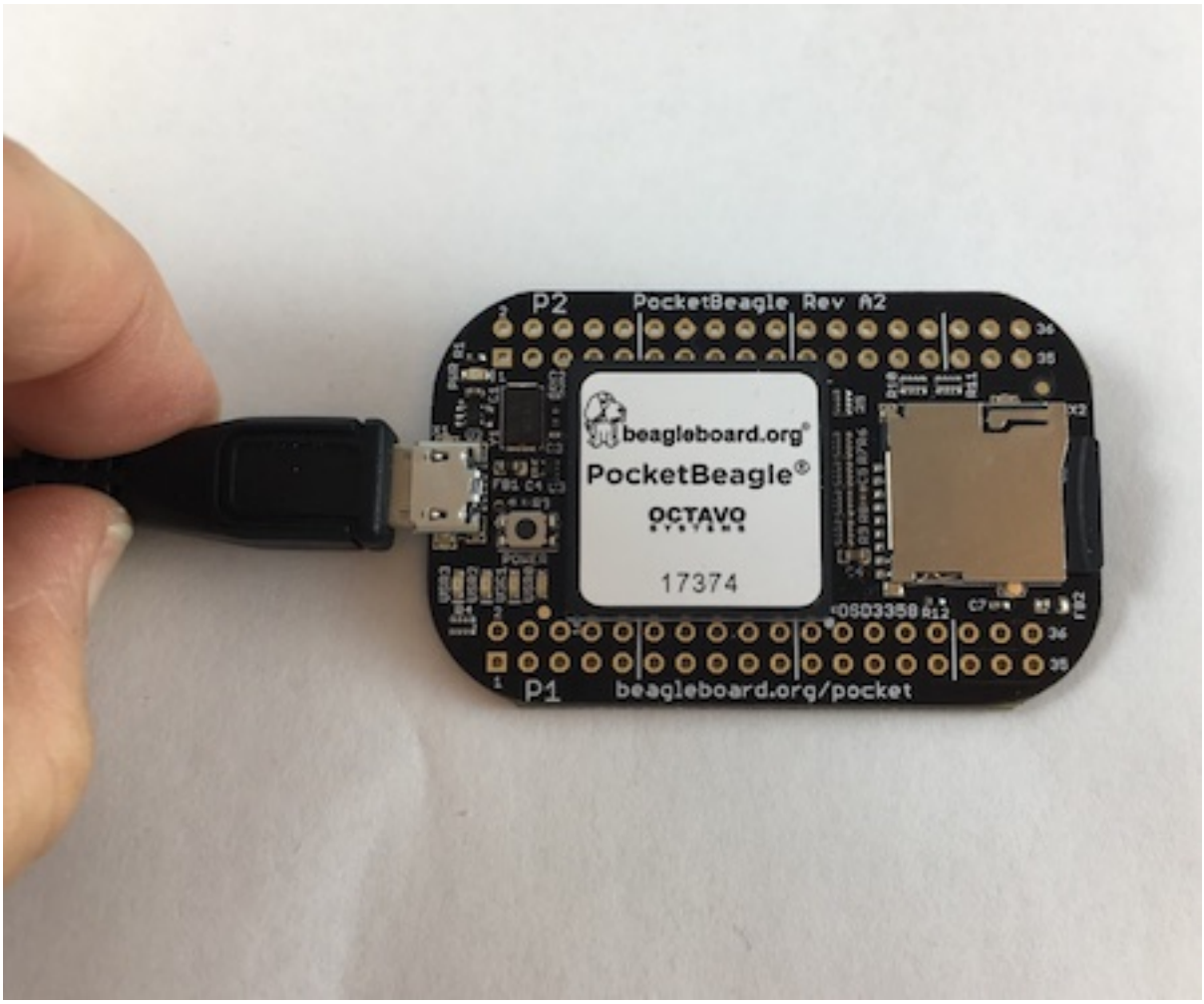
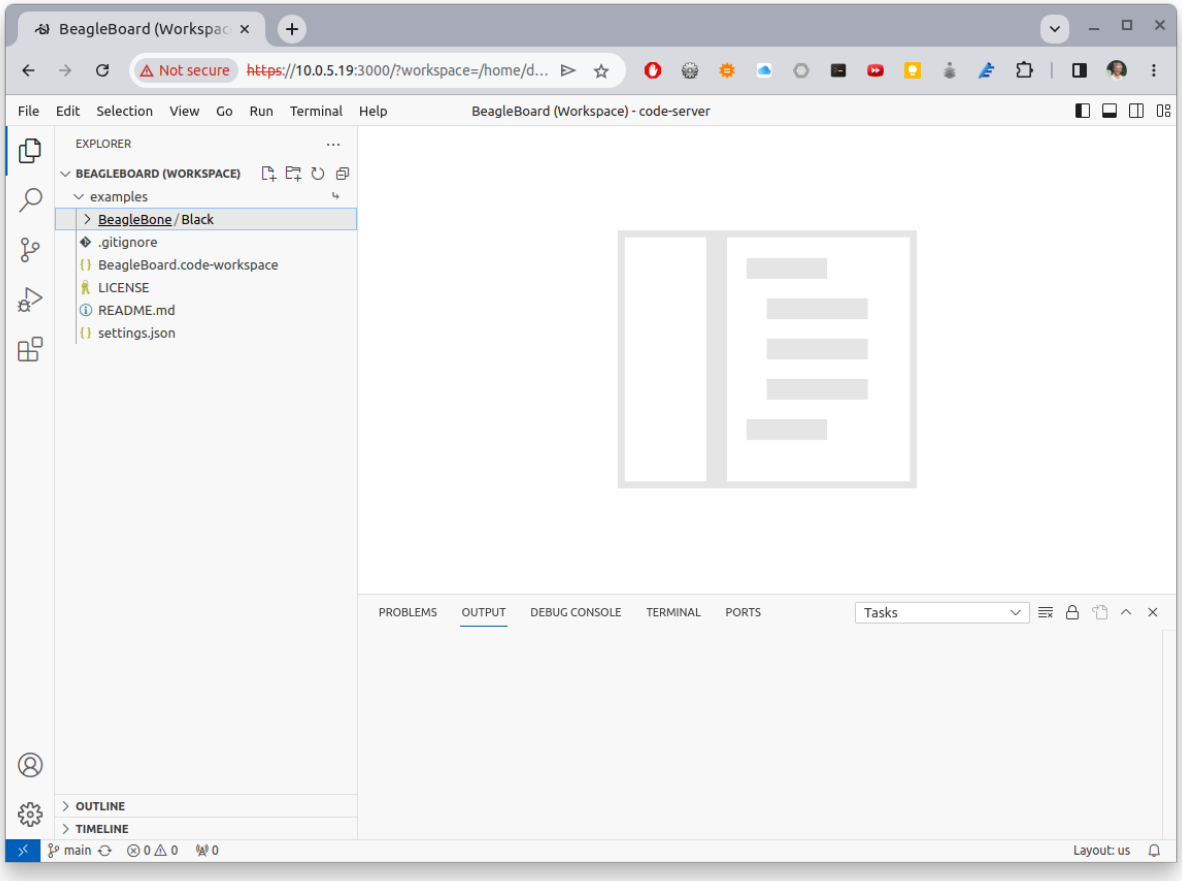


Fig. 1.9: Micro USB Connection

Using VS Code

Important: If VS code is not installed on your board please skip this section and refer to next section on how to login and run the code via command line.

Recent Beagles come with the IDE Visual Studio Code (https://code.visualstudio.com/) installed and running. To access it, open a web browser on your host computer and browse to: `192.168.7.2:3000` (use `192.168.6.2:3000` for the Mac) and you will see something like:



At this point you can either run the scripts via a command line within VS Code, or run them by clicking the `RUN Code` button.

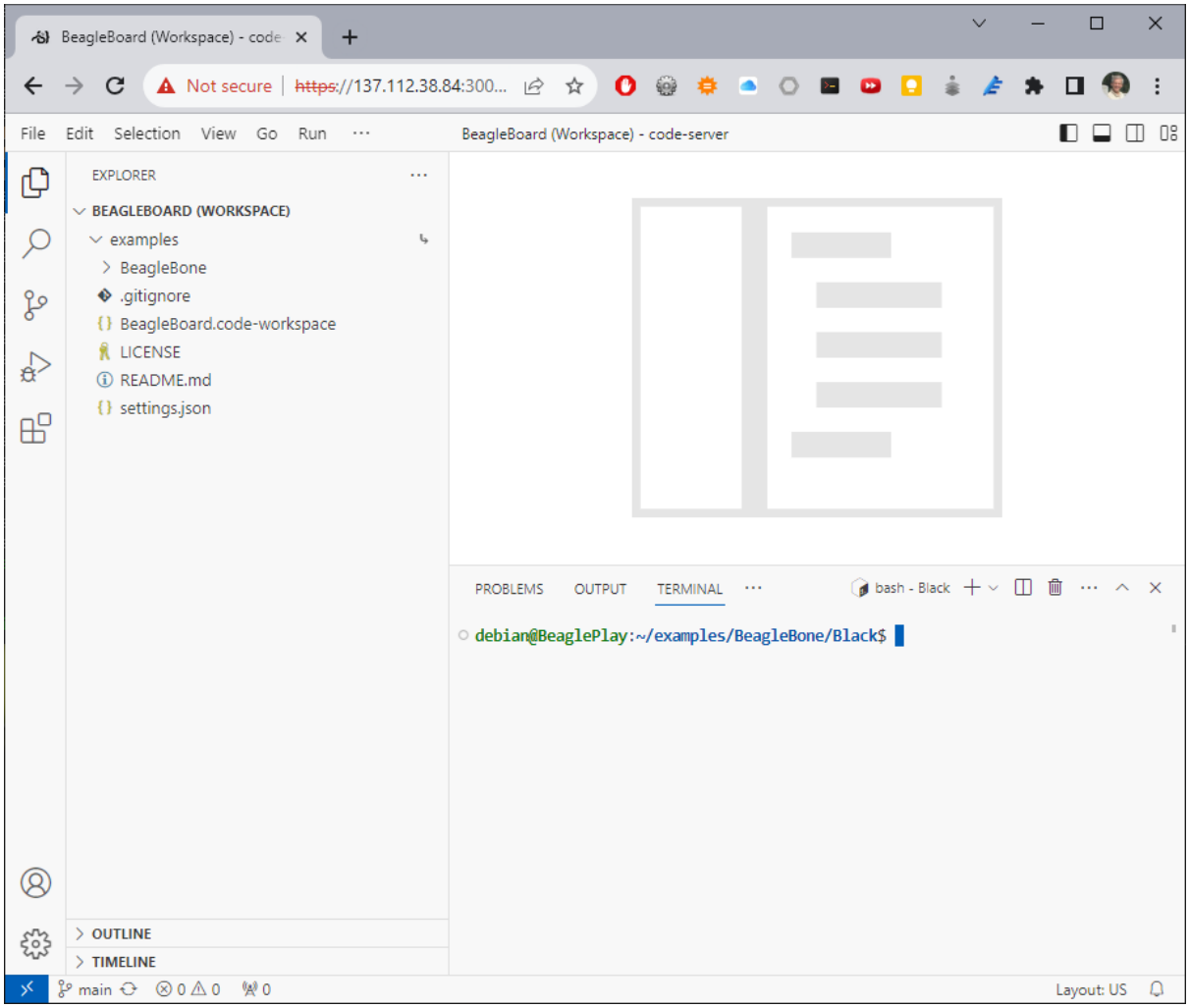
Running via the command line Open a terminal window in VS Code by dropping down the `Terminal` menu and selecting `New Terminal` (or entering `Ctrl+`` `). The terminal window appears at the bottom of the screen as shown below.

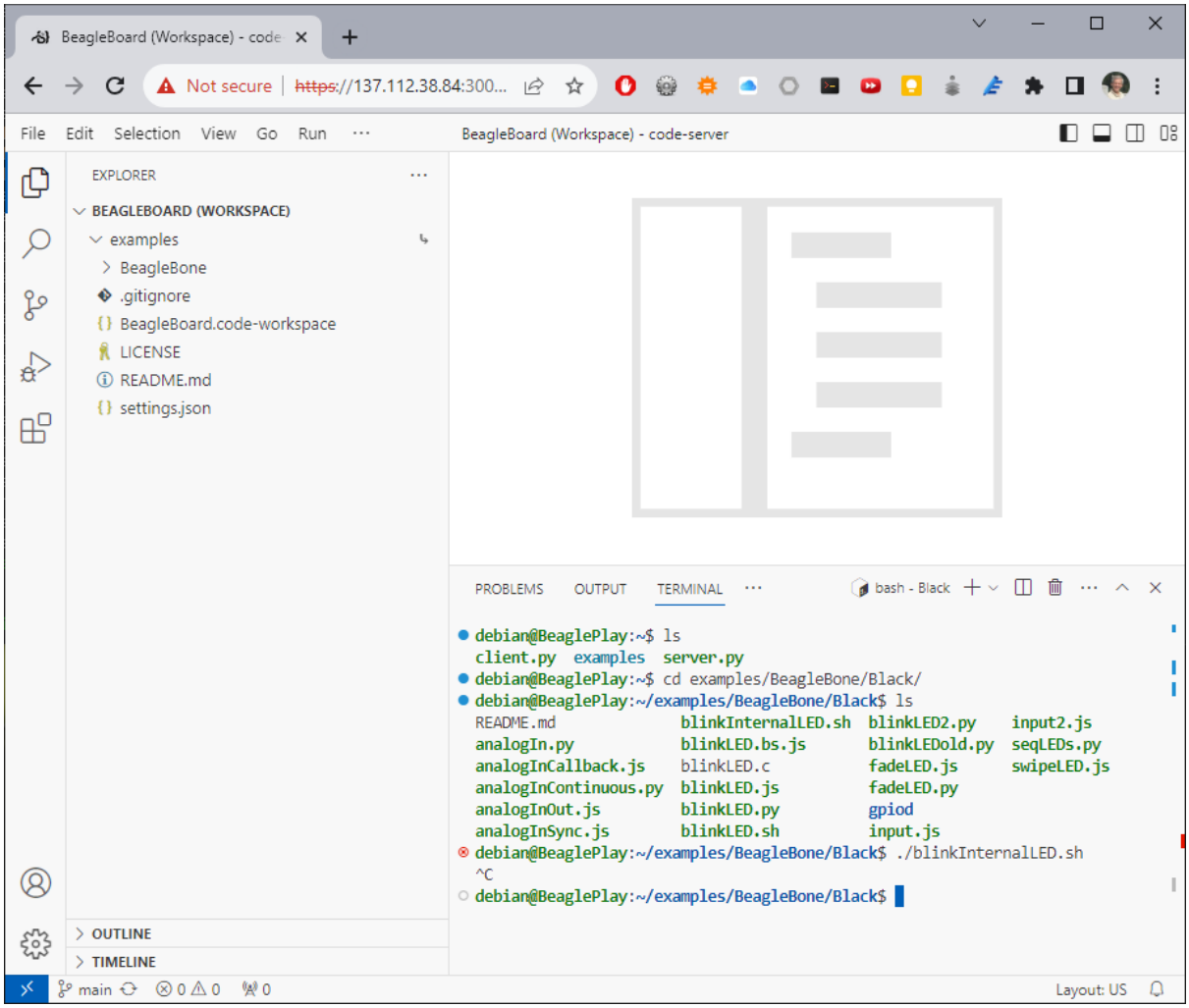
You can now enter commands and see them run as shown below.

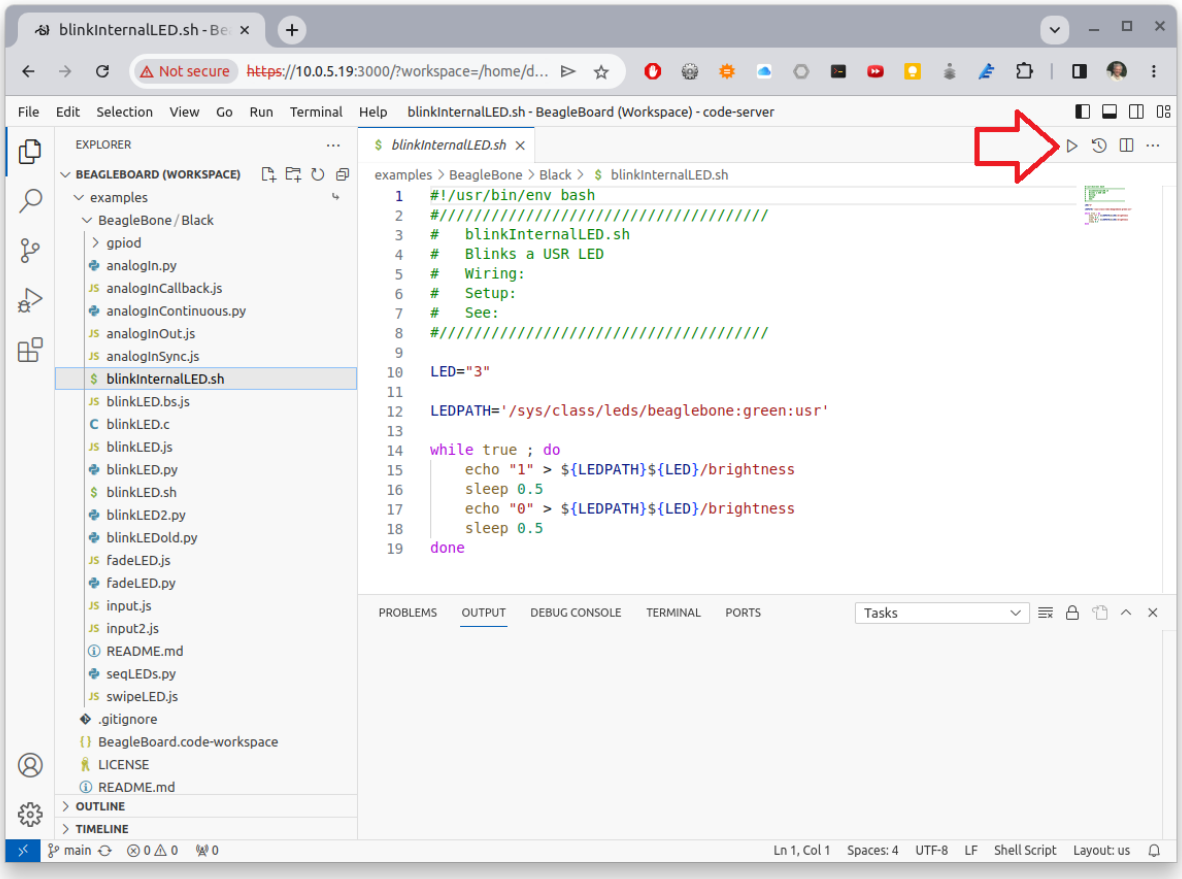
Running via the RUN button Use the file navigator on the left to navigate to `examples/BeagleBone/Black/blinkInternalLED.sh` and you will see:

This code blinks one of the USR LEDs built into the board. Click on the `RUN Code` triangle on the upper right of the screen (see red arrow) to run the code. (You could also enter `Ctrl+Alt+N`) The USR3 LED should now be blinking.

Click on the `Stop Code Run` (`Ctrl+Alt+M`) square to the right of the `Run Code` button.


BeagleBoard (Workspace) - code
Not secure
https://137.112.38.84:300...
File Edit Selection View Go Run
BeagleBoard (Workspace) - code-server
EXPLORER
BEAGLEBOARD (WORKSPACE)
examples
BeagleBone
.gitignore
BeagleBoard.code-workspace
LICENSE
README.md
settings.json
PROBLEMS OUTPUT TERMINAL
bash - Black
debian@BeaglePlay:~/examples/BeagleBone/Black$
OUTLINE
TIMELINE
main
Layout: US


BeagleBoard (Workspace) - code
Not secure
https://137.112.38.84:300...
File Edit Selection View Go Run
BeagleBoard (Workspace) - code-server
EXPLORER
BEAGLEBOARD (WORKSPACE)
examples
BeagleBone
.gitignore
BeagleBoard.code-workspace
LICENSE
README.md
settings.json
OUTLINE
TIMELINE
PROBLEMS OUTPUT TERMINAL
bash - Black
debian@BeaglePlay:~$ ls
client.py examples server.py
debian@BeaglePlay:~$ cd examples/BeagleBone/Black/
debian@BeaglePlay:~/examples/BeagleBone/Black$ ls
README.md blinkInternalLED.sh blinkLED2.py input2.js
analogIn.py blinkLED.bs.js blinkLEDold.py seqLEDs.py
analogInCallback.js blinkLED.c fadeLED.js swipeLED.js
analogInContinuous.py blinkLED.js fadeLED.py
analogInOut.py blinkLED.py gpiod
analogInSync.js blinkLED.sh input.js
debian@BeaglePlay:~/examples/BeagleBone/Black$ ./blinkInternalLED.sh
^C
debian@BeaglePlay:~/examples/BeagleBone/Black$
main
Layout: US



Time to play! Try changing the LED number (on line 10) from 3 to something else. Click the `Run Code` button (no need to save the file, autosave is on by default).

Try running `seqLEDs.py`.

Using command line

To access the command line and your host is a Mac, take the `ssh (Mac)` tab. If you are running Linux on your host, take the `ssh (Linux)` tab. Finally take the `putty (Windows)` tab for command line from Windows.

ssh (Mac)

ssh (Linux)

putty (Windows)

If you are running a Mac host, open a terminal widow and run

```
host:~$ ssh debian@192.168.6.2
```

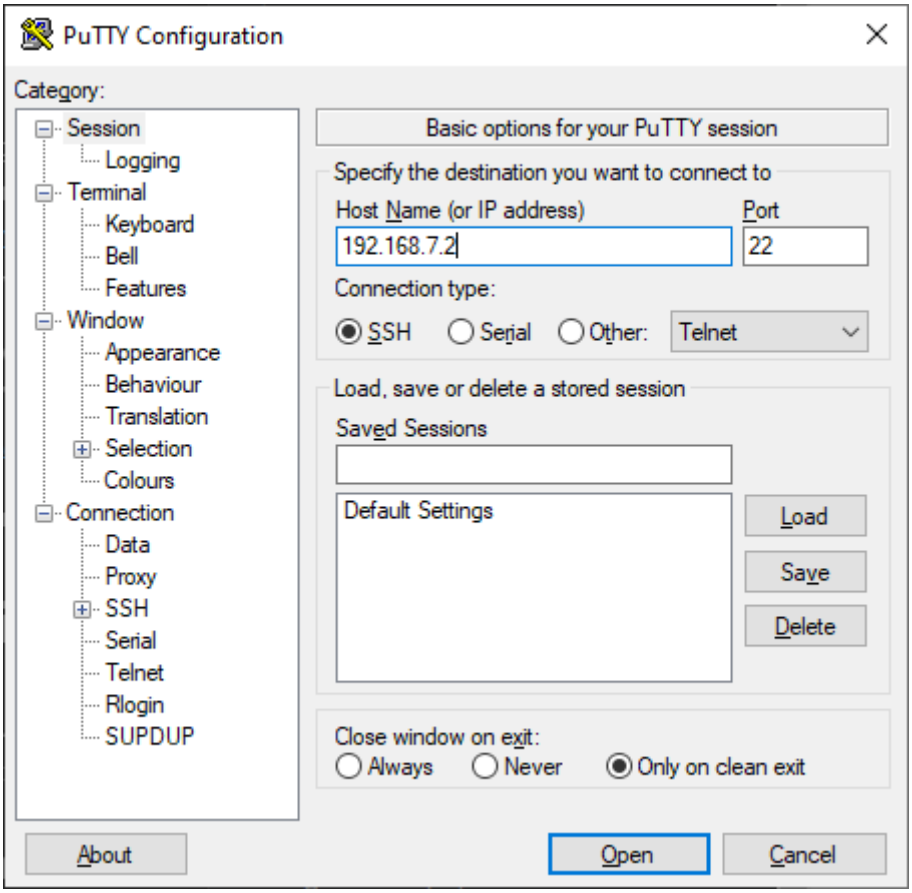
Use the password `temppwd`.

If you are running a Linux host, open a terminal widow and run

```
host:~$ ssh debian@192.168.7.2
```

Use the password `temppwd`.

If you are running Window you need to run an `ssh` client to connect to the Beagle. I suggest you use `putty`. You can download it here: https://www.putty.org/. Once installed, launch it and connect to your Beagle by sshing to `192.168.7.2`.



Login with user `debian` and password `temppwd`.

Blink an LED Once logged in the rest is easy. First:

```
bone:~$ cd ~/examples/BeagleBone/Black
bone:~$ ls
README.md              blinkInternalLED.sh  blinkLED2.py    input2.js
analogIn.py            blinkLED.bs.js       blinkLEDold.py  seqLEDs.py
analogInCallback.js    blinkLED.c           fadeLED.js      swipeLED.js
analogInContinuous.py  blinkLED.js          fadeLED.py
analogInOut.js         blinkLED.py          gpiod
analogInSync.js        blinkLED.sh          input.js
```

Here you see a list of many scripts that demo simple input/output on the Beagle. Try one that works on the internal LEDs.

```
bone:~$ cat blinkInternalLED.py
LED=”3”

LEDPATH='/sys/class/leds/beaglebone:green:usr'

while true ; do
    echo ”1” > ${LEDPATH}${LED}/brightness
    sleep 0.5
    echo ”0” > ${LEDPATH}${LED}/brightness
    sleep 0.5
done
bone:~$ ./blinkInternalLED.py
^c
```

Here you see a simple bash script that turns an LED on and off. Enter Ctrl+c to stop the script.

Blinking via Python Here's a script that sequences the LEDs on and off.

```
bone:~$ cat seqLEDs.py
import time
import os

LEDs=4
LEDPATH='/sys/class/leds/beaglebone:green:usr'

# Open a file for each LED
f = []
for i in range(LEDs):
    f.append(open(LEDPATH+str(i)+"/brightness", "w"))

# Sequence
while True:
    for i in range(LEDs):
        f[i].seek(0)
        f[i].write("1")
        time.sleep(0.25)
    for i in range(LEDs):
        f[i].seek(0)
        f[i].write("0")
        time.sleep(0.25)
bone:~$ ./seqLEDs.py
^c
```

Again, hit Ctrl+c to stop the script.

Blinking from Command Line You can control the LEDs from the command line.

```
bone:~$ cd /sys/class/leds
bone:~$ ls
beaglebone:green:usr0  beaglebone:green:usr2  mmc0::
beaglebone:green:usr1  beaglebone:green:usr3  mmc1::
```

Here you see a list of LEDs. Your list may be slightly different depending on which Beagle you are running. You can blink any of them. Let's try `usr1`.

```
bone:~$ cd beaglebone\:green\:usr1/
bone:~$ ls
brightness  device  max_brightness  power  subsystem  trigger  uevent
bone:~$ echo 1 > brightness
bone:~$ echo 0 > brightness
```

When you echo 1 into `brightness` the LED turns on. Echoing a 0 turns it off.

Blinking other LEDs You can blink the other LEDs by changing in to thier directories and doing the same. Let's blink the USR0 LED.

```
bone:~$ cd ../beaglebone\:green\:usr0/
bone:~$ echo 1 > brightness
bone:~$ echo 0 > brightness
```

Did you notice that LED `usr0` blinks on it's own in a heartbeat pattern? You can set an LED trigger. Here's what triggers you can set:

```
bone:~$ cat trigger
none usb-gadget usb-host rfkill-any rfkill-none
kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock
kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock
kbd-shiftllock kbd-shiftrlock kbd-ctrlllock kbd-ctrlrlock
timer oneshot disk-activity disk-read disk-write i
de-disk mtd nand-disk [heartbeat] backlight gpio c
pu cpu0 cpu1 cpu2 cpu3 activity default-on panic
netdev mmc0 mmc1 mmc2 phy0rx phy0tx phy0assoc phy0radio
rfkill0 gpio-0:00:link gpio-0:00:1Gbps gpio-0:00:100Mbps
gpio-0:00:10Mbps gpio-0:01:link gpio-0:01:10Mbps
bone:~$ echo none > trigger
```

Notice `[heartbeat]` is in brackets. This shows it's the current trigger. The echo changes the trigger to `none`.

Try experimenting with some of the other triggers and see if you can figure them out.

Another way to Blink an LED An interesting thing about Linux is there are often many ways to do the same thing. For example, I can think of at least five ways to blink an LED. Here's another way using the `gpiod` system.

First see where the LEDs are attached.

```
bone:~$ gpioinfo | grep -e chip -ie  usr
gpiochip0 - 32 lines:
gpiochip1 - 32 lines:
    line  21: "[usr0 led]" "beaglebone:green:usr0" output active-high [used]
    line  22: "[usr1 led]" "beaglebone:green:usr1" output active-high [used]
    line  23: "[usr2 led]" "beaglebone:green:usr2" output active-high [used]
    line  24: "[usr3 led]" "beaglebone:green:usr3" output active-high [used]
gpiochip2 - 32 lines:
gpiochip3 - 32 lines:
```

Here we asked how the LEDs are attached to the General Purpose IO (gpio) system. The answer is, (yours will be different for a different Beagle) there are four interface chips and the LEDs are attached to chip 1. You can control the gpios (and thus the LEDs) using the `gpioset` command.

```
bone:~$ gpioset --mode=time --sec=2 1 22=1
bone:~$ gpioset --mode=time --sec=2 1 22=0
```

The first command sets chip 1, line 22 (the usr1 LED) to 1 (on) for 2 seconds. The second command turns it off for 2 seconds.

Try it for the other LEDs.

Note: This may not work on all Beagles since it depends on which version of Debian you are running.

Blinking in response to a button Some Beagles have a USR button that can be used to control the LEDs. You can test the USR button with `evtest`

```
bone:~$ evtest
No device specified, trying to scan all of /dev/input/event*
Not running as root, no devices may be available.
Available devices:
/dev/input/event0:  tps65219-pwrbutton
/dev/input/event1:  gpio-keys
Select the device event number [0-1]: 1
```

We want to use `gpio-keys`, so enter `1`. Press and release the USR button and you'll see:

```
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: ”gpio-keys”
Supported events:
Event type 0 (EV_SYN)
Event type 1 (EV_KEY)
    Event code 256 (BTN_0)
Key repeat handling:
Repeat type 20 (EV_REP)
    Repeat code 0 (REP_DELAY)
    Value    250
    Repeat code 1 (REP_PERIOD)
    Value     33
Properties:
Testing ... (interrupt to exit)
Event: time 1692994988.305846, type 1 (EV_KEY), code 256 (BTN_0), value 1
Event: time 1692994988.305846, -------------- SYN_REPORT ------------
Event: time 1692994988.561786, type 1 (EV_KEY), code 256 (BTN_0), value 2
Event: time 1692994988.561786, -------------- SYN_REPORT ------------
Event: time 1692994988.601883, type 1 (EV_KEY), code 256 (BTN_0), value 2
Event: time 1692994988.601883, -------------- SYN_REPORT ------------
Event: time 1692994988.641754, type 1 (EV_KEY), code 256 (BTN_0), value 2
Event: time 1692994988.641754, -------------- SYN_REPORT ------------
Event: time 1692994988.641754, type 1 (EV_KEY), code 256 (BTN_0), value 0
Event: time 1692994988.641754, -------------- SYN_REPORT ------------
Ctrl+c
```

The following script uses evtest to wait for the USR button to be pressed and then turns on the LED.

Listing 1.1: buttonEvent.sh

```
#!/usr/bin/env bash
#/////////////////////////////////////
#        buttonEvent.sh
#        Blinks a USR LED when USR button is pressed
#       Waits for a button event
#        Wiring:
#        Setup:
#        See:        https://unix.stackexchange.com/questions/428399/how-can-i-run-a-shell-script-on-input-device-event
#/////////////////////////////////////

device='/dev/input/by-path/platform-gpio-keys-event'

LED=”3”
LEDPATH='/sys/class/leds/beaglebone:green:usr'

key_off='*value 0*'
 key_on='*value 1*'

evtest --grab ”$device” | while read line; do
    case $line in
        ($key_off) echo 0 > ${LEDPATH}${LED}/brightness ;;
        ($key_on)  echo 1 > ${LEDPATH}${LED}/brightness ;;
    esac
done
```

buttonEvent.sh

Try running it and pressing the USR button.

The next script polls the USR button and toggles the LED.

Listing 1.2: buttonLED.sh

```
#!/usr/bin/env bash
#////////////////////////////////////
#        buttonLED.sh
#        Blinks a USR LED when USR button is pressed
#       Polls the button
#        Wiring:
#        Setup:
#        See:
#////////////////////////////////////

LED=”3”

BUTTONPATH='/dev/input/by-path/platform-gpio-keys-event'
LEDPATH='/sys/class/leds/beaglebone:green:usr'

while true ; do
    # evtest returns 0 if not pressed and a non-0 value if pressed.
    evtest --query $BUTTONPATH EV_KEY BTN_0
    echo $? > ${LEDPATH}${LED}/brightness
    sleep 0.1
done
```

buttonLED.sh

1.1.2 An Introduction to Linux

This article seeks to give you some quick exploration of Linux. For a deeper training, scroll down to *Training*.

Linux is designed to make the details of the hardware it is running on not matter so much to users. It gives you a *somewhat* common experience on any hardware.

It also goes a bit further, providing some description of the harware as part of the running “file system”.

Typical Command-line Utilities

Most of what a new user experiences with Linux is the command-line.

Table 1.1: Typical Linux commands

command	function	command	function
pwd	*show current directory*	echo	*print/dump value*
cd	*change current directory*	env	*dump environment variables*
ls	*list directory contents*	export	*set environment variable*
chmod	*change file permissions*	history	*dump command history*
cp	*copy files*	man	*get help on command*
mv	*move files*	apropos	*show list of man pages*
rm	*remove files*	find	*search for files*
mkdir	*make directory*	tar	*create/extract file archives*
rmdir	*remove directory*	gzip	*compress a file*
cat	*dump file contents*	gunzip	*decompress a file*
less	*progressively dump file*	du	*show disk usage*
vi	*edit file (complex)*	df	*show disk free space*
nano	*edit file (simple)*	mount	*mount disks*
head	*trim dump to top*	tee	*write dump to file in parallel*
tail	*trim dump to bottom*	hexdump	*readable binary dumps*

Kernel.org Documentation

See https://www.kernel.org/doc.

Linux Standard Base

See https://refspecs.linuxfoundation.org/lsb.shtml.

```
$ lsb_release -a
```

Filesystem Hierarchy Standard

See https://www.pathname.com/fhs/

Kernel Application Binary Interface

See https://www.kernel.org/doc/Documentation/ABI/.

Busybox

Even though large distros like Debian and Ubuntu do not make extensive use of *busybox*, it is still very useful to learn

See http://www.busybox.net/.

Training

To continue learning more about Linux, we highly recommend https://bootlin.com/training/embedded-linux/.

1.1.3 QWIIC, STEMMA and Grove Add-ons in Linux

Note: This article is under construction.

I'm creating a place for me to start taking notes on how to load drivers for I2C devices (mostly), but also other Grove add-ons.

Todo: Create a simple drawing of BeaglePlay connecting to an external add-on with an interesting device on it.

For simplicity sake, I'll use these definitions

- **add-on**: the QWIIC, STEMMA (QT) or Grove add-on separate from your Linux computer
- **device**: the "smart" IC on the add-on to which we will interface from your Linux computer
- **board**: the Linux single board computer with the embedded interface controller you are using
- **module**: a kernel module that might contain the driver

Using I2C with Linux drivers

Linux has a ton of drivers for I2C devices. We just need a few parameters to load them.

Using a Linux I2C kernel driver module can be super simple, like in the below example for monitoring a digital light sensor.

```
cd /dev/bone/i2c/2
echo tsl2561 0x29 > new_device
watch -n0 cat "2-0029/iio:device0/in_illuminance0_input"
```

Once you issue this, your screen continuously refresh with luminance values from the add-on sensor.

In the above example, */dev/bone/i2c/2* comes from which I2C controller we are using on the board and there are specific pins on the board where you can access it. On BeagleBone boards, there is often a symbolic link to the controller based upon the cape expansion header pins being used. See *I2C* for the cape expansion header pin assignments.

tsl2561 is the name of the driver we want to load and *0x29* is the address of the device on the I2C bus. If you want to know about I2C device addresses, the Sparkfun I2C tutorial isn't a bad place to start. The *new_device* virtual file is documented in the Linux kernel documentation on instantiating I2C devices.

On the last line, watch is a program that will repeatedly run the command that follows. The *-n0* sets the refresh rate. The program cat will share the contents of the file *2-0029/iio:device0/in_illuminance0_input*.

2-0029/iio:device0/in_illuminance0_input is not a file on a disk, but output directly from the driver. The leading 2 in *2-0029* represents the I2C controller index. The *0029* represents the device I2C address. Most small sensor and actuator drivers will show up as Industrial I/O (IIO) devices. New IIO devices get incrementing indexes. In this case, *iio:device0* is the first IIO device driver loaded. Finally, *in_illuminance0_input* comes from the SYSFS application binary interface for this type of device, a light sensor. The Linux kernel ABI documentation for sysfs-bus-iio provides the definition of available data often provided by light sensor drivers.

```
What:          /sys/.../iio:deviceX/in_illuminance_input
What:          /sys/.../iio:deviceX/in_illuminance_raw
What:          /sys/.../iio:deviceX/in_illuminanceY_input
What:          /sys/.../iio:deviceX/in_illuminanceY_raw
What:          /sys/.../iio:deviceX/in_illuminanceY_mean_raw
What:          /sys/.../iio:deviceX/in_illuminance_ir_raw
What:          /sys/.../iio:deviceX/in_illuminance_clear_raw
KernelVersion:         3.4
Contact:       linux-iio@vger.kernel.org
Description:
               Illuminance measurement, units after application of scale
               and offset are lux.
```

Read further to discover how to find these bits of magic text used above.

The generic steps are fairly simple:

1. *Identify driver name and address*
2. *Ensure driver is enabled in kernel build*
3. *Identify I2C signals on board and controller in Linux*
4. *Ensure pinmux set to I2C*
5. *Ensure add-on connection is good*
6. *Issue Linux command to load driver*
7. *Identify and utilize interface provided by driver*

Driver name One resource that is very helpful is the list that Vaishnav put together for supporting Mikroelektronika Click add-ons. This list of Click add-ons with driver information can help a lot with matching a device to the driver name, device address, and kernel configuration setting.

Note: Documentation for your particular add-on might indicate a different device address than is configured on Click add-ons.

I'm not aware of a trivial way of discovering the mapping that Vaishnav created outside of looking at the kernel sources. As an example, let's look at the Grove Digital Light Sensor add-on which is documented to utilize a TSL2561.

Searching through the kernel sources, we can find the driver code at *drivers/iio/light/tsl2563.c*. There is a list of driver names in a i2c_device_id table:

```
static const struct i2c_device_id tsl2563_id[] = {
      { "tsl2560", 0 },
      { "tsl2561", 1 },
      { "tsl2562", 2 },
      { "tsl2563", 3 },
      {}
};
```

Important: Don't miss that the driver, *tsl2561* , is actually part of a superset driver, *tsl2563* . This can make things a bit trickier to find, so you have to look within the text of the driver source, not just the filenames.

Kernel configuration

I2C signals and controller

Pinmuxing

Wiring

Load driver

Interface

Finding I2C add-on modules

Note: There are some great resources out there:

- Adafruit list of I2C devices
- Sparkfun list of QWIIC devices
- Adafruit STEMMA QT introduction

Pitfalls Not all I2C devices with drivers in the Linux kernel can be loaded this way. The most common reason is that the device driver expects an interrupt signal or other GPIO along with the I2C communication. In these cases, a device tree overlay or driver modification may be necessary.

1.1.4 Linux-enabled boards

Most Beagles have on-board flash preconfigured to run Linux. These resources will get you started quickly.

- Get started at *Blink LED*.
- Learn to reset a board back to factory defaults and dive a bit deeper into the IDE at *Getting Started Guide*.
- Learn a bit about Linux at *An Introduction to Linux*.

- Learn about accessories at *Accessories*
- Learn about using 3rd party I2C add-on boards at *QWIIC, STEMMA and Grove Add-ons in Linux*.
- Learn about using mikroBUS add-on boards at *Using mikroBUS*.
- Learn about using Cape add-on boards at *Capes*.
- Read *BeagleBone Cookbook*.
- Read *PRU Cookbook*.
- Find more books at https://www.beagleboard.org/books.

1.1.5 Zephyr-enabled boards

Our Zephyr-enabled boards ship with a build of Micropython and, in the future, will also ship with a BeagleConnect Greybus node service for quick, transparent access from any BeagleConnect Greybus host enabled system.

- See *Using Micropython* to get started quickly.
- See *Using Zephyr* to learn to setup the Zephyr SDK.
- See *BeagleConnect Technology* to learn about BeagleConnect Greybus.

Todo: Make sure we have everything critical from https://beagleboard.github.io/bone101/Support/bone101/

1.2 Support

1.2.1 Getting Started Guide

Beagles are tiny computers ideal for learning and prototyping with electronics. Read the step-by-step getting started tutorial below to begin developing with your Beagle in minutes.

Update board with latest software

This step may or may not be necessary, depending on how old a software image you already have, but executing this step, the longest step, will ensure the rest will go as smooth as possible.

Download the latest software image Download the latest software image from beagleboard.org distros page. The “IoT” images provide more free disk space if you don’t need to use a graphical user interface (GUI).

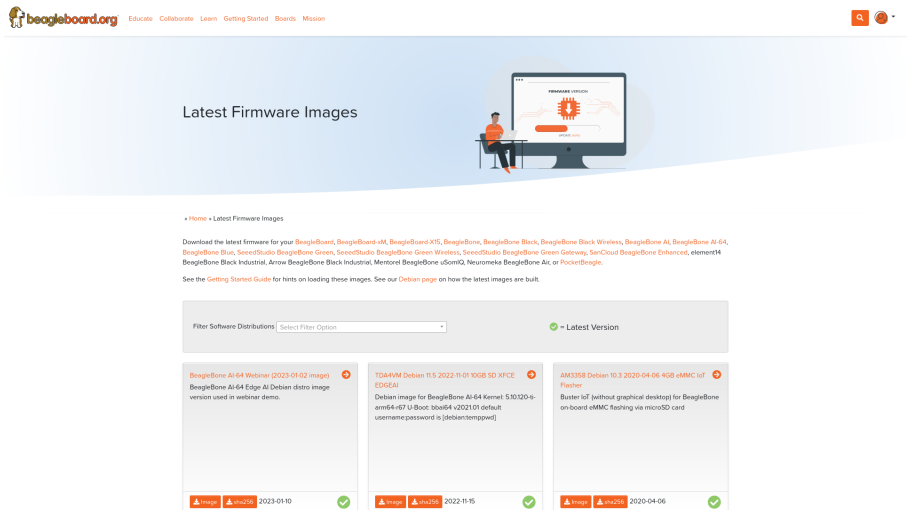
Note: Due to sizing necessities, this download may take 30 minutes or more.

---

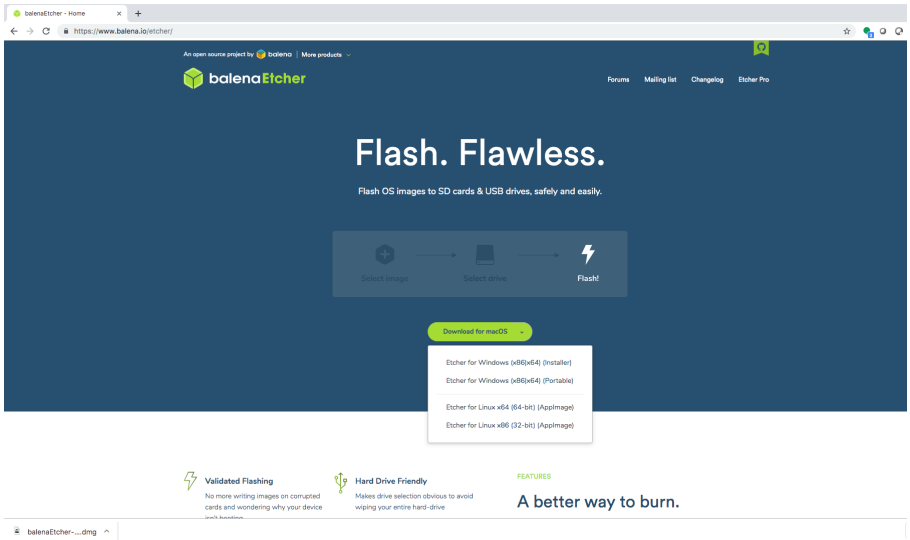
1. BeaglePlay latest image (xfce)
2. BeaglePlay latest image (home-assistant)
3. AM57xx latest image (IoT) for BeagleBone AI and BeagleBone-X15
4. AM57xx latest image (Xfce) for BeagleBone AI and BeagleBone-X15
5. BeagleBone AI-64 latest image (Minimal)
6. BeagleBone AI-64 latest image (TI EDGEAI)
7. BeagleBone AI-64 latest image (Xfce)

8. AM335x latest image for BeagleBone Black, PocketBeagle, BeagleBone Blue, etc.
9. AM335x latest image (Xfce) for BeagleBone Black, PocketBeagle, BeagleBone Blue, etc.
10. AM335x latest image (IoT) for BeagleBone Black, PocketBeagle, BeagleBone Blue, etc.
11. BeagleConnect Freedom latest image (micropython)
12. BeagleV-Ahead latest image (Ubuntu)
13. BeagleV-Ahead latest image (Yocto)

The Debian/Ubuntu distribution is provided for the boards. The file you download will have an .img.xz extension. This is a compressed sector-by-sector image of the SD card.



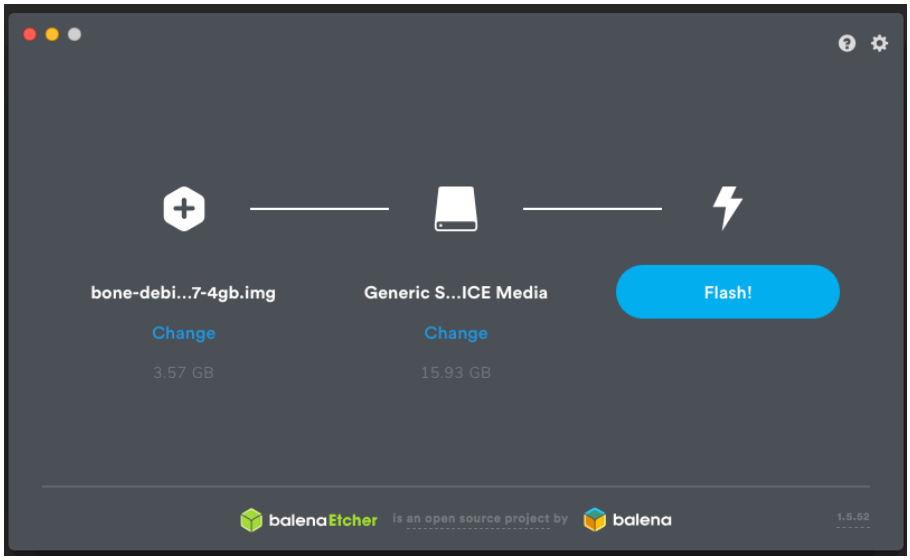
Install SD card programming utility Download and install balenaEtcher.





Connect SD card to your computer Use your computer's SD slot or a USB adapter to connect the SD card to your computer.

Write the image to your SD card Use Etcher to write the image to your SD card. Etcher will transparently decompress the image on-the-fly before writing it to the SD card.



Eject the SD card Eject the newly programmed SD card.

Boot your board off of the SD card Insert SD card into your (powered-down) board, hold down the USER/BOOT button and apply power, either by the USB cable or 5V adapter.

If using an original BeagleBone or PocketBeagle, you are done.

Note: If using BeagleBone Black, BeagleBone Blue, BeagleBone AI, BeagleBone AI-64, BeaglePlay or other board with on-board eMMC flash and you desire to write the image to your on-board eMMC, you'll need to

follow the instructions at http://elinux.org/Beagleboard:BeagleBoneBlack_Debian#Flashing_eMMC. When the flashing is complete, all 4 USRx LEDs will be steady off and possibly power down the board upon completion. This can take up to 45 minutes. Power-down your board, remove the SD card and apply power again to finish.

Start your Beagle

If any step fails, it is recommended to update to the latest software image using the instructions above.

Power and boot Most Beagles can be powered via a USB cable, providing a convenient way to provide both power to your Beagle and connectivity to your computer. Be sure the cable is of good quality and your source can provide enough power.

Alternatively, your Beagle may have a barrel jack which can take power from a wall adapter. Checkout *Power supplies* to get the correct adapter for your Beagle.

> **Danger:** Make sure to use only a 5V center positive adapter for all Beagles except BeagleBone Blue and BeagleBoard-X15 (12V).

If you are using your Beagle with an SD (microSD) card, make sure it is inserted ahead of providing power. Most Beagles include programmed on-board flash and therefore do not require an SD card to be inserted.

You'll see the power (PWR or ON) LED lit steadily. Within a minute or so, you should see the other LEDs blinking in their default configurations. Consult your boards documentation to locate these LEDs.

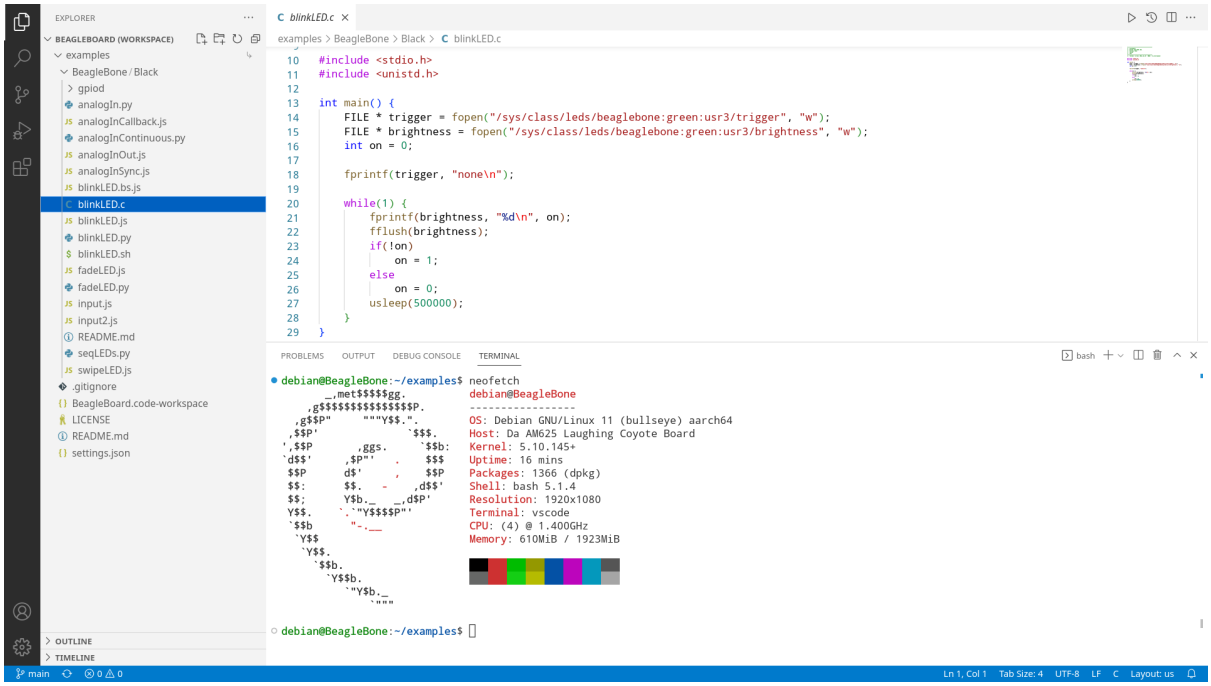
- USR0 is typically configured at boot to blink in a heartbeat pattern.
- USR1 is typically configured at boot to light during SD (microSD) card accesses.
- USR2 is typically configured at boot to light during CPU activity.
- USR3 is typically configured at boot to light during eMMC accesses.
- USR4/WIFI is typically configured at boot to light with WiFi (client) network association (Only on boards with built-in WiFi or M.2).

Enable a network connection If connected via USB, a network adapter should show up on your computer. Your Beagle should be running a DHCP server that will provide your computer with an IP address of either 192.168.7.1 or 192.168.6.1, depending on the type of USB network adapter supported by your computer's operating system. Your Beagle will reserve 192.168.7.2 or 192.168.6.2 for itself.

If your Beagle includes WiFi, an access point called "BeagleBone-XXXX" where "XXXX" varies between boards. The access point password defaults to "BeagleBone". Your Beagle should be running a DHCP server that will provide your computer with an IP address in the 192.168.8.x range and reserve 192.168.8.1 for itself.

If your Beagle is connected to your local area network (LAN) via either Ethernet or WiFi, it will utilize mDNS to broadcast itself to your computer. If your computer supports mDNS, you should see your Beagle as beaglebone.local. Non-BeagleBone boards will utilize alternate names. Multiple BeagleBone boards on the same network will add a suffix such as beaglebone-2.local.

Browse to your Beagle A web server with an Visual Studio Code (IDE) should be running on your Beagle. Point your browser to **http://192.168.7.2:3000** to begin development.



Note: Use either Firefox or Chrome (Internet Explorer will NOT work), browse to the web server running on your board. It will load a presentation showing you the capabilities of the board. Use the arrow keys on your keyboard to navigate the presentation.

The below table summarizes the typical addresses.

Link	Connection type	Operating System(s)
http://192.168.7.2	USB	Windows
http://192.168.6.2	USB	Mac OS X, Linux
http://192.168.8.1	WiFi	all
http://beaglebone.local	all	mDNS enabled
http://beaglebone-2.local	all	mDNS enabled

Troubleshooting

Do not use Internet Explorer.

Virtual machines are not recommended when using the direct USB connection. It is recommended you use only network connections to your board if you are using a virtual machine.

When using ‘ssh’ with the provided image, the username is ‘debian’ and the password is ‘temppwd’.

With the latest images, it should no longer be necessary to install drivers for your operating system to give you network-over-USB access to your Beagle. In case you are running an older image, an older operating system or need additional drivers for serial access to older boards, links to the old drivers are below.

Operating system	USB Driver	Comments
Windows (64-bit)	64-bit installer	If in doubt, try the 64-bit installer first.
Windows (32-bit)	32-bit installer	
Mac OS X	Network Serial	Install both sets of drivers.
Linux	mkudevrules.sh	Driver installation isn’t required, but you might find a few udev rules helpful.

Note: For Windows (64-bit):

1. Windows Driver Certification warning may pop up two or three times. Click “Ignore”, “Install” or “Run”.

2. To check if you're running 32 or 64-bit Windows see this.
3. On systems without the latest service release, you may get an error (0xc000007b). In that case, please perform the following and retry: https://answers.microsoft.com/en-us/windows/forum/all/windows-10-error-code-0xc000007b/02b74e7d-ce19-4ba4-90f0-e16e8d911866
4. You may need to reboot Windows.
5. These drivers have been tested to work up to Windows 10

Additional FTDI USB to serial/JTAG information and drivers are available from https://www.ftdichip.com/Drivers/VCP.htm

Additional USB to virtual Ethernet information and drivers are available from http://www.linux-usb.org/gadget/ and https://joshuawise.com/horndis

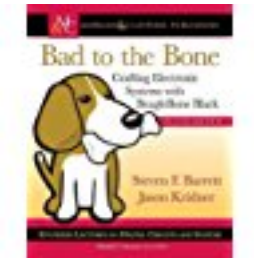
Visit https://docs.beagleboard.org/latest/intro/support/index.html for additional debugging tips.

Hardware documentation

Be sure to check check the latest hardware documentation for your board at https://docs.beagleboard.org. Detailed design materials for various boards can be found at https://git.beagleboard.org/explore/projects/topics/boards.

Books

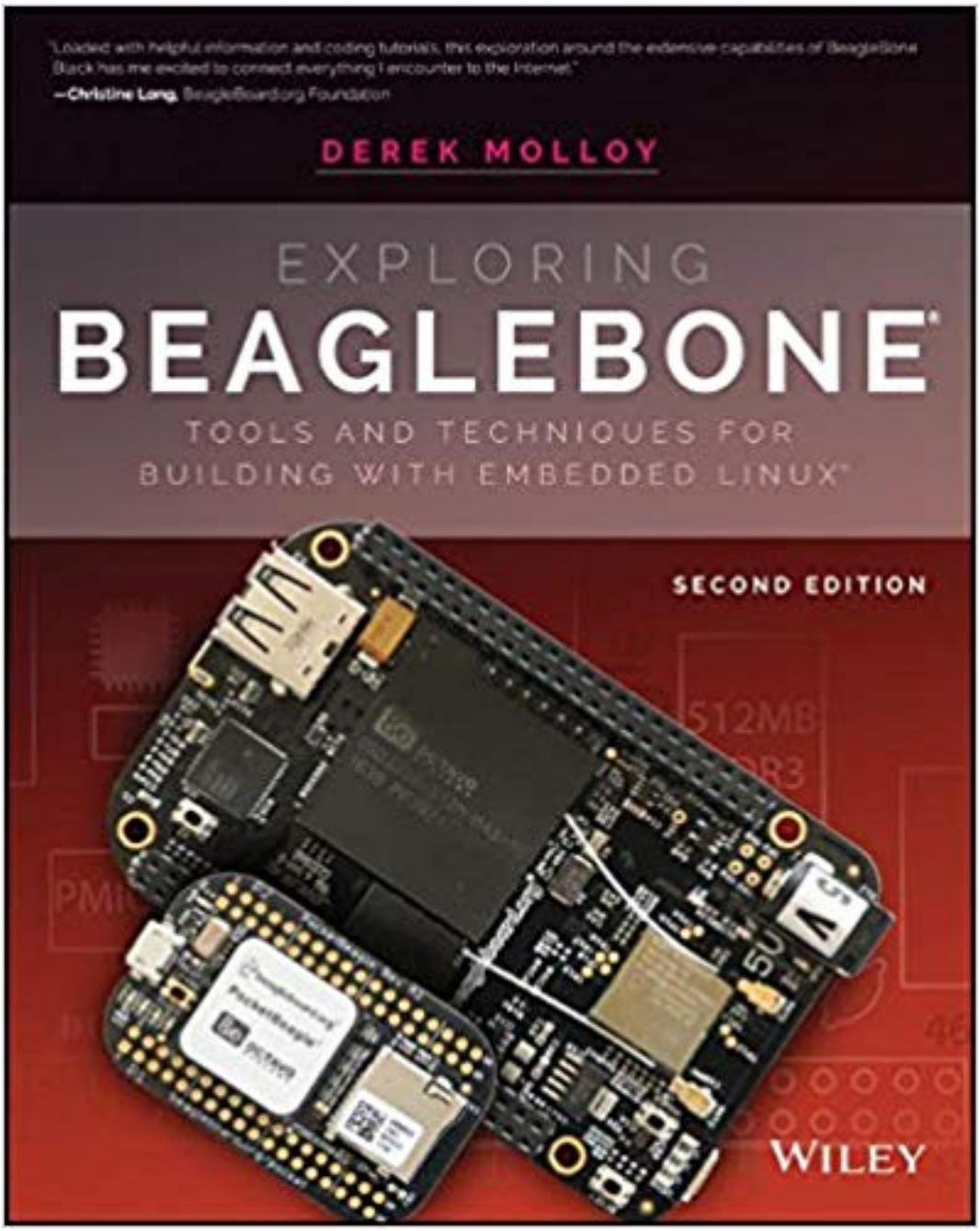
For a complete list of books on BeagleBone, see beagleboard.org/books.



Perfect for high-school seniors or freshman univerisity level text, consider using "Bad to the Bone"



A lighter treatment suitable for a bit broader audience without the backgrounders on programming and electronics, consider "BeagleBone Cookbook"



To take things to the next level of detail, consider “Exploring BeagleBone” which can be considered the missing software manual.



utilize “Embedded Linux Primer” as a companion textbook to provide a strong base on embedded Linux suitable for working with any hardware that will run Linux.

1.2.2 First, read the manual

Before reaching out for support, make sure you've gone through the process of resetting your board back to factory conditions.

For any of the Beagle Linux-based open hardware computers, visit *Getting Started Guide*.

1.2.3 Getting support

BeagleBoard.org products are open hardware designs supported via the on-line community resources. We are very confident in our community's ability to provide useful answers in a timely manner. If you don't get a productive response within 24 hours, please escalate issues to Jason Kridner (contact info available on the About Page). In case it is needed, Jason will help escalate issues to suppliers, manufacturers or others. Be sure to provide a link to your questions on the community forums as answers will be provided there.

Be sure to ask smart questions that provide the following:

- What are you trying to accomplish?
- What did you find when researching how to accomplish it?
- What are the detailed results of what you tried?
- How did these results differ from what you expected?
- What would you consider to be a success?

Important: Remember that community developers are volunteering their expertise. Respect developers time and expertise and they might be happy to share with you. If you want paid support, there are *Consulting and other resources* options for that.

Diagnostic tools

Best to be prepared with good diagnostic information to aide with support.

- Output of *beagle-version* script needed for support requests
- Beagle Tester source

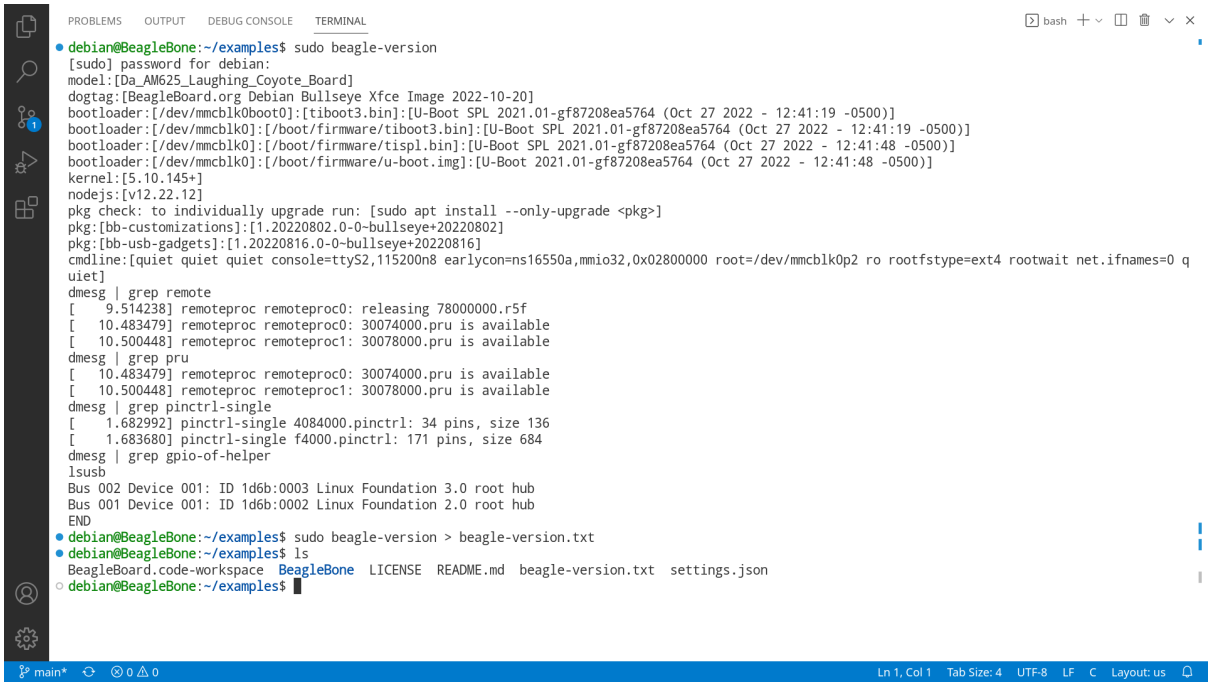
Tip: For debugging purposes you can either share the `beagle-version.txt` file you just downloaded using the steps shown in pictures above Or you can just paste the terminal output of `sudo beagle-version` to https://pastebin.com/ and send us the link.

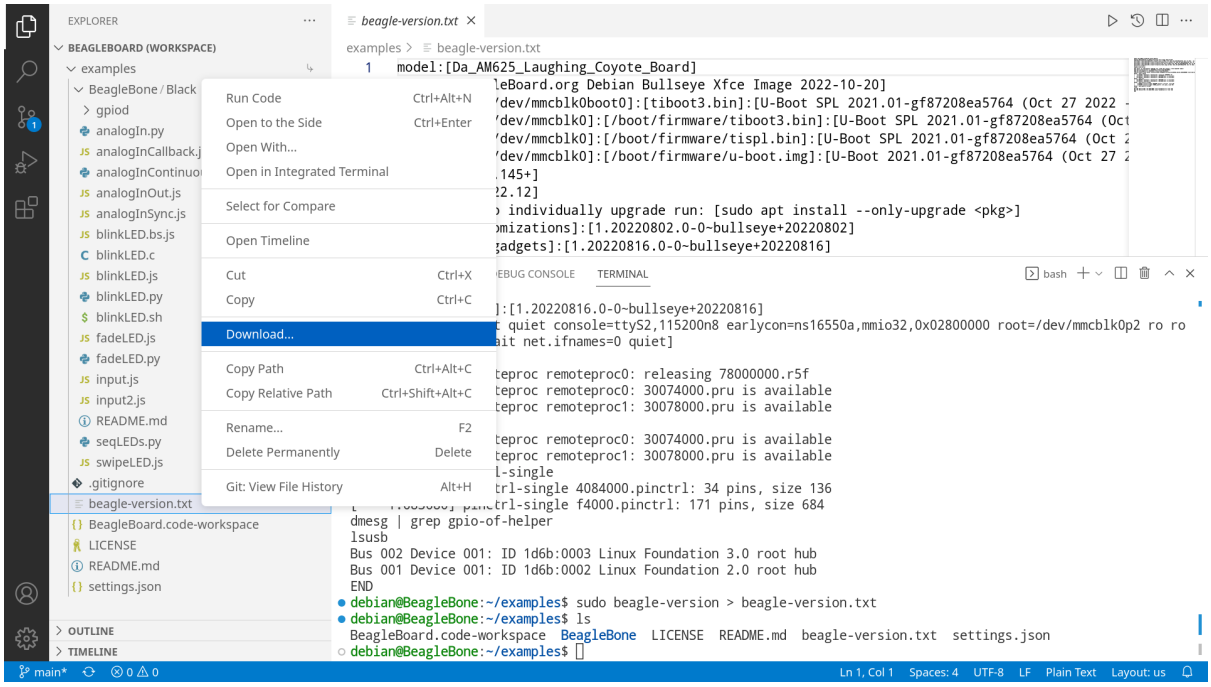
Community resources

Please execute the board diagnostics, review the hardware documentation, and consult the form and live chat for support. BeagleBoard.org is a "community" project with free support only given to those who are willing to discuss their issues openly for the benefit of the entire community.

- Frequently Asked Questions
- Forum
- Live Chat

If you need to escalate an issue already reported over 24 hours ago on the forum, please schedule a meeting to discuss it with Jason via contact information near the bottom of the about page.


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
bash
debian@BeagleBone:~/examples$ sudo beagle-version
[sudo] password for debian:
model:[Da_AM625_Laughing_Coyote_Board]
dogtag:[BeagleBoard.org Debian Bullseye Xfce Image 2022-10-20]
bootloader:[/dev/mmcblk0boot0]:[tiboot3.bin]:[U-Boot SPL 2021.01-gf87208ea5764 (Oct 27 2022 - 12:41:19 -0500)]
bootloader:[/dev/mmcblk0]:[/boot/firmware/tiboot3.bin]:[U-Boot SPL 2021.01-gf87208ea5764 (Oct 27 2022 - 12:41:19 -0500)]
bootloader:[/dev/mmcblk0]:[/boot/firmware/tispl.bin]:[U-Boot SPL 2021.01-gf87208ea5764 (Oct 27 2022 - 12:41:48 -0500)]
bootloader:[/dev/mmcblk0]:[/boot/firmware/u-boot.img]:[U-Boot 2021.01-gf87208ea5764 (Oct 27 2022 - 12:41:48 -0500)]
kernel:[5.10.145+]
nodejs:[v12.22.12]
pkg check: to individually upgrade run: [sudo apt install --only-upgrade <pkg>]
pkg:[bb-customizations]:[1.20220802.0-0~bullseye+20220802]
pkg:[bb-usb-gadgets]:[1.20220816.0-0~bullseye+20220816]
cmdline:[quiet quiet quiet console=ttyS2,115200n8 earlycon=ns16550a,mmio32,0x02800000 root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait net.ifnames=0 q
uiet]
dmesg | grep remote
[9.514238] remoteproc remoteproc0: releasing 78000000.r5f
[10.483479] remoteproc remoteproc0: 30074000.pru is available
[10.500448] remoteproc remoteproc1: 30078000.pru is available
dmesg | grep pru
[10.483479] remoteproc remoteproc0: 30074000.pru is available
[10.500448] remoteproc remoteproc1: 30078000.pru is available
dmesg | grep pinctrl-single
[1.682992] pinctrl-single 4084000.pinctrl: 34 pins, size 136
[1.683680] pinctrl-single f4000.pinctrl: 171 pins, size 684
dmesg | grep gpio-of-helper
lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
END
debian@BeagleBone:~/examples$ sudo beagle-version > beagle-version.txt
debian@BeagleBone:~/examples$ ls
BeagleBoard.code-workspace BeagleBone LICENSE README.md beagle-version.txt settings.json
debian@BeagleBone:~/examples$
main*
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
C
Layout: us


EXPLORER
BEAGLEBOARD (WORKSPACE)
examples
BeagleBone / Black
gpiod
analogIn.py
analogInCallback.j
analogInContinuo
analogInOut.js
analogInSync.js
blinkLED.bs.js
blinkLED.c
blinkLED.js
blinkLED.py
blinkLED.sh
fadeLED.js
fadeLED.py
input.js
input2.js
README.md
seqLEDs.py
swipeLED.js
.gitignore
beagle-version.txt
BeagleBoard.code-workspace
LICENSE
README.md
settings.json
OUTLINE
TIMELINE
beagle-version.txt
examples > beagle-version.txt
1 model:[Da_AM625_Laughing_Coyote_Board]
Run Code Ctrl+Alt+N
Open to the Side Ctrl+Enter
Open With...
Open in Integrated Terminal
Select for Compare
Open Timeline
Cut Ctrl+X
Copy Ctrl+C
Download...
Copy Path Ctrl+Alt+C
Copy Relative Path Ctrl+Shift+Alt+C
Rename... F2
Delete Permanently Delete
Git: View File History Alt+H
TERMINAL
bash
debian@BeagleBone:~/examples$ sudo beagle-version > beagle-version.txt
debian@BeagleBone:~/examples$ ls
BeagleBoard.code-workspace BeagleBone LICENSE README.md beagle-version.txt settings.json
debian@BeagleBone:~/examples$
main*
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text
Layout: us

Consulting and other resources

Need timely response or contract resources because you are building a product?

- Resources

Repairs

Repairs and replacements only provided on unmodified boards purchased via an authorized distributor within the first 90 days. All repaired board will have their flash reset to factory contents. For repairs and replacements, please contact `support` at BeagleBoard.org using the RMA form:

- RMA request

1.2.4 Understanding Your Beagle

Spend some time getting to know your Beagle via *An Introduction to Beagles*

1.3 Contribution

Note: This section is under developmement right now.

Important: First off, thanks for taking the time to think about contributing!

Note: For donations, see BeagleBoard.org - Donate.

The BeagleBoard.org Foundation maintains source for many open source projects.

Example projects suitable for first contributions:

- BeagleBoard project documentation
- Debian image bug repository
- Debian image builder

These guidelines are mostly suggestions, not hard-set rules. Use your best judgment, and feel free to propose changes to this document in a pull request.

1.3.1 Code of Conduct

This project and everyone participating are governed by the same code of conduct.

Note: Check out https://forum.beagleboard.org/faq as a starting place for our code of conduct.

By participating, you are expected to uphold this code. Please report unacceptable behavior to contact one of our administrators or moderators on https://forum.beagleboard.org/about.

1.3.2 Frequently Asked Questions

Please refer to the technical and contribution frequently asked questions pages before posting any of your own questions. Please feel encouraged to ask follow-up questions if any of the answers are not clear enough.

- Frequently asked questions contribution category on the BeagleBoard.org Forum

1.3.3 What should I know before I get started?

The more you know about Linux and contributing to upstream projects, the better, but this knowledge isn't strictly required. Simply reading about contributing to Linux and upstream projects can help build your vocabulary in a meaningful way to help out. Learn about the skills required for Linux contributions in the *Upstream Kernel Contributions* section.

The most useful thing to know is how to ask smart questions. Read about this in the *Getting support* section. If you ask smart questions on the issue trackers and forum, you'll be doing a lot to help us improve the designs and documentation.

1.3.4 How can I contribute?

The most obvious way to contribute is using the git.beagleboard.org Gitlab server to report bugs, suggest enhancements and providing merge requests, also called pull requests, the provide fixes to software, hardware designs and documentation.

Reading the help guide is a great way to get started using our Gitlab server.

This documentation has a number of `todo` items where help is needed that can be searched in the source. This list will show up directly in the staging documentation at https://docs.beagleboard.io/latest/intro/contribution/index.html#how-can-i-contribute.

Todo: We need a 404 document to help people handle broken links (report, find, etc.).

(The *original entry* is located in /builds/paul_buxton/docs.beagleboard.io/404.rst, line 8.)

Todo: add cape compatibility details

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/01-introduction.rst, line 65.)

Todo: This section needs more work and references to greater detail. Other boot modes are possible. Software to support USB and serial boot modes is not provided by beagleboard.org._Please contact TI for support of this feature.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/03-design-and-specifications.rst, line 239.)

Todo:

- Variable & MAC Memory
- VSYS_IO_3V3

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/04-connectors-and-pinouts.rst, line 1637.)

Todo: *Clean/Update table*

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/04-connectors-and-pinouts.rst, line 1662.)

Todo: Align with other boards and migrate away from pin usage entries for BeagleBone Black expansion

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/04-connectors-and-pinouts.rst, line 1732.)

Todo: Add BeagleBone AI-64 cape mechanical characteristics**

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/04-connectors-and-pinouts.rst, line 2102.)

Todo: IMX219 CSI sensor connection with BeagleBone® AI-64 for Edge AI

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/edge_ai_apps/getting_started.rst, line 78.)

Todo: BeagleBone® AI-64 wallpaper upon boot

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/edge_ai_apps/getting_started.rst, line 182.)

Todo: Microsoft Visual Studio Code for connecting to BeagleBone® AI-64 for Edge AI via SSH

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai-64/edge_ai_apps/getting_started.rst, line 243.)

Todo: Need info on BBAI boot mode settings

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch05.rst, line 259.)

Todo: Need info on BBAI power management

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch05.rst, line 264.)

Todo: Add WiFi/Bluetooth/Ethernet

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch05.rst, line 269.)

Todo: This text needs to go somewhere.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch05.rst, line 276.)

Todo: This table needs entries

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1490.)

Todo: Table entries needed

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1582.)

Todo: Need info on BealgeBone AI serial debug

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1596.)

Todo: Need info on BealgeBone AI USB Type-C connection

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1601.)

Todo: Need info on BealgeBone AI USB Type-A connection

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1606.)

Todo: Need info on BealgeBone AI USB Gigabit Ethernet connection

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1611.)

Todo: Need info on BealgeBone AI u.FL antenna connection

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1616.)

Todo: Need info on BealgeBone AI uSD card slot

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1621.)

Todo: Need info on BealgeBone AI uHDMI connection

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch07.rst, line 1626.)

Todo: Reference https://docs.beagleboard.org/latest/intro/support/index.html and https://beagleboard.org/resources

Related TI documentation: http://www.ti.com/tool/BEAGLE-3P-BBONE-AI

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/ai/ch11.rst, line 6.)

Todo: Make all figure references actual references

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/black/ch07.rst, line 1163.)

Todo: move accessory links to a single common document for all boards.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/black/ch07.rst, line 1184.)

Todo: We should include all support information in docs.beagleboard.org now and leave eLinux to others, freeing it as much as possible

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/black/ch07.rst, line 1194.)

Todo: We are going to work on a unified accessories page for all the boards and it should replace this.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglebone/blue/accessories.rst, line 6.)

Todo: Image with what's inside the box and a better description.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleconnect/freedom/02-quick-start.rst, line 14.)

Todo: Describe how to get a serial connection.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleconnect/freedom/02-quick-start.rst, line 55.)

Todo: Describe how to get an IEEE802.15.4g connection from BeaglePlay.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleconnect/freedom/02-quick-start.rst, line 62.)

Todo: Describe how to get to a local console and websockets console.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleconnect/freedom/02-quick-start.rst, line 73.)

Todo: Need to describe functionality of 0.2.2

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleconnect/freedom/demos-and-tutorials/using-micropython.rst, line 201.)

Todo: provide images demonstrating Jupyter Notebook visualization

(The *original entry* is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleconnect/index.rst, line 64.)

Todo: think a bit more about this section with some feedback from Cathy.

(The *original entry* is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleconnect/index.rst, line 83.)

Todo: Add specific power-up/down sequence notes here as well a highlight any limitations and known issues.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/03-design.rst, line 123.)

Todo: Add Compiler download link and env setup instructions

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/play-kernel-development.rst, line 31.)

Todo: Add steps to config kernel for play

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/play-kernel-development.rst, line 37.)

Todo: Need an image of the logo

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/using-mikrobus.rst, line 44.)

Todo: To make it stick, …

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/using-mikrobus.rst, line 142.)

Todo: Document kernel version that integrates this overlay and where to get update instructions.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/using-mikrobus.rst, line 248.)

Todo:

- How do turn off the driver?

- How do turn on spidev?
- How do I enable GPIO?
- How do a provide a manifest?

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/using-mikrobus.rst, line 293.)

Todo:

- Needs udev
- Needs live description

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/using-mikrobus.rst, line 301.)

Todo: Describe how to know it is working

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/zephyr-cc1352-development.rst, line 56.)

Todo: A big part of what is missing here is to put your BeaglePlay on the Internet such that we can download things in later steps. That has been initially brushed over.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/zephyr-cc1352-development.rst, line 67.)

Todo: Describe how to handle the serial connection

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beagleplay/demos-and-tutorials/zephyr-cc1352-development.rst, line 405.)

Todo: remove “<To-Do>” items in the table below.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/ahead/01-introduction.rst, line 55.)

Todo: add instructions for flashing in Mac.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/ahead/02-quick-start.rst, line 273.)

Todo: We need a CSI capture and DSI display demos

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/ahead/05-demos.rst, line 7.)

Todo: We need a cape compatibility layer demo

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/ahead/05-demos.rst, line 11.)

Todo: update details

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/ahead/06-support.rst, line 14.)

Todo: update details

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/ahead/06-support.rst, line 23.)

Todo: add image & information about box content.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/fire/02-quick-start.rst, line 11.)

Todo: We need a CSI capture demos

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/fire/05-demos.rst, line 6.)

Todo: We need a cape compatibility layer demo

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/fire/05-demos.rst, line 10.)

Todo: update details

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/fire/06-support.rst, line 14.)

Todo: update details

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/fire/06-support.rst, line 23.)

Todo: This is the *hard* way! Special cables and FlashPros are not required when using the firmware we initially ship on the board. This tutorial should be rescripted as how to _unbrick_ your board. Also, we have other workarounds using software and GPIOs rather than FlashPros. Let's not put this in user's face as *the* experience when it is far more painful than using the *change-gateware.sh* script and "hold BOOT button when applying power" solutions we've created!

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/fire/demos-and-tutorials/flashing-board.rst, line 6.)

Todo: Make sure people know about the alternative and we provide links to details on that before we send them down this process.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/beaglev/fire/demos-and-tutorials/mchp-fpga-tools-installation-guide.rst, line 13.)

Todo: figure out if BONE-SPI0_0 and BONE-SPI0_1 can be loaded at the same time

---

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 581.)

Todo: We need a udev rule to make sure the ADC shows up at /dev/bone/adc! There's nothing for sure that IIO devices will show up in the same place.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 750.)

Todo: I think we can also create symlinks for each channel based on which device is there, such that we can do /dev/bone/adc/Px_y

---

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 752.)

Todo: I believe a multiplexing IIO driver is the future solution

---

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 754.)

Todo: remove deep references to git trees

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 849.)

Todo: Additional quadrature encoders can be implemented in PRU firmware.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 947.)

Todo: This doesn't include any abstraction yet.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 1018.)

Todo: Describe I^2S and ALSA

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 1218.)

Todo: Document dynamic DT overlays

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 1471.)

Todo: Document dynamic pinmux control

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 1478.)

Todo: Describe how the Device Trees expose symbols for reuse across boards

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 1490.)

Todo: The steps used to verify all of these configurations is to be documented here. It will serve to document what has been tested, how to reproduce the configurations, and how to verify each major triannual release. All faults will be documented in the issue tracker.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/cape-interface-spec.rst, line 1526.)

Todo: Get OSHWA certification for all of our capes and update the documentation to reflect that

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/boards/capes/index.rst, line 9.)

Todo: Add cape examples of various sizes

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/books/beaglebone-cookbook/09capes/capes.rst, line 18.)

Todo: Update display cape example

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/books/beaglebone-cookbook/09capes/capes.rst, line 23.)

Todo: Make a mapping table for the Black

https://github.com/FalconChristmas/fpp/blob/master/src/pru/OctoscrollerV2.hp

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/books/pru-cookbook/05blocks/blocks.rst, line 1809.)

Todo: Make sure we have everything critical from https://beagleboard.github.io/bone101/Support/bone101/

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/intro/beagle101/index.rst, line 41.)

Todo: Create a simple drawing of BeaglePlay connecting to an external add-on with an interesting device on it.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/intro/beagle101/qwiic-stemma-grove-addons.rst, line 12.)

Todo: Why “device” drivers?

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/intro/contribution/linux-upstream.rst, line 148.)

Todo: Why do we need drivers?

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/intro/contribution/linux-upstream.rst, line 152.)

Todo: What do drivers look like?

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/intro/contribution/linux-upstream.rst, line 156.)

Todo: The terminology `Implicit` and `Explicit` is not accurate here.

(The original entry is located in /builds/paul_buxton/docs.beagleboard.io/intro/contribution/rst-cheat-sheet.rst, line 299.)

Reporting bugs

Start by reading the Gitlab Issues help page.

Please request an account and report any issues on the appropriate project issue tracker at https://git.beagleboard.org.

Report issues on the software images at https://git.beagleboard.org/explore/topics/distros.

Report issues on the hardware at https://git.beagleboard.org/explore/projects/topics/boards.

Suggesting enhancements

An issue doesn’t have to be something wrong, it can just be about making something better. If in doubt how to make a productive suggestion, hop on the forum and live chat groups to see what other people say. Check the current ideas that are already out there and give us your idea. Try to be constructive in your suggestion. We are a primarily a volunteer community looking to make your experience better, as those that follow you, and your suggestion could be key in that endeavor.

Where available, use the “enhancement” label on your issue to make sure we know you are looking for a future improvement, not reporting something critically wrong.

Submitting merge requests

If you want to contribute to a project, the most practical way is with a merge request. Start by creating a fork, which is your own copy of the project you can feel free to edit how you see fit. When ready, create a merge request and we’ll review your work and give comments back to you. If suitable, we’ll update the code to include your contribution!

A bit more detailed suggestions can be found in the articles linked below.

1.3.5 Articles on contribution

Contributing to docs.beagleboard.io

First off, thanks for taking the time to think about contributing!

The following is a set of guidelines for contributing to docs.beagleboard.io, which is hosted by the BeagleBoard.org Foundation at https://git.beagleboard.org/docs/docs.beagleboard.io. These are mostly guidelines, not rules. Use your best judgment, and feel free to propose changes to this document in a pull request.

Contribution

Contributions in any form are appreciated. If you want to contribute to the docs but not sure where to start then you can checkout our contribution documents to help you with your first pull request.

Code of Conduct

This project and everyone participating in it is governed by the [BeagleBoard.org Code of Conduct](https://docs.beagleboard.org/latest/intro/contribution/index.html#code-of-conduct). By participating, you are expected to uphold this code. Please report unacceptable behavior to coc@bbb.io or contact one of the administrators on https://forum.beagleboard.org.

Style and usage guidelines

If you are new to Sphinx, RST, or git then you can checkout out our [Style and usage guidelines](https://docs.beagleboard.org/latest/intro/contribution/index.html#style-and-usage-guidelines).

See more at https://docs.beagleboard.org/en/latest/intro/contribution/.

Forking the Project

You can [fork](https://docs.gitlab.com/ee/user/project/repository/forking_workflow.html) this project repository to create and submit changes to the documentation. BeagleBoard documentation is generated using [**Sphinx**](https://www.sphinx-doc.org/en/master/) documentation tool. Sphinx uses the [**reStructuredText**](https://docutils.sourceforge.io/rst.html) (.rst) markup language by default. Some files are also written in [**Markdown**](https://docs.gitlab.com/ee/user/markdown.html) (.md) format. GitLab can render previews for both RST and MD files. So if you do not want to use any external tools to edit the documentation, you can do that within your browser and use the *Preview* button in GitLab to view how the changes will actually look like.

Using A Browser

Using a browser is the easiest way to contribute to this project, if you are a beginner, or if your suggestions for changes are small. Though, some Git version control knowledge is preferrable. You can learn more about [Git and GitLab from here](https://docs.gitlab.com/ee/topics/git/).

You can create a fork of this repository to your GitLab account using the *Forks* button found on the main page of the project. This will create a stand-alone copy of the repository for you to work with.

You can now edit the files without affecting the original project repository. After making the changes, you can commit (save) them and initiate a pull-request. One of the maintainers will review your changes and if accepted, will merge your changes with the original project.

In Local Machine

Another method is to clone the forked project into local computer and make the edits from there. If you want to take advantage of the powerful editing features of a modern IDE for writing documentation, we suggest using [**Visual Studio Code**](https://code.visualstudio.com/) for it. With an IDE, you will get syntax highlighting, linting, code completion, IntelliSense, and live preview among other things. Depending on your operating system, you need to install the following softwares or packages.

- [Visual Studio Code](https://code.visualstudio.com/)
- [Python 3](https://www.python.org/downloads/)
- [Python Extension](https://marketplace.visualstudio.com/items?itemName=ms-python.python)
- [reStructuredText Extension](https://marketplace.visualstudio.com/items?itemName=lextudio.restructuredtext)

The reStructuredText extension provides rich reStructuredText language support for VS Code. It will install everything required to work with RST files. If you are prompted for installing other recommended extensions or tools such as Python packages, just continue with their installs. The extension will also install the **Esbonio** language server for showing document previews right on the IDE.

Open the terminal of your choice and run the following command to clone the forked project to your computer. The project is saved to the location where the terminal is opened from. If you want to change the location, use *cd* command to change the directory.

```
` git clone <your forked repository link> `
```

Additionally, you can use the *Clone Repository* function of VS Code and paste the URL there. Wait until all of the project files are downloaded. A folder with the name *docs.beagleboard.io* will be created. You need to enter into the directory with the *cd* command. After that, you can launch VS Code from the terminal with the following command.

```
` code . `
```

This will open the project folder as a workspace in VS Code. If you have all the necessary tools installed, you will see an *esbonio:* status button on the status bar of VS Code. You can click it to build the Sphinx project. The build log will be printed to the *OUTPUT* window of VS Code. If you run into any issues, the output log is the place you want to look at to see what went wrong.

![Successful build of BeagleBoard docs in VS Code](/_images/BeagleBoard-Docs-on-VS-Code-Sphinx-Build-Successful.png “Successful build of BeagleBoard docs in VS Code”)

VS Code is not limited to running from a folder as a workspace, or as a native application. It can run on your browser or use a remote directory as the workspace. For example, if you are on Windows 11, you can host your project on [**WSL**](https://learn.microsoft.com/en-us/windows/wsl/about) (Windows Subsystem for Linux), run an instance of VS Code in your Windows, and connect to the repository residing in the WSL. You can learn more about this from the [VS Code docs](https://code.visualstudio.com/docs/remote/wsl). This is useful if you do not want to change your main operating system to a GNU/Linux distro but still want to work with Linux-based tools.

FAQ

- [Frequently Asked Questions category on the BeagleBoard.org Forum](https://forum.beagleboard.org/c/faq/19)

Git Usage

Note: For detailed information on Git and Gitlab checkout the official Git and GitLab help page. Also, for good GitLab workflow you can checkout the Introduction to GitLab Flow (FREE) page.

These are (draft) general guidelines taken from BioPython project to be used for BeagleBoard development using git. We’re still working on the finer details.

This document is meant as an outline of the way BeagleBoard projects are developed. It should include all essential technical information as well as typical procedures and usage scenarios. It should be helpful for core developers, potential code contributors, testers and everybody interested in BeagleBoard code.

Note: This version is an unofficial draft and is subject to change.

Relevance This page is about actually using git for tracking changes.

If you have found a problem with any BeagleBoard project, and think you know how to fix it, then we suggest following the simple route of filing a bug and describe your fix. Ideally, you would upload a patch file showing the differences between the latest version of BeagleBoard project (from our repository) and your modified version. Working with the command line tools *diff* and *patch* is a very useful skill to have, and is almost a precursor to working with a version control system.

Technicalities This section describes technical introduction into git usage including required software and integration with GitLab. If you want to start contributing to BeagleBoard, you definitely need to install git and learn how to obtain a branch of the BeagleBoard project you want to contribute. If you want to share your changes easily with others, you should also sign up for a BeagleBoard GitLab account and read the corresponding section of the manual. Finally, if you are engaged in one of the collaborations on experimental BeagleBoard modules, you should look also into code review and branch merging.

Installing Git You will need to install Git on your computer. Git is available for all major operating systems. Please use the appropriate installation method as described below.

Linux Git is now packaged in all major Linux distributions, you should find it in your package manager.

Ubuntu/Debian You can install Git from the *git-core* package. e.g.,

```
sudo apt-get install git-core
```

You'll probably also want to install the following packages: *gitk*, *git-gui*, and *git-doc*

Redhat/Fedora/Mandriva git is also packaged in rpm-based linux distributions.

```
dnf install gitk
```

should do the trick for you in any recent fedora/mandriva or derivatives

Mac OS X Download the *.dmg* disk image from http://code.google.com/p/git-osx-installer/

Windows Download the official installers from Windows installers

Testing your git installation If your installation succeeded, you should be able to run

```
$ git --help
```

in a console window to obtain information on git usage. If this fails, you should refer to git documentation for troubleshooting.

Creating a GitLab account (Optional) Once you have Git installed on your machine, you can obtain the code and start developing. Since the code is hosted at GitLab, however, you may wish to take advantage of the site's offered features by signing up for a GitLab account. While a GitLab account is completely optional and not required for obtaining the BeagleBoard code or participating in development, a GitLab account will enable all other BeagleBoard developers to track (and review) your changes to the code base, and will help you track other developers' contributions. This fosters a social, collaborative environment for the BeagleBoard community.

If you don't already have a GitLab account, you can create one here. Once you have created your account, upload an SSH public key by clicking on *SSH and GPG keys <https://git.beagleboard.org/-/profile/keys>* after logging in. For more information on generating and uploading an SSH public key, see this GitLab guide.

Working with the source code In order to start working with the BeagleBoard source code, you need to obtain a local clone of our git repository. In git, this means you will in fact obtain a complete clone of our git repository along with the full version history. Thanks to compression, this is not much bigger than a single copy of the tree, but you need to accept a small overhead in terms of disk space.

There are, roughly speaking, two ways of getting the source code tree onto your machine: by simply "cloning" the repository, or by "forking" the repository on GitLab. They're not that different, in fact both will result in a

directory on your machine containing a full copy of the repository. However, if you have a GitLab account, you can make your repository a public branch of the project. If you do so, other people will be able to easily review your code, make their own branches from it or merge it back to the trunk.

Using branches on GitLab is the preferred way to work on new features for BeagleBoard, so it's useful to learn it and use it even if you think your changes are not for immediate inclusion into the main trunk of BeagleBoard. But even if you decide not to use GitLab, you can always change this later (using the .git/config file in your branch.) For simplicity, we describe these two possibilities separately.

Cloning BeagleBoard directly Getting a copy of the repository (called "cloning" in Git terminology) without GitLab account is very simple:

```
git clone https://git.beagleboard.org/docs/docs.beagleboard.io.git
```

This command creates a local copy of the entire BeagleBoard repository on your machine (your own personal copy of the official repository with its complete history). You can now make local changes and commit them to this local copy (although we advise you to use named branches for this, and keep the main branch in sync with the official BeagleBoard code).

If you want other people to see your changes, however, you must publish your repository to a public server yourself (e.g. on GitLab).

Forking BeagleBoard with your GitLab account If you are logged in to GitLab, you can go to the BeagleBoard Docs repository page:

https://git.beagleboard.org/docs/docs.beagleboard.io/-/tree/main

and click on a button named 'Fork'. This will create a fork (basically a copy) of the official BeagleBoard repository, publicly viewable on GitLab, but listed under your personal account. It should be visible under a URL that looks like this:

https://git.beagleboard.org/yourusername/docs.beagleboard.io/

Since your new BeagleBoard repository is publicly visible, it's considered good practice to change the description and homepage fields to something meaningful (i.e. different from the ones copied from the official repository).

If you haven't done so already, setup an SSH key and upload it to gitlab for authentication.

Now, assuming that you have git installed on your computer, execute the following commands locally on your machine. This "url" is given on the GitLab page for your repository (if you are logged in):

```
git clone https://git.beagleboard.org/yourusername/docs.beagleboard.io.git
```

Where *yourusername*, not surprisingly, stands for your GitLab username. You have just created a local copy of the BeagleBoard Docs repository on your machine.

You may want to also link your branch with the official distribution (see below on how to keep your copy in sync):

```
git remote add upstream https://git.beagleboard.org/docs/docs.beagleboard.io/
```

If you haven't already done so, tell git your name and the email address you are using on GitLab (so that your commits get matched up to your GitLab account). For example,

```
git config --global user.name "David Jones" config --global user.email "d.
↪jones@example.com"
```

Making changes locally Now you can make changes to your local repository - you can do this offline, and you can commit your changes as often as you like. In fact, you should commit as often as possible, because smaller commits are much better to manage and document.

First of all, create a new branch to make some changes in, and switch to it:

```
git branch demo-branch checkout demo-branch
```

To check which branch you are on, use:

```
git branch
```

Let us assume you've made changes to the file beaglebone-black/ch01.rst Try this:

```
git status
```

So commit this change you first need to explicitly add this file to your change-set:

```
git add beaglebone-black/ch01.rst
```

and now you commit:

```
git commit -m "added updates X in BeagleBone Black ch01"
```

Your commits in Git are local, i.e. they affect only your working branch on your computer, and not the whole BeagleBoard tree or even your fork on GitLab. You don't need an internet connection to commit, so you can do it very often.

Pushing changes to GitLab If you are using GitLab, and you are working on a clone of your own branch, you can very easily make your changes available for others.

Once you think your changes are stable and should be reviewed by others, you can push your changes back to the GitLab server:

```
git push origin demo-branch
```

This will not work if you have cloned directly from the official BeagleBoard branch, since only the core developers will have write access to the main repository.

Merging upstream changes We recommend that you don't actually make any changes to the **main** branch in your local repository (or your fork onGitLab). Instead, use named branches to do any of your own work. The advantage of this approach it is the trivial to pull the upstream **main** (i.e. the official BeagleBoard branch) to your repository.

Assuming you have issued this command (you only need to do this once):

```
git remote add upstream https://git.beagleboard.org/docs/docs.beagleboard.io/
```

Then all you need to do is:

```
git checkout main pull upstream main
```

Provided you never commit any change to your local **main** branch, this should always be a simple *fast forward* merge without any conflicts. You can then deal with merging the upstream changes from your local main branch into your local branches (and you can do that offline).

If you have your repository hosted online (e.g. at GitLab), then push the updated main branch there:

```
git push origin main
```

Submitting changes for inclusion in BeagleBoard If you think you changes are worth including in the main BeagleBoard distribution, then file an (enhancement) bug on our bug tracker, and include a link to your updated branch (i.e. your branch on GitLab, or another public Git server). You could also attach a patch to

the bug. If the changes are accepted, one of the BeagleBoard developers will have to check this code into our main repository.

On GitLab itself, you can inform keepers of the main branch of your changes by sending a 'pull request' from the main page of your branch. Once the file has been committed to the main branch, you may want to delete your now redundant bug fix branch on GitLab.

If other things have happened since you began your work, it may require merging when applied to the official repository's main branch. In this case we might ask you to help by rebasing your work:

```
git fetch upstream checkout demo-branch

git rebase upstream/main
```

Hopefully the only changes between your branch and the official repository's main branch are trivial and git will handle everything automatically. If not, you would have to deal with the clashes manually. If this works, you can update the pull request by replacing the existing (pre-rebase) branch:

```
git push origin demo-branch --force
```

If however the rebase does not go smoothly, give up with the following command (and hopefully the BeagleBoard developers can sort out the rebase or merge for you):

```
git rebase --abort
```

Evaluating changes Since git is a fully distributed version control system, anyone can integrate changes from other people, assuming that they are using branches derived from a common root. This is especially useful for people working on new features who want to accept contributions from other people.

This section is going to be of particular interest for the BeagleBoard core developers, or anyone accepting changes on a branch.

For example, suppose Jason has some interesting changes on his public repository:

https://git.beagleboard.org/jkridner/docs.beagleboard.io

You must tell git about this by creating a reference to this remote repository:

```
git remote add jkridner https://git.beagleboard.org/jkridner/BeagleBoard.git
```

Now we can fetch *all* of Jason's public repository with one line:

```
git fetch jkridner
```

Now we can run a diff between any of our own branches and any of Jason's branches. You can list your own branches with:

```
git branch
```

Remember the asterisk shows which branch is currently checked out.

To list the remote branches you have setup:

```
git branch -r
```

For example, to show the difference between your **main** branch and Jason's **main** branch:

```
git diff main jkridner/main
```

If you are both keeping your **main** branch in sync with the upstream BeagleBoard repository, then his **main** branch won't be very interesting. Instead, try:

```
git diff main jkridner/awesomebranch
```

You might now want to merge in (some) of Jason's changes to a new branch on your local repository. To make a copy of the branch (e.g. awesomebranch) in your local repository, type:

```
git checkout --track jkridner/awesomebranch
```

If Jason is adding more commits to his remote branch and you want to update your local copy, just do:

```
git checkout awesomebranch  # if you are not already in branch awesomebranch␣
↪pull
```

If you later want to remove the reference to this particular branch:

```
git branch -r -d jkridner/awesomebranch
Deleted remote branch jkridner/awesomebranch (#######)
```

Or, to delete the references to all of Jason's branches:

```
git remote rm jkridner

git branch -r
    upstream/main
    origin/HEAD
    origin/main
```

Alternatively, from within GitLab you can use the fork-queue to cherry pick commits from other people's forked branches. While this defaults to applying the changes to your current branch, you would typically do this using a new integration branch, then fetch it to your local machine to test everything, before merging it to your main branch.

Committing changes to main branch This section is intended for BeagleBoard developers, who are allowed to commit changes to the BeagleBoard main "official" branch. It describes the typical activities, such as merging contributed code changes both from git branches and patch files.

Prerequisites Currently, the main BeagleBoard branch is hosted on GitLab. In order to make changes to the main branch you need a GitLab account and you need to be added as a collaborator/Maintainer to the BeagleBoard account. This needs to be done only once. If you have a GitLab account, but you are not yet a collaborator/Maintainer and you think you should be ask Jason to be added (this is meant for regular contributors, so in case you have only a single change to make, please consider submitting your changes through one of developers).

Once you are a collaborator/Maintainer, you can pull BeagleBoard official branch using the private url. If you want to make a new repository (linked to the main branch), you can just clone it:

```
git clone https://git.beagleboard.org/lorforlinux/docs.beagleboard.io.git
```

It creates a new directory "BeagleBoard" with a local copy of the official branch. It also sets the "origin" to the GitLab copy This is the recommended way (at least for the beginning) as it minimizes the risk of accidentally pushing changes to the official GitLab branch.

Alternatively, if you already have a working git repo (containing your branch and your own changes), you can add a link to the official branch with the git "remote command"... but we'll not cover that here.

In the following sections, we assume you have followed the recommended scenario and you have the following entries in your .git/config file:

```
[remote "origin"]
    url = https://git.beagleboard.org/lorforlinux/docs.beagleboard.io.git
```

(continues on next page)

(continued from previous page)

```
[branch "main"]
    remote = origin
```

Committing a patch If you are committing from a patch, it's also quite easy. First make sure you are up to date with official branch:

```
git checkout main pull origin
```

Then do your changes, i.e. apply the patch:

```
patch -r someones_cool_feature.diff
```

If you see that there were some files added to the tree, please add them to git:

```
git add beaglebone-black/some_new_file
```

Then make a commit (after adding files):

```
git commit -a -m "committed a patch from a kind contributor adding feature X"
```

After your changes are committed, you can push toGitLab:

```
git push origin
```

Tagging the official branch If you want to put tag on the current BeagleBoard official branch (this is usually done to mark a new release), you need to follow these steps:

First make sure you are up to date with official branch:

```
git checkout main pull origin
```

Then add the actual tag:

```
git tag new_release
```

And push it to GitLab:

```
git push --tags origin main
```

Additional Resources There are a lot of different nice guides to using Git on the web:

- Understanding Git Conceptually
- git ready: git tips
- https://web.archive.org/web/20121115132047/http://cheat.errtheblog.com/s/git
- https://docs.scipy.org/doc/numpy-1.15.1/dev/gitwash/development_workflow.html Numpy is also evaluating git
- https://github.github.com/training-kit/downloads/github-git-cheat-sheet
- https://skills.github.com/
- Pro Git

Documentation Style Guide

Note: This is currently a work-in-progress placeholder for some notes on how to style the BeagleBoard Documentation Project.

See the Zephyr Project Documentation Guidelines as a starting point.

ReStructuredText Cheat Sheet

BeagleBoard.org docs site uses ReStructuredText (rst) which is a file format[1] for textual data used primarily in the Python programming language community for technical documentation. It is part of the Docutils project of the Python Doc-SIG, aimed at creating a set of tools for Python similar to Javadoc for Java or Plain Old Documentation for Perl. If you are new with rst you may go through this rst cheat sheet[2][3][4] chapter to gain enough skills to edit and update any page on the BeagleBoard.org docs site. some things you should keep in mind while working with rst,

1. like Python, RST syntax is sensitive to indentation !
2. RST requires blank lines between paragraphs

Tip: Why not use Markdown for documentation? because reST stands out against Markdown as,

1. It's more fully-featured.
2. It's much more standardized and uniform.
3. It has built-in support for extensions.

For more detailed comparison you can checkout this article on reStructuredText vs. Markdown for technical documentation

Text formatting With asterisk you can format the text as italic & bold,

1. Single asterisk (`*`) like `*emphasis*` gives you *italic text*
2. Double asterisk (`**`) like `**strong emphasis**` gives you **bold text**

With backquote character (‘) you can format the text as link & inline literal.

1. See *Links* section on how single backquote can be used to create a link like this.
2. With double back quotes before and after text you can easily create `inline lierals`.

Note: backquote can be found below escape key on most keyboards.

Headings For each document we divide sections with headings and in ReStructuredText we can use matching overline and underline to indicate a heading.

1. Document heading (H1) use `#`.
2. First heading (H2) use `*`.
3. Second heading (H3) use `=`.

[1] reStructuredText wiki page
[2] Sphinx and RST syntax guide (0.9.3)
[3] Quick reStructuredText (sourceforge)
[4] A two-page cheatsheet for restructured text

4. Third heading (H4) use −.
5. Fourth heading (H5) use ~.

Note: You can include only one (H1) # in a single documentation page.

Make sure the length of your heading symbol is at least (or more) the at least of the heading text, for example:

```
incorrect H1
##### ①

correct H1
############ ②
```

① Length of heading symbol # is smaller than the content above.

② Shows the correct way of setting the document title (H1) with #.

Code For adding a code snippet you can use tab indentation to start. For more refined code snippet display we have the `code-block` and `literalinclude` directives as shown below.

Indentation This the simplest way of adding code snippet in ReStructuredText.

Example

```
This is python code:: ①
    ②
    import numpy as np ③
    import math
```

① Provide title of your code snippet and add `::` after the text.

② Empty line after the title is required for this to work.

③ Start adding your code.

Output This is python code:

```
import numpy as np
import math
```

Code block Simple indentation only supports python program highlighting but, with code block you can specify which language is your code written in. `code-block` also provides better readability and line numbers support you can useas shown below.

Example

```
.. code-block:: python ①
    :linenos: ②

    import numpy as np ③
    import math
```

① Start with adding `.. code-block::` and then add language of code like python, bash, javascript, etc.

② Optionally, you can enable line numbers for your code.

③ Start adding your code.

Output

```
import numpy as np
import math
```

Literal include To include the entire code or a code snippet from a program file you can use this directive.

Example

```
.. literalinclude:: filename.cpp ①
    :caption: Example C++ file ②
    :linenos: ③
    :language: C++ ④
    :lines: 2, 4-7 ⑤
    :lineno-start: 113 ⑥
```

① Provide the code file destination.

② Provide caption for the code.

③ Enable line numbers.

④ Set programming language.

⑤ Cherry pick some lines from a big program file.

⑥ Instead of starting line number from 1 start it with some other number. It's useful when you use :lines:, :start-after:, and :end-before:.

Annotations We have a plug-in installed that enables annotated code blocks. Below is an example.

Example

```
.. callout:: ①

    .. code-block:: python ?

        import numpy as np # <1> ③
        import math # <2>

    .. annotations:: ④

        <1> Comment #1 ⑤

        <2> Comment #2

.. annotations::

    ① Indent everything under a `callout`

    ② Create a normal block for what you want to annotate

    ③ Add ``<number>`` everywhere you want to annotate. Put it under a␣
↪comment block if you want the code to run when copied directly.

    ④ Create an `annotations` block to hold your callout comments

    ⑤ Create an entry, separating each with a blank line and prefixing them␣
↪with ``<number>``
```

Output

```
import numpy as np # ①
import math # ②
```

① Comment #1

② Comment #2

Important: In the example, I inserted the invisible UTF character U+FEFF after the opening < to avoid it being interpreted as a callout symbol. Be sure to remove that character if you attempt to copy-and-paste the example.

Links We have three types of links to use in sphinx,

1. External links (http(s) links).
2. Implicit links to title (within same rst file).
3. Explicit links (labels that can be used anywhere in the project).

External links For a simple link to a site the format is

```
`<www.beagleboard.org>`_
```

this will be rendered as www.beagleboard.org.

You can also include a label to the link as shown below.

```
`BeagleBoard.org <www.beagleboard.org>`_
```

this will be rendered as BeagleBoard.org.

Implicit Links These are basically the headings inside the rst page which can be used as a link to that section within document.

```
`Links`_
```

when rendered it becomes *Links*

Explicit link

Todo: The terminology `Implicit` and `Explicit` is not accurate here.

These are special links you can assign to a specific part of the document and reference anywhere in the project unlike implicit links which can be used only within the document they are defined. On top of each page you'll see some text like `.. _rst-cheat-sheet:` is used to create a label for this chapter. These are called the explicit links amd you can reference these using `ref:`.

Note: This can be used inside or outside of the document and the rendered link will take you directly to that specific section.

```
:ref:`rst-cheat-sheet`
```

When rendered it becomes *ReStructuredText Cheat Sheet*.

YouTube Videos This section shows you the typical way of adding a YouTube video to docs.BeagleBoard.org in a way that you see on page playable embedded YouTube video when you look at HTML version of the docs and only a clicable thumnail linked to the YouTube video when you see the PDF.

```
.. only:: latex

    .. image:: https://img.youtube.com/vi/<YouTube_video_ID>/maxresdefault.
↪jpg ①
        :alt: BeagleConnect unboxing YouTube video
        :width: 1280
        :target: https://www.youtube.com/watch?v=<YouTube_video_ID> ②

.. only:: html

    .. raw:: html

        <iframe style="display: block; margin: auto;" width="1280" height=
↪"720" style="align:center"
        src="https://www.youtube.com/embed/<YouTube_video_ID>" ③
        title="YouTube video player"
        frameborder="0"
        allow="accelerometer; autoplay; clipboard-write; encrypted-media;␣
↪gyroscope; picture-in-picture; web-share"
        allowfullscreen>
    </iframe>
```

① ② ③ Here you have to replace the <YouTube_video_ID> with your actual youtube ID.

More

footnotes

Upstream Kernel Contributions

Note: For detailed information on Kernel Developmement checkout the official kernel.org kernel docs.

For a person or company who wishes to submit a change to the Linux kernel, the process can sometimes be daunting if you're not familiar with "the system." This text is a collection of suggestions which can help you get started and greatly increase the chances of your change being accepted.

Note: This version is an unofficial draft and is subject to change.

Pre-requisites The following are the skills that are needed before you actually start to contribute to the linux kernel:

- *More Git!*
- *C-Programming*
- *Cross-arch Development*
- *Basics of embedded buses (I2C, UART, SPI, etc.)*
- *Device Drivers in Embedded Systems*
- *Device Trees*

For more guidance, check out the *Additional Resources*.

More Git! It is highly recommended that you go through *Git Usage* before starting to read and follow these guidelines. You will need to have a proper git setup on your computer in order to effectively follow these steps.

Creating your first patch When you first enter the world of Linux Kernel development from a background in contributing over gitlab or github, the terminologies slightly change.

Your Pull Requests (PRs) now become Patches or Patch Series. You no longer just go to some website and click on a "Create Pull Request" button. Whatever code/changes you want to add will have to be sent as patches via emails.

As an example, let's consider a commit to add the git section to these docs. I stage these changes first using `git add -p`.

```
diff --git a/contribution/contribute.rst b/contribution/contribute.rst
index def100b..0af08c5 100644
--- a/contribution/contribute.rst
+++ b/contribution/contribute.rst
```

Then, commit the above changes.

Note: Don't forget to make your commit message descriptive of the feature you are adding or the work that you have done in that commit. The commit has to be self explanatory in itself. Link any references if you have used and paste any logs to prove your code works or if there is a fix.

```
git commit -vs

[linux-contrib 3bc0821] contribute.rst: Add git section
 1 file changed, 27 insertions(+), 1 deletion(-)
```

Now, let's say we want to send this new feature to upstream kernel. You then have to create a patch file using the following command:

```
git format-patch -1 HEAD

0001-contribute.rst-Add-git-section.patch
```

This will generate one file that is generally referred to as the patch file. This is what you will now be sending upstream in order to get your patch merged. But wait, there are a few more things we need to setup for sending a patch via e-mail. That is, of course your email!

For configuring your email ID for sending patches refer to this excellent stackoverflow thread, configure git-send-email.

Finally, after you have configured you email properly, you can send out a patch using:

```
git send-email 0001-contribute.rst-Add-git-section.patch
```

replacing of course the above patchfile name with whatever was your own patch. This command will then ask you `To whom should the emails be sent (if anyone)?` Here, you have to write the email address of the list you want to send out the patch to.

`git send-email` also has command line options like `--to` and `--cc` that you can also use to add more email addresses of whoever you want to keep in CC. Generally it is a good idea to keep yourself in CC.

C-Programming It is highly recommended that you have proficiency in C-Programming, because well the kernel is mostly written in C! For starters, you can go through Dennis Ritchie's C Programming book to understand the language and also solve the exercises given there for getting hands on.

Cross-arch Development While working with the kernel, you'll most likely not be compiling it on the machine that you intend to actually boot it on. For example if you are compiling the Kernel for BeageBone Black it's probably not ideal for you to actually clone the entire kernel on BeagleBone Black and then compile it there.

What you'd do instead is pick a much powerful machine like a Desktop PC or laptop and then use cross arch compilers like the arm-gcc for instance to compile the kernel for your target device.

Basics of embedded buses (I2C, UART, SPI, etc.) In the world of embedded, you often need to communicate with peripherals over very low level protocols. To name a few, I2C, UART, SPI, etc. are all serial protocols used to communicate with a variety of devices and peripherals.

It's recommended to understand at least the basics of each of the protocol so you know what's actually going on when you write for instance an I2C or SPI driver to communicate with let's say a sensor.

Device Drivers in Embedded Systems I used the term "Drivers" in the above section, but what does it really mean?

Todo: Why "device" drivers?

Todo: Why do we need drivers?

Todo: What do drivers look like?

Device Trees We just learned about drivers, and it's time that once you have written a driver in the kernel, you obviously want it to work! So how do we really tell the kernel which drivers to load? How do we, at boot time, instruct which devices are present on the board you are booting on?

The kernel does not contain the description of the hardware, it is located in a separate binary: the device tree blob.

What is a Device Tree?

A device tree is used to describe system hardware. A boot program loads a device tree into a client program's memory and passes a pointer to the device tree to the client.

A device tree is a tree data structure with nodes that describe the physical devices in a system.

Additional Resources

1. Device Trees for Dummies PDF
2. What are Device Drivers
3. Submitting your patches upstream

Contributors

This is a list of people who have made contributions to BeagleBoard.org unified documentation project.

People are listed alphabetically, as verified with Unix sort:

```
$ grep "^- " CONTRIB.rst | LC_ALL=C sort -u -c -f
```

This is certainly not comprehensive, and if you've been overlooked (sorry!), please open an issue on GitLab or mention it on the forum.

- Andrei Aldea
- Ayush Singh

- Cathy Wicks
- Deepak Khatri
- Dhruva Gole
- Gerald Coley
- Harshil Bhatt
- Jason Kridner
- Jian De
- Joshua Neal
- Kai Yamada
- Krishna Narayanan
- Mark A. Yoder
- Robert Nelson
- Robert P J Day
- Sabeeh Khan
- Seth N

Chapter 2

BeaglePlay

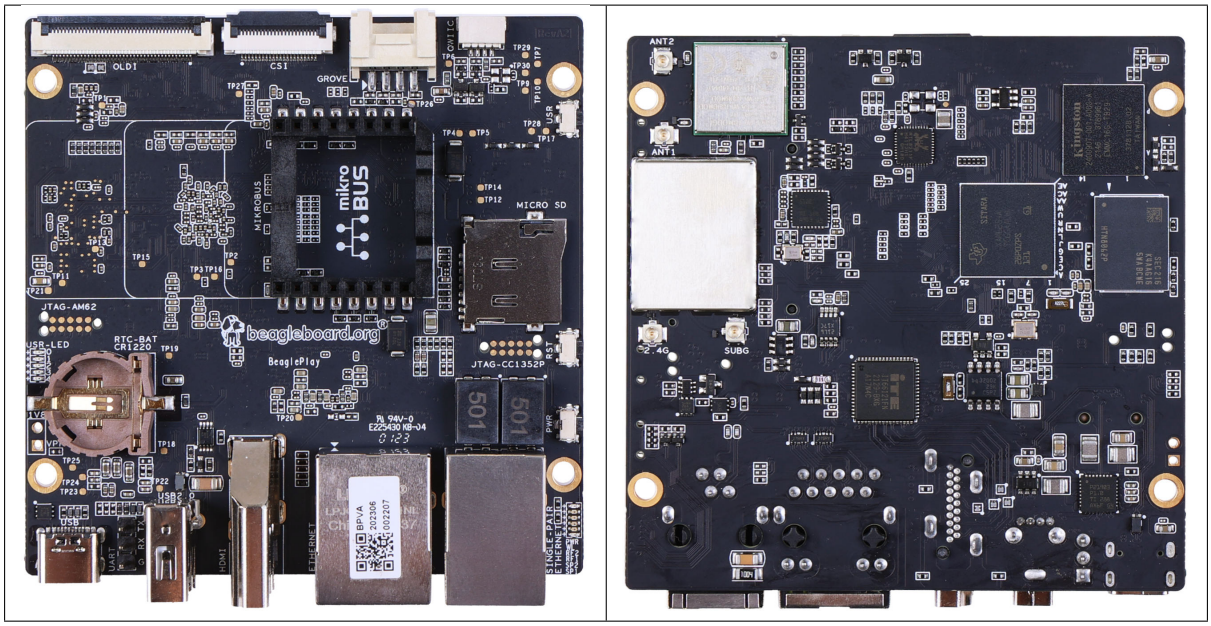
Important: This is a work in progress, for latest documentation please visit https://docs.beagleboard.org/latest/

BeaglePlay is an open-source single board computer based on the Texas Instruments AM6254 quad-core Cortex-A53 Arm SoC designed to simplify the process of adding sensors, actuators, indicators, human interfaces, and connectivity to a reliable embedded system.



License Terms

- This documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page
- For export, emissions and other compliance, see *Support*



2.1 Introduction

BeaglePlay is an open-source single board computer designed to simplify the process of adding sensors, actuators, indicators, human interfaces, and connectivity to a reliable embedded system. It features a powerful 64-bit, quad-core processor and innovative connectivity options, including WiFi, Gigabit Ethernet, sub-GHz wireless, and single-pair Ethernet with power-over-data-line. With compatibility with 1,000s of off-the-shelf add-ons and a customized Debian Linux image, BeaglePlay makes expansion and customization easy. It also includes ribbon-cable connections for cameras and touch-screen displays, and a socket for a battery-backed real-time-clock, making it ideal for human-machine interface designs. With its competitive price and user-friendly design, we expect BeaglePlay to provide you with a positive development experience. Some of the real world applications for BeaglePlay include:

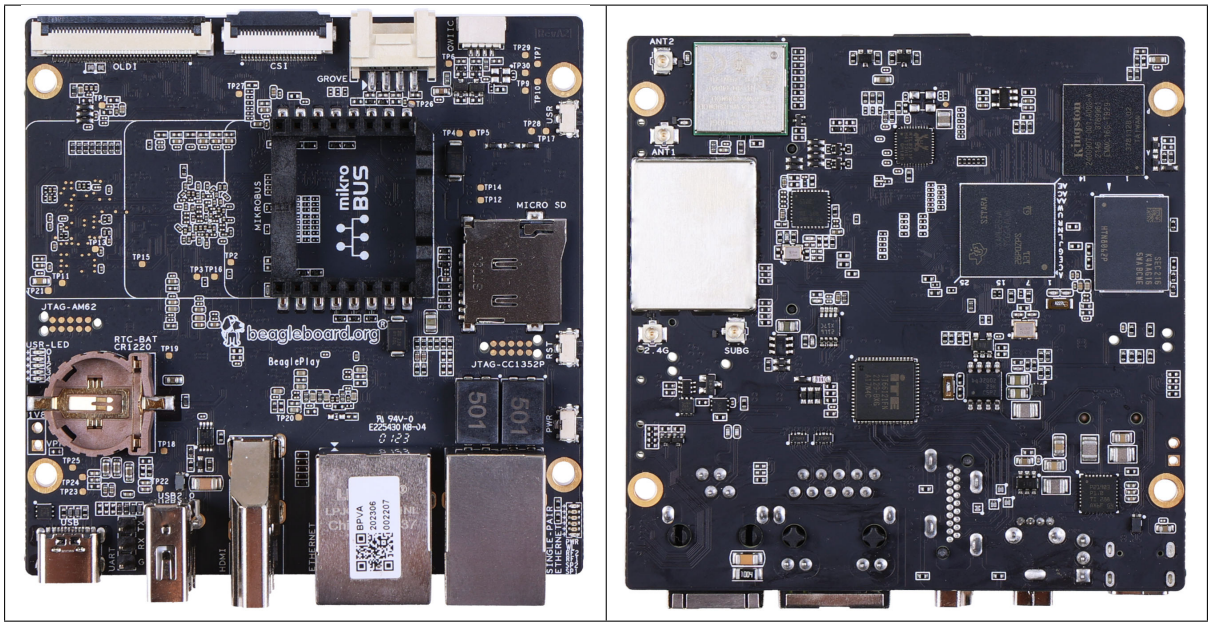
- Building/industrial automation gateways
- Digital signage
- Human Machine Interface (HMI)
- BeagleConnect sensor gateways

Contributors

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

Note: Make sure to read and accept all the terms & condition provided in the *Terms & Conditions* page.

Use of either the boards or the design materials constitutes agreement to the T&C including any modifications done to the hardware or software solutions provided by beagleboard.org foundation.



2.1.1 Detailed overview

BeaglePlay is built around Texas Instruments AM62x Sitara™ Processors which is a Quad-Core Arm® Cortex®-A53 Human-machine-interaction SoC. It comes with 2GB DDR4 RAM, 16GB eMMC storage, Full size HDMI, USB-A host port, USB-C power & connectivity port, serial debug interface, and much more.

Table 2.1: BeaglePlay features

Feature	Description
Processor	TI AM6254 (multicore A53s with R5, M4s and PRUs)
PMIC	TPS6521901
Memory	2GB DDR4
Storage	16GB eMMC
WiFi	• PHY: WL1807MOD (roadmap to next-gen TI CC33XX WiFi 6 & BLE) • Antennas: 2.4GHz & 5GHz
BLE/SubG	• CC1352P7 M4+M0 with BeagleConnect firmware • BeagleConnect Wireless enabled • Antennas: 2.4GHz & SubG IEEE802.15.4 software defined radio (SDR)
Ethernet	• PHY: Realtek RTL8211F-VD-CG Gigabit Ethernet phy • Connector: integrated magnetics RJ-45
Single-pair Ethernet	• BeagleConnect Wired enabled • PHY: DP83TD510E 10Mbit 10BASE-T1L single-pair Ethernet phy • Connector: RJ-11 jack • Power (PoDL): Input: N/A (protection to 12V), Output: 5V @ 250mA
USB type-C	• PD/CC: None, HS shorted to both sides • Power: Input: 5V @ 3A, Output: N/A (USB-C DRP Not supported)
HDMI	• Transmitter: IT66121 • Connector: full-size
Other connectors	• microSD • USB 2.0 type-A (480Mbit) • mikroBUS connector (I2C/UART/SPI/MCAN/MCASP/PWM/GPIO) • Grove connector (I2C/UART/ADC/PWM/GPIO) • QWIIC connector (I2C) • CSI connector compatible with BeagleBone AI-64, Raspberry Pi Zero / CM4 (22-pin) • OLDI connector (40-pin)

AM6254 SoC

The low-cost Texas Instruments AM625 family of application processors are built for Linux® application development. With scalable Arm® Cortex®-A53 performance and embedded features, such as: dual-display support and 3D graphics acceleration, along with an extensive set of peripherals that make the AM62x device well-suited for a broad range of industrial and automotive applications while offering intelligent features and optimized power architecture as well.

Some of these applications include:

- Industrial HMI
- EV charging stations

- Touchless building access
- Driver monitoring systems

AM625 processors are industrial-grade in the 13 x 13 mm package (ALW) and can meet the AEC-Q100 automotive standard in the 17.2 x 17.2 mm package (AMC). Industrial and Automotive functional safety requirements can be addressed using the integrated Cortex-M4F core and dedicated peripherals, which can all be isolated from the rest of the AM62x processor.

Tip: For more details checkout https://www.ti.com/product/AM625

The 3-port Gigabit Ethernet switch has one internal port and two external ports with Time-Sensitive Networking (TSN) support. An additional PRU module on the device enables real-time I/O capability for customer's own use cases. In addition, the extensive set of peripherals included in AM62x enables system-level connectivity, such as: USB, MMC/SD, CSI Camera interface, OSPI, CAN-FD and GPMC for parallel host interface to an external ASIC/FPGA. The AM62x device also employs advanced power management support for portable and power-sensitive applications.

Board components location

This section describes the key components on the board, their location and function.

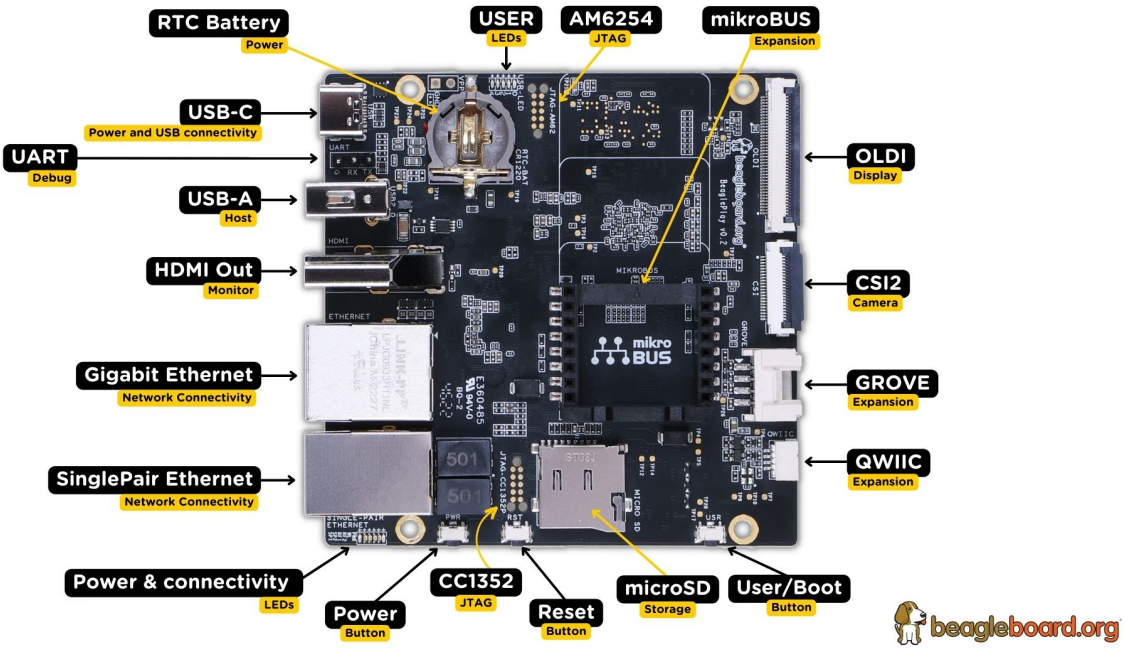


Fig. 2.1: BeaglePlay board front components location

Front components location

Table 2.2: BeaglePlay board front components location

Feature	Description
RTC Battery	BQ32002 Real Time Clock (RTC) Battery holder takes CR1220 3V battery
User LEDs	Five user LEDs, *Power and boot* section provides more details. These LEDs are connect to the AM6254 SoC
JTAG (AM62)	AM6254 SoC JTAG debug port
mikroBUS	mikroBUS for MikroE Click boards or any compliant add-on
OLDI	AM6254 OpenLDI(OLDI) display port
CSI	AM6254 Camera Serial Interface (MIPI CSI-2)
Grove	SeeedStudio Grove modules connection port
QWIIC	SparkFun QWIIC / Adafruit STEMMA-QT port for I2C modules connectivity
User Button	Programmable user button, also servers as boot mode slect button (SD Card/eMMC). Press down to select SD Card as boot medium
SD Card	Use to expand storage, boot linux image or flash latest image on eMMC
Reset button	Press to reset BeaglePlay board (AM6254 SoC)
JTAG (CC1352)	JTAG debug port for CC1352P7
Power button	Press to shut-down (OFF), hold down to boot (ON)
Power & Connectivity LEDs	Indicator LEDs for Power ON, CC1352 RF, and Single-pair connectivity
Single-pair Ethernet	Single-pair Ethernet connectivity port with power over data line
GigaBit Ethernet	1Gb/s Wired internet connectivity
HDMI Output	Full size HDMI port for connecting to external display monitors
USB-A host port	Port to connect USB devices like cameras, keyboard & mouse combos, etc
USB-C port	Power and Device data role port

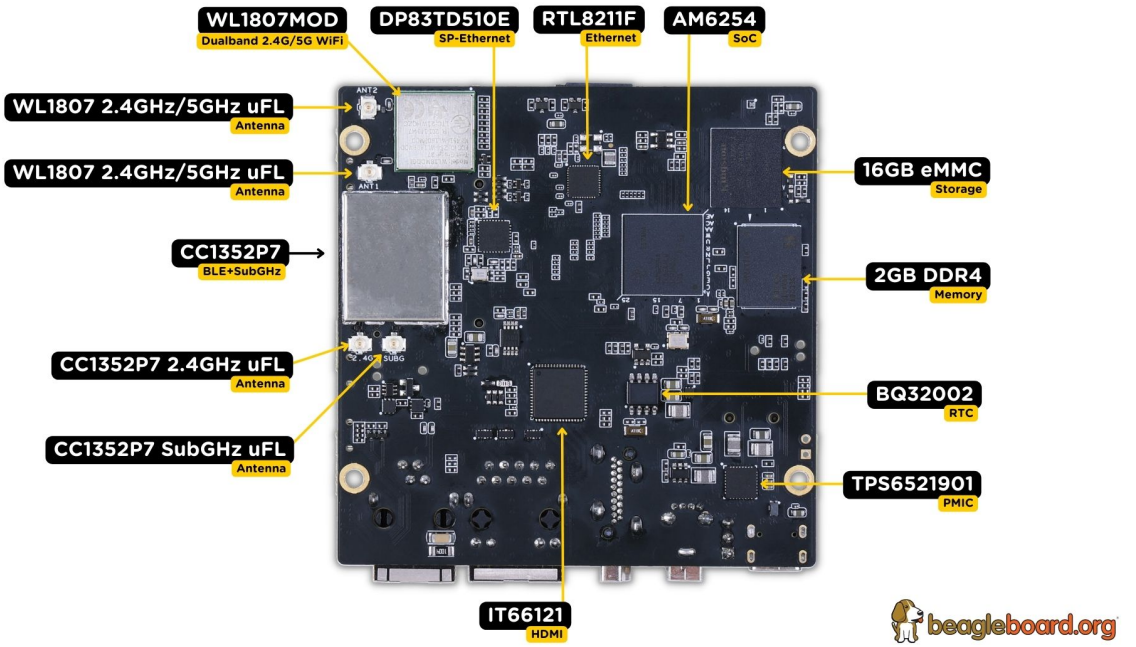


Fig. 2.2: BeaglePlay board back components location

Back components location

Table 2.3: BeaglePlay board back components location

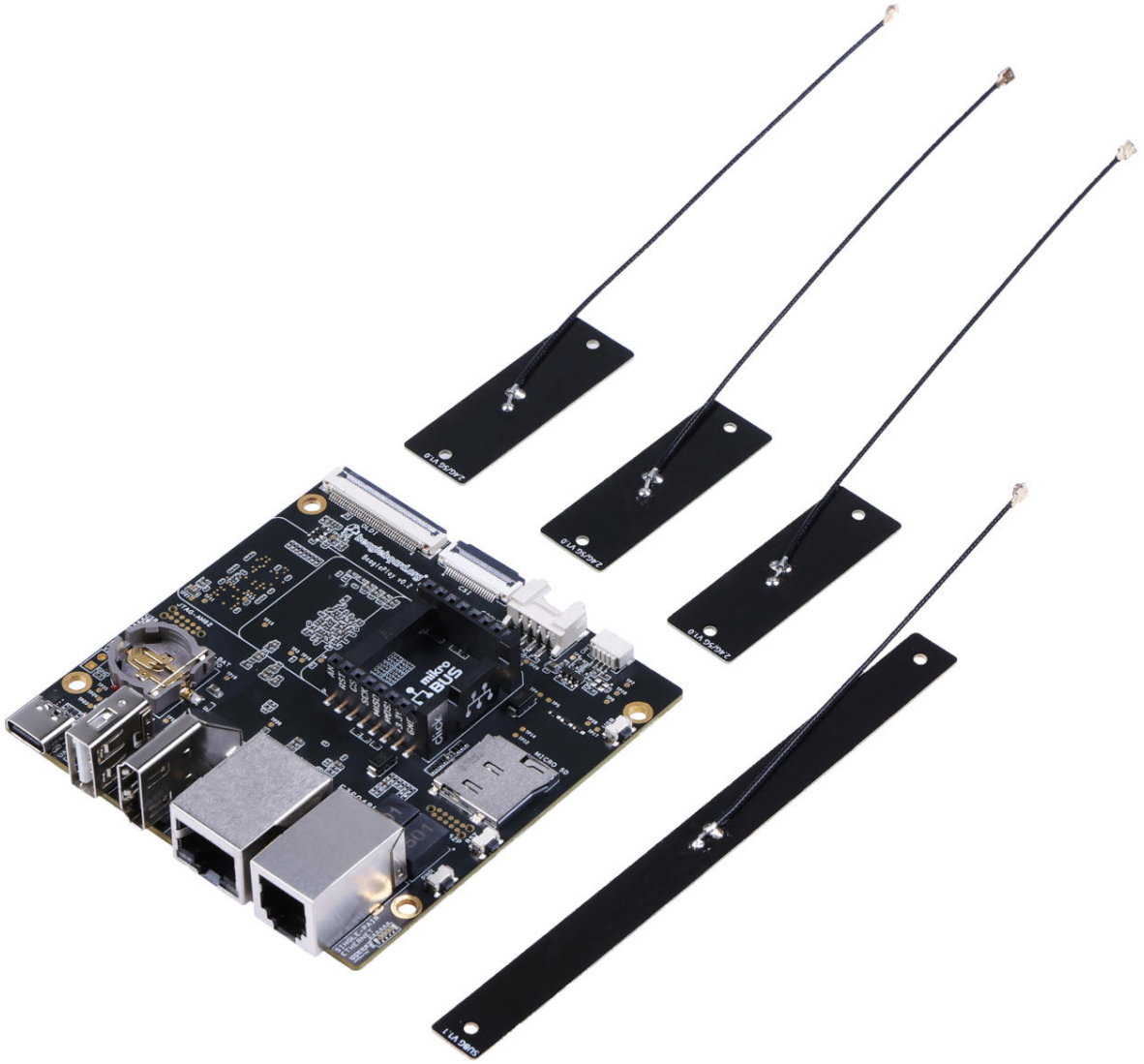
Feature	Description
CC1352P7	2.4GHz BLE + SubG IEEE 802.15.4 with 1 x 2.4GHz + 1 x SubG uFL antenna
WL1807MOD	Dual band (2.4GHz & 5GHz) WiFi module with 2 x uFL antennas
DP83TD510E	Single-pair IEEE 802.3cg 10BASE-T1L Ethernet PHY
RTL8211F	Gigabit IEEE 802.11 Ethernet PHY
AM6254	Main SoC
16GB eMMC	Flash storage
2GB DDR4	RAM / Memory
BQ32002	Real Time Clock (RTC)
TPS6521901	Power Management IC
IT66121	HDMI Transmitter

2.2 Quick Start Guide

2.2.1 What's included in the box?

When you purchase a brand new BeaglePlay, In the box you'll get:

1. BeaglePlay board
2. One (1) sub-GHz antenna
3. Three (3) 2.4GHz/5GHz antennas
4. Plastic standoff hardware
5. Quick-start card



2.2.2 Attaching antennas

You can watch this video to see how to attach the attennas.



2.2.3 Tethering to PC

Tip: Checkout *Getting Started Guide* for,

1. Updating to latest software.
2. Power and Boot.
3. Network connection.
4. Browsing to your Beagle.
5. Troubleshooting.

For tethering to your PC you'll need a USB-C data cable.

2.2.4 Access VSCode

Once connected, you can browse to 192.168.7.2:3000 to access the VSCode IDE to browse documents and start programming your BeaglePlay!

Note: You may get a warning about an invalid or self-signed certificate. This is a limitation of not having a public URL for your board. If you have any questions about this, please as on https://forum.beagleboard.org/tag/play.

2.2.5 Demos and Tutorials

- *Using Serial Console*
- *Connect WiFi*
- *Using QWIIC*
- *Using Grove*

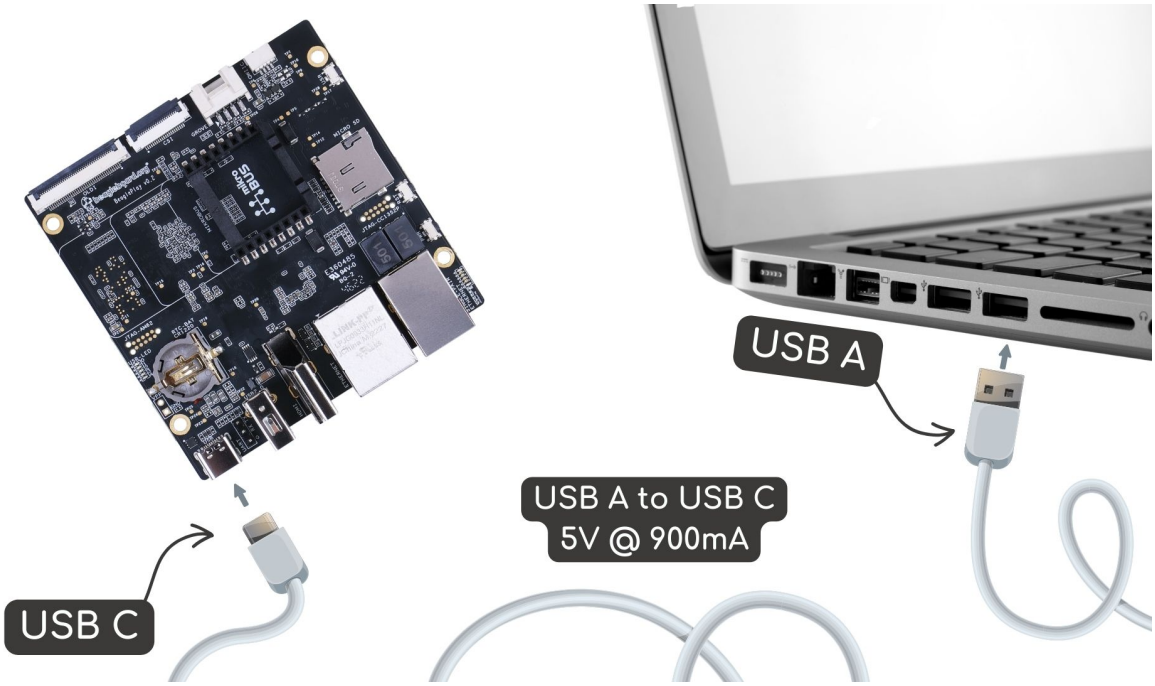


Fig. 2.3: Tethering BeaglePlay to PC

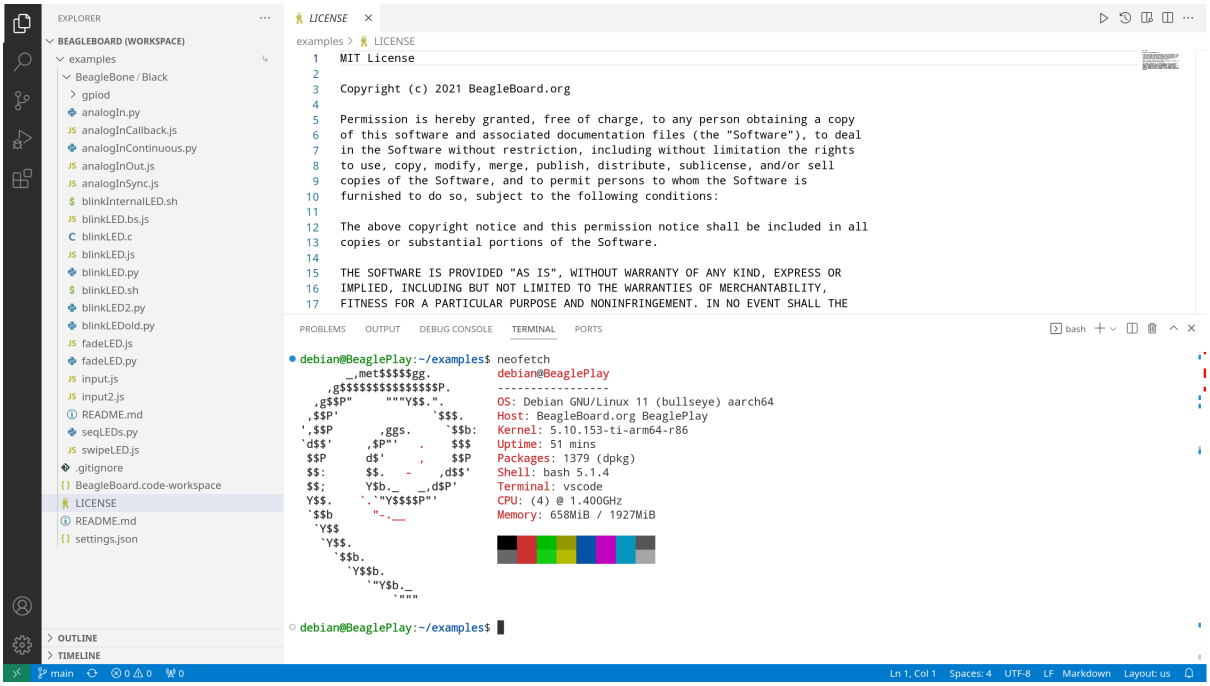


Fig. 2.4: BeaglePlay VSCode IDE (192.168.7.2:3000)

- *Using mikroBUS*
- *Using OLDI Displays*
- *Using CSI Cameras*
- *Wireless MCU Zephyr Development*

2.3 Design and specifications

If you want to know how BeaglePlay is designed and the detailed specifications, then this chapter is for you. We are going to attept to provide you a short and crisp overview followed by discussing each hardware design element in detail.

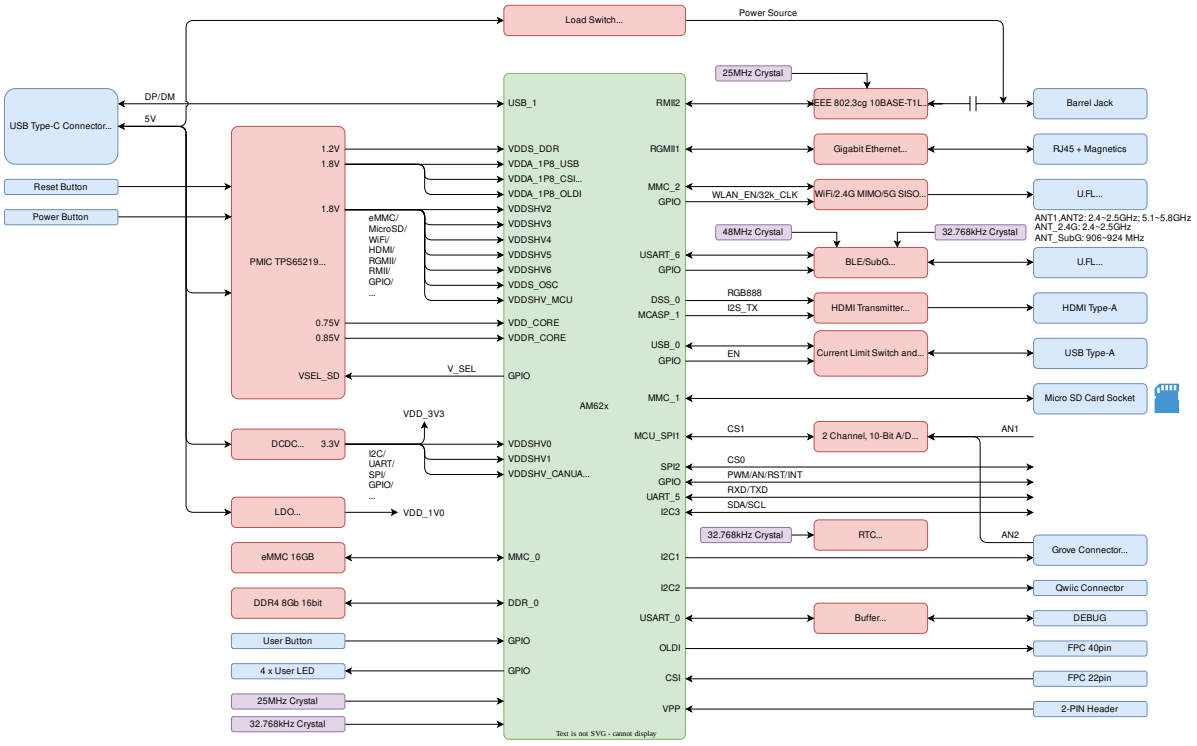
Tip: You can download BeaglePlay schematic to have clear view of all the elements that makes up the BeaglePlay hardware.

BeaglePlay design repository

2.3.1 Block diagram

The block diagram below shows all the parts that makes up your BeaglePlay board. BeaglePlay as mentioned in previous chapters is based on AM6254 SoC which is shown in the middle. Connection of other parts like power supply, memory, storage, wifi, ethernet, and others is also clearly shown in the block diagram. This block diagram shows the high level specifications of the BeaglePlay hardware and the sections below this are going to show you the individual part in more detail with schematic diagrams.

BeaglePlay System Block Diagram



2.3.2 System on Chip (SoC)

AM62x Sitara™ Processors from Texas Instruments are Human-machine-interaction SoC with Arm® Cortex®-A53-based edge AI and full-HD dual display. AM6254 which is on your BeaglePlay board has a multi core design with Quad 64-bit Arm® Cortex®-A53 microprocessor subsystem at up to 1.4 GHz, Single-core Arm® Cortex®-M4F MCU at up to 400MHz, and Dedicated Device/Power Manager. Talking about the multimedia capabilities of the processor you can connect upto two display monitors with 1920x1080 @ 60fps each, additionally there is a OLDI/LVDS (4 lanes - 2x) and 24-bit RGB parallel interface for connecting external display panels. One 4 Lane CSI camera interface is also available which has support for 1,2,3 or 4 data lane mode up to 2.5Gbps speed. The list of features is very long and if you are interested to know more about the AM62x SoC you may take a look at AM62x Sitara™ Processors datasheet.

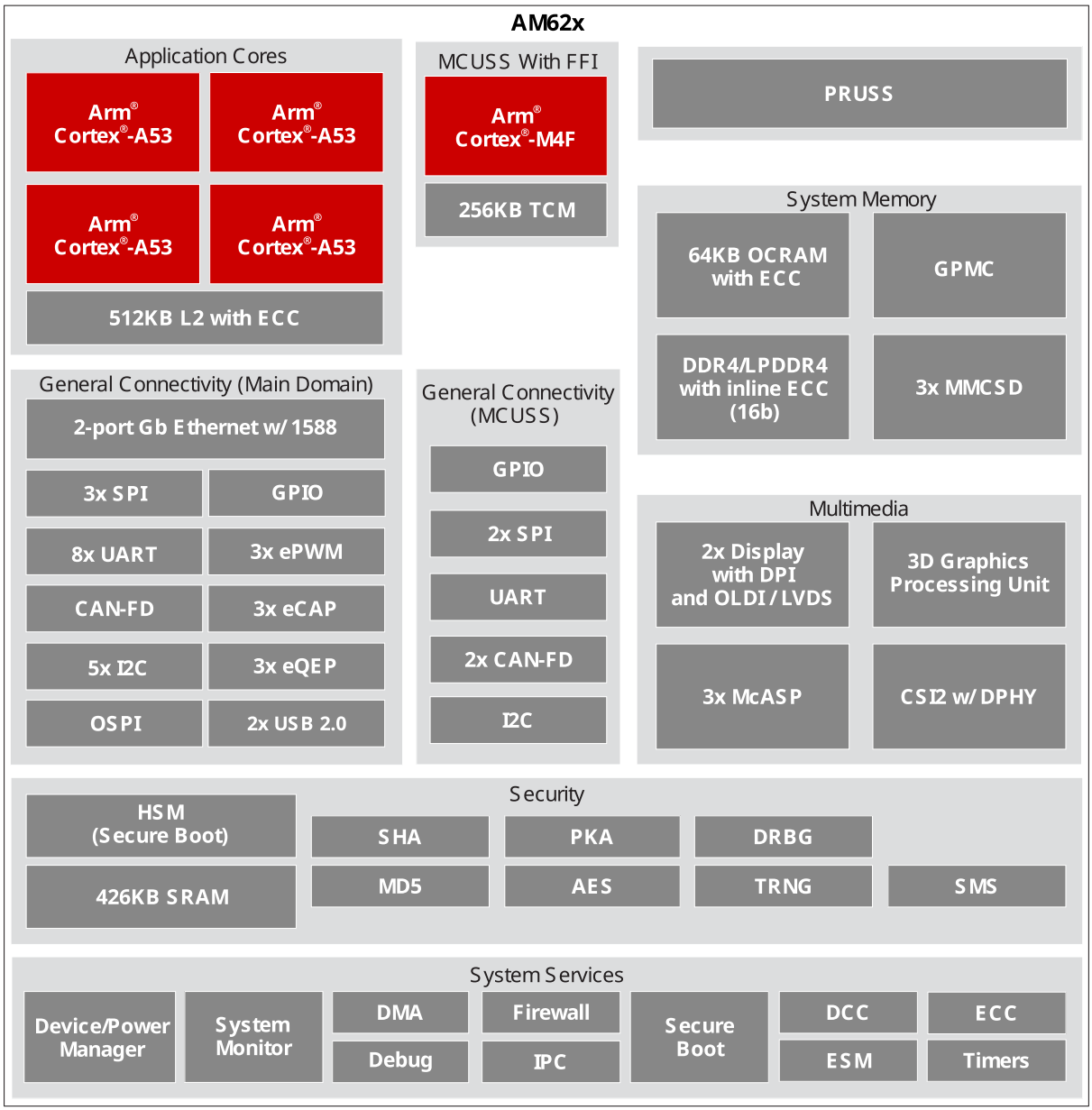


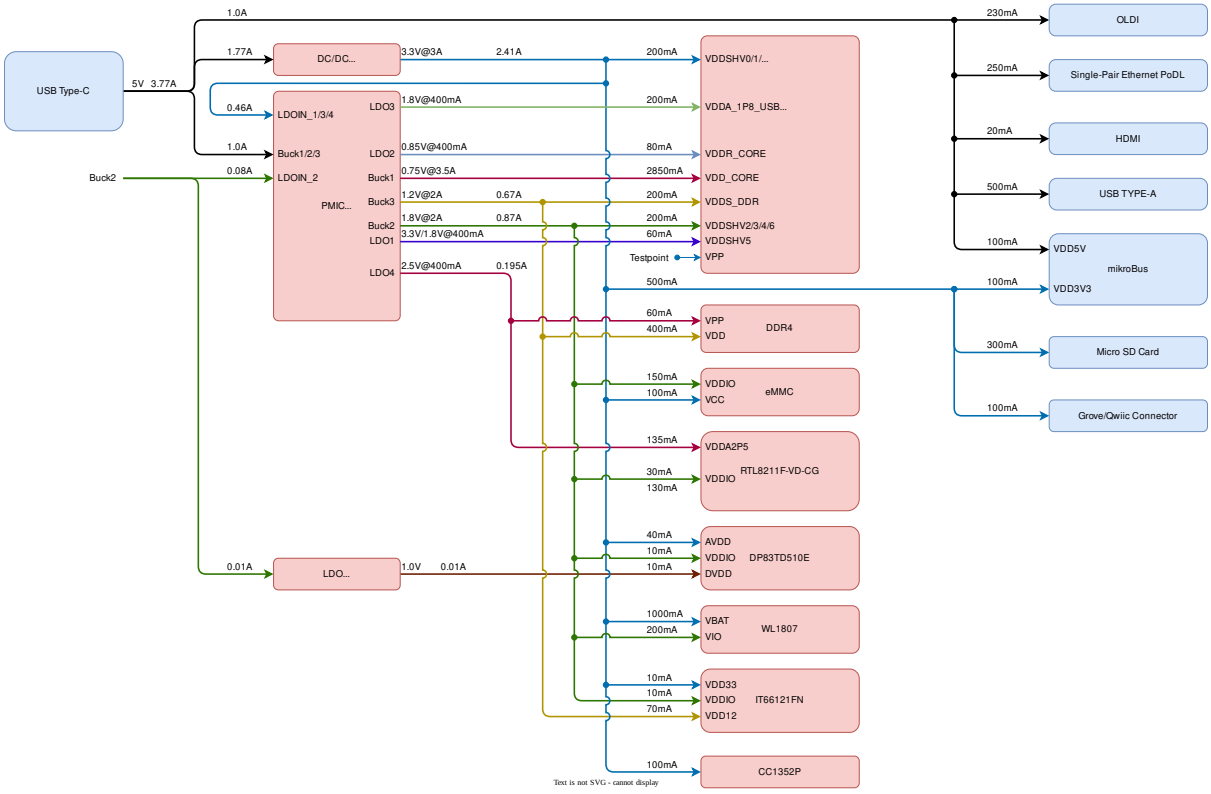
Fig. 2.5: AM6254 SoC block diagram

2.3.3 Power management

Different parts of the board requires different voltages to operate and to fulfill requirements of all the chips on BeaglePlay we have Low Drop Out (LDO) voltage regulators for fixed voltage output and Power Management

Integrated Circuit (PMIC) that interface with SoC to generate software programable voltages. 2 x LDOs and 1 x PMIC used on BeaglePlay are shown below.

BeaglePlay Power Block Diagram



TLV75801 - LDO

This provides 1.0V required by the single-pair Ethernet PHY (U13 - DP83TD510ERHBR). It was decided this was less likely to be needed than the other rails coming off of the primary PMIC and therefore was given its own regulator when running low on power rails.

Note: The voltage drop from 1.8V to 1.0V is rated up to 0.3A (240mW), but the typical current from the DP83TD51E data sheet (SNLS656C) is stated at 3.5mA (2.8mW) and the maximum is 7.5mA (6mW). This isn't overly significant on a board typically consuming 400mA at 5V (2W). However, this is an area where some power optimization could be performed if concerned about sleep modes.

TLV62595 - DC/DC regulator

This provides 3.3V for the vast majority of 3.3V I/Os on the board, off-board 3.3V power to microSD, mikroBUS, QWIIC and Grove connectors, as well as to the PMIC LDO to provide power for the 1.8V on-board I/Os, DDR4, and gigabit Ethernet PHY. Due to the relatively high current rating (3A), a highly efficient (up to 97%) was chosen.

The primary TPS65219 PMIC firmware uses GPO2 to provide the enable signal (VDD_3V3_EN). The power-good signal (VDD_3V3_PG) is available at TP19 and is unused on the rest of the board.

TPS65219 - PMIC

This is the primary power management integrated circuit (PMIC) for the design. It coordinates the power sequencing and provides numerous power rails required for the core of the system, including dynamic voltages

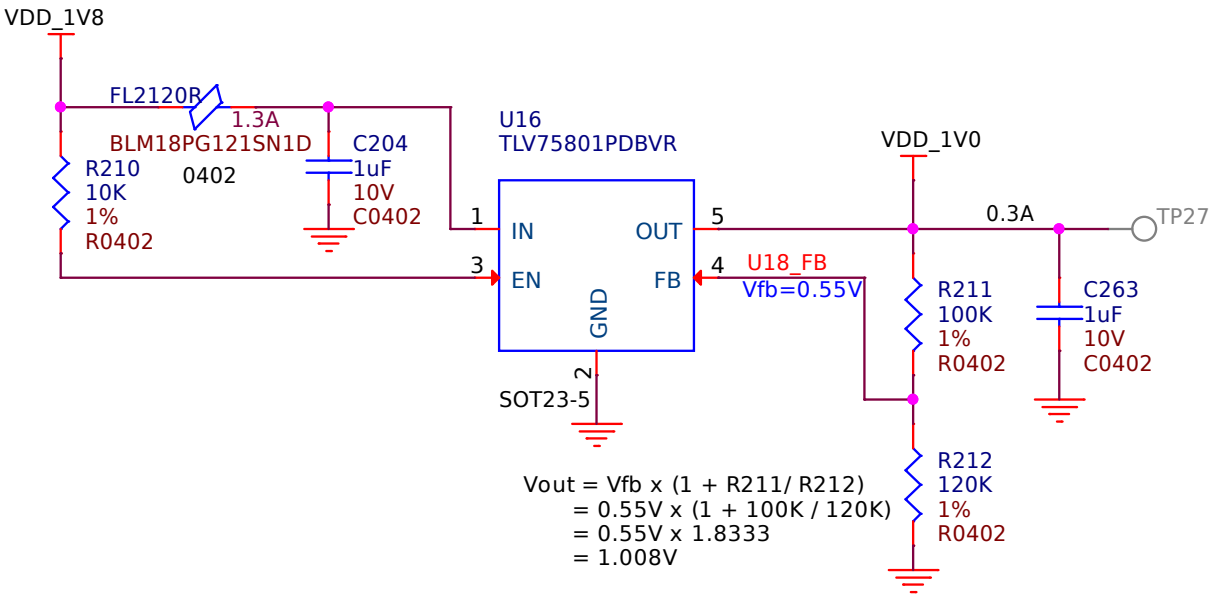


Fig. 2.6: TLV75801PDBVR LDO schematic for 1V0 output

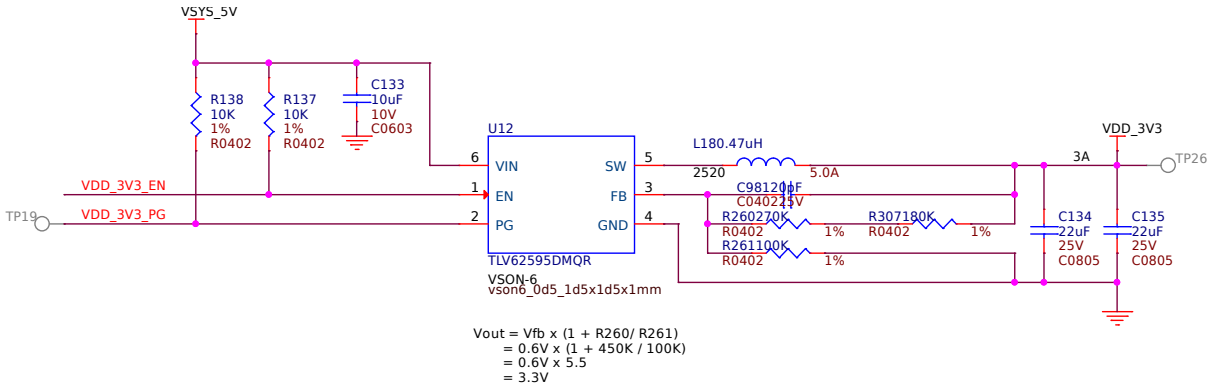


Fig. 2.7: TLV62595DMQR LDO schematic for 3V3 output

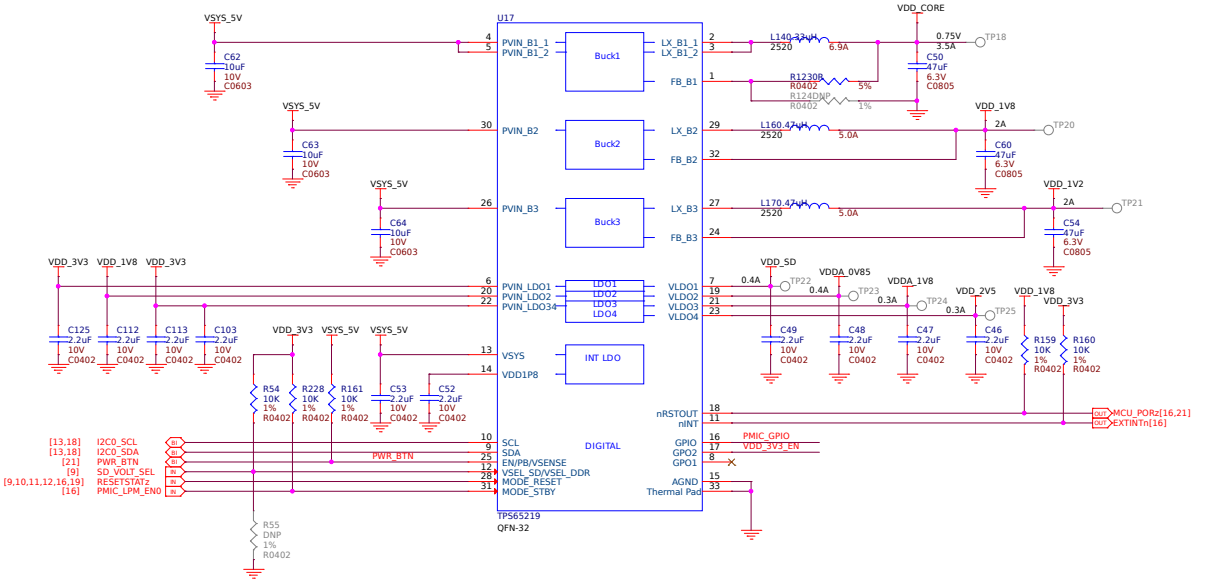


Fig. 2.8: TPS65219 Power Management Integrated Circuit (PMIC) schematic

for the processor core and microSD card. The TPS6521903 variant is used for this DDR4-based system. The 03 at the end indicates the sequencing programmed into the device and is covered in the TPS6521903 Technical Reference Manual SLVUCJ2.

Todo: Add specific power-up/down sequence notes here as well a highlight any limitations and known issues.

2.3.4 General Connectivity and Expansion

One of the main advantage of using a Single Board Computer (SBC) is having direct accessibility of general purpose input & output (GPIO) pins and other interfaces like I2C, SPI, ADC, PWM. Your BeaglePlay board shines in this domain as well with mikroBUS connector that can take 1000s of click board from MikroElektronika, Grove connector allows to connect hundereds of Grove modules from Seeed Studio, and QWIIC connector allows to connect I2C modules like QWIIC modules from SparkFun or STEMMA QT modules from Adafruit. Note that you also get one USB-A port and one USB-C port. BeaglePlay's USB-A port with host support enables you to connect any USB device like your keyboard & mouse. The USB-C connector allows you to power the board and to connect the board to a PC. You can then connect via SSH or use the pre-installed VisualStudio Code editor by putting the address `192.168.7.2:3000` in your web browser.

USB A & USB C

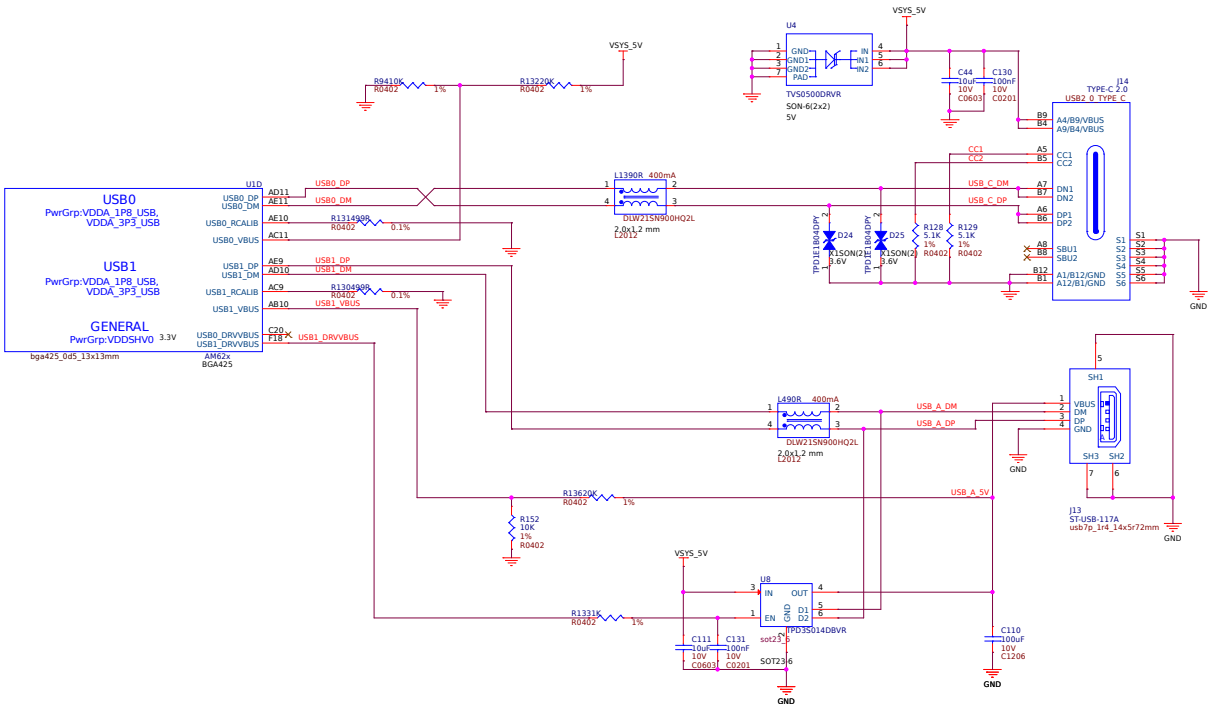


Fig. 2.9: USB-A and USB-C

ADC102S051 - 2ch 10bit ADC

mikroBUS

Grove

QWIIC

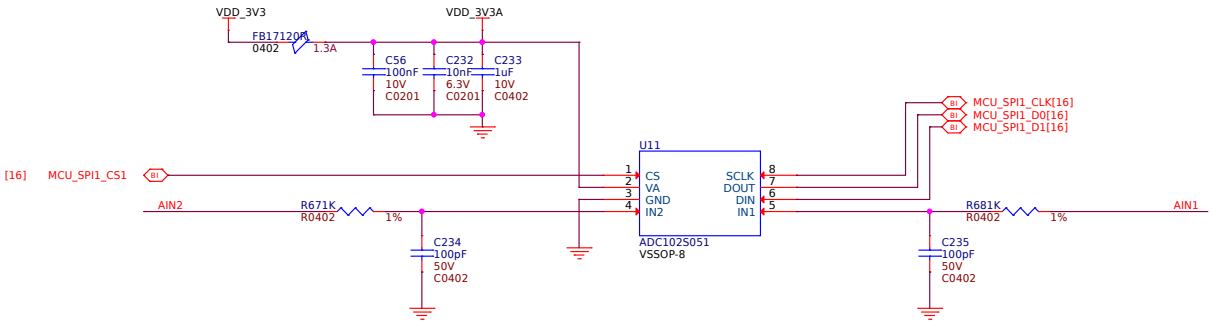


Fig. 2.10: ADC102S051 - 12bit Aanalog to Digital Converter (ADC)

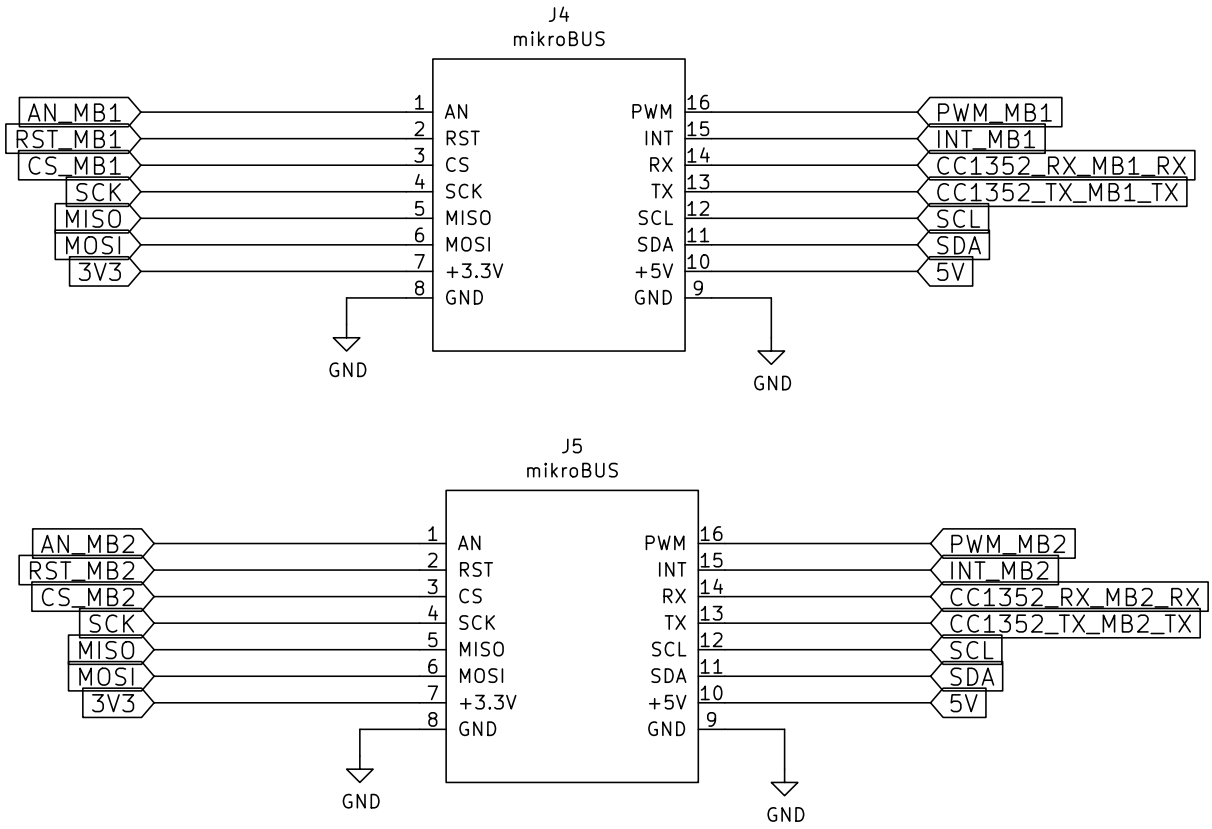


Fig. 2.11: mikroBUS connector schematic

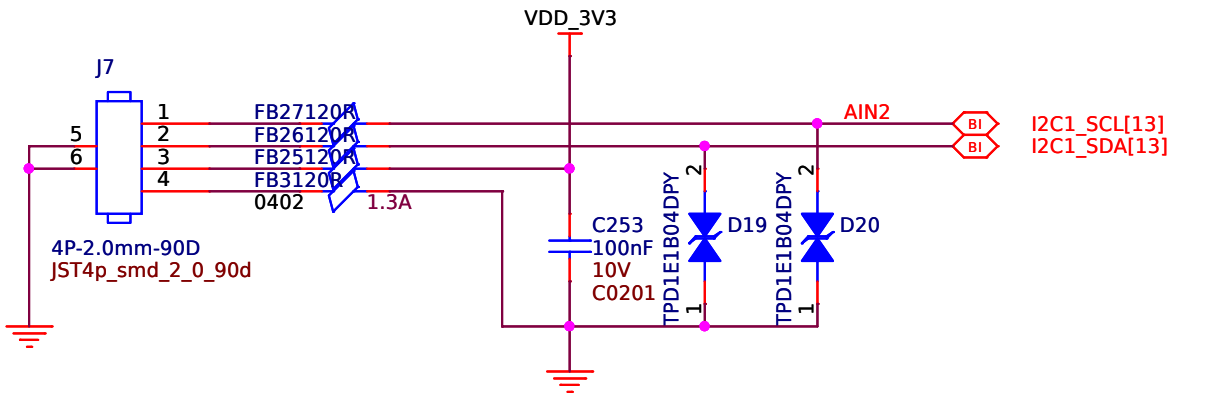


Fig. 2.12: Grove connector schematic

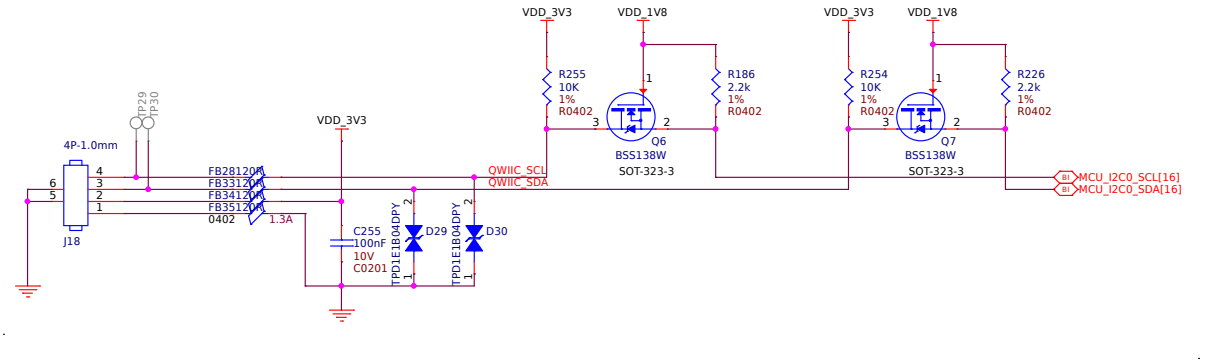


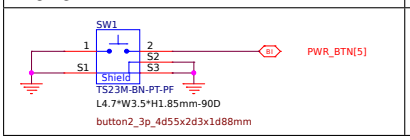
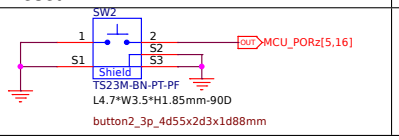
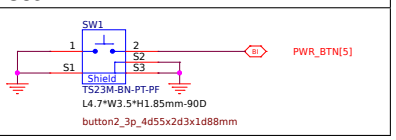
Fig. 2.13: QWIIC connnector for I2C modules

2.3.5 Buttons and LEDs

To interact with the Single Board Computers we use buttons for input and LEDs for visual feedback. On your BeaglePlay board you will find 3 buttons each with a specific purpose: power, reset, and user. For visual feedback you will find 5 user LEDs near USB-C port and 6 more indicator LEDs near your BeaglePlay's Single Pair ethernet port. Schematic diagrams below show how these buttons and LEDs are wired.

Buttons

Table 2.4: BeaglePlay buttons

Power	Reset	User
		

LEDs

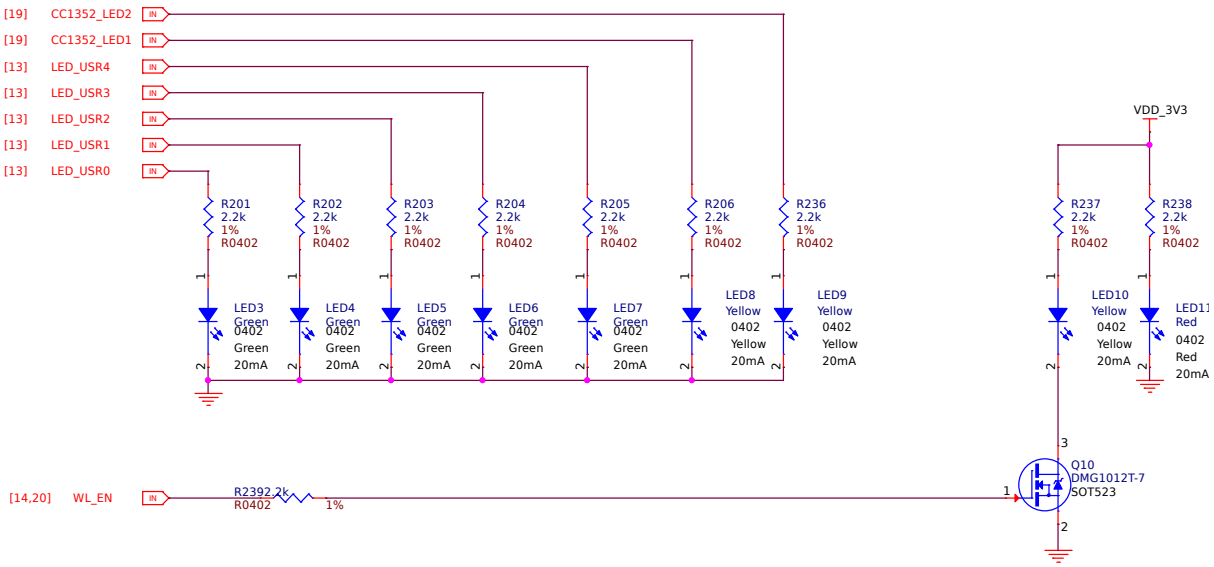


Fig. 2.14: BeaglePlay LEDs

2.3.6 Wired and wireless connectivity

Gigabit ethernet

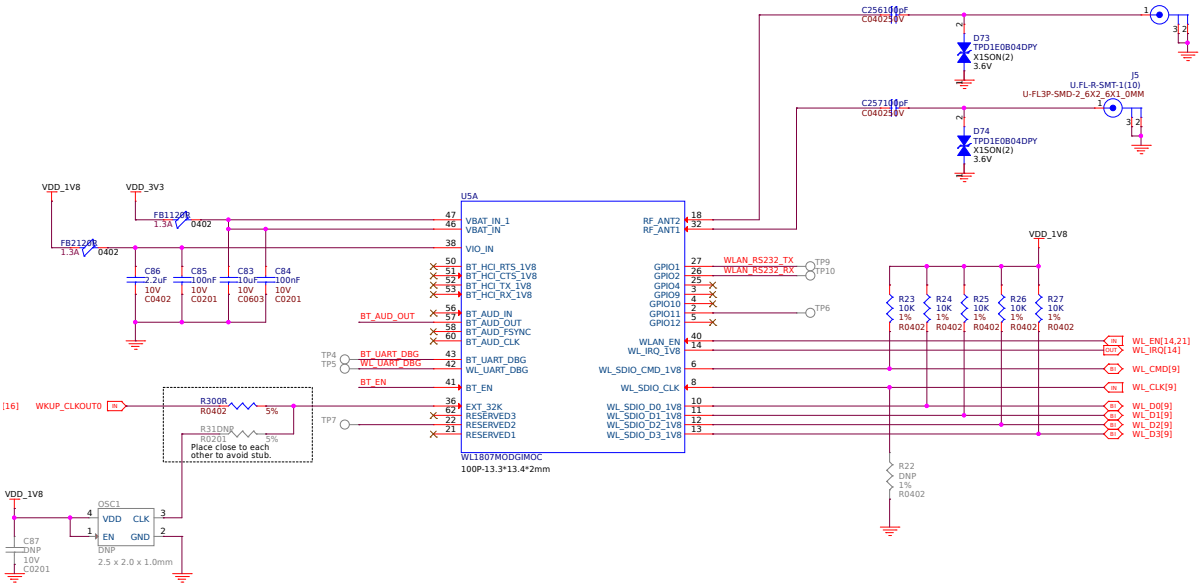


Fig. 2.15: Gigabit ethernet

Single pair ethernet

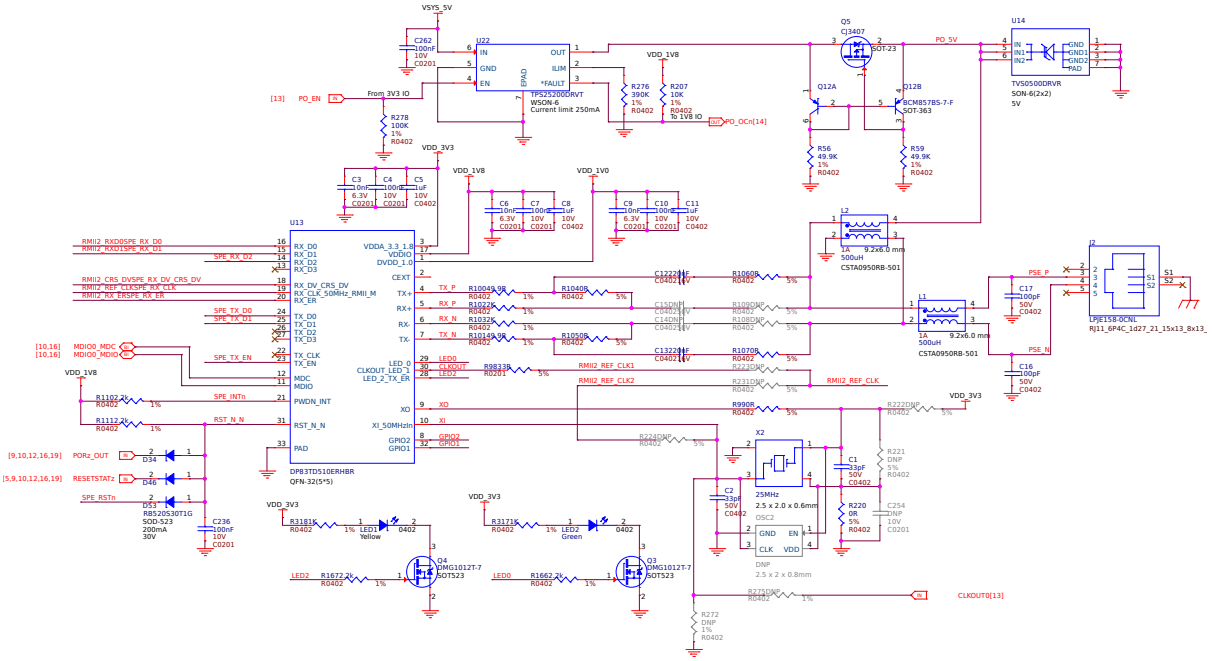


Fig. 2.16: Single pair ethernet

WL1807MOD - WiFi 2.4G/5G

CC1352P7 - BLE & SubGHz

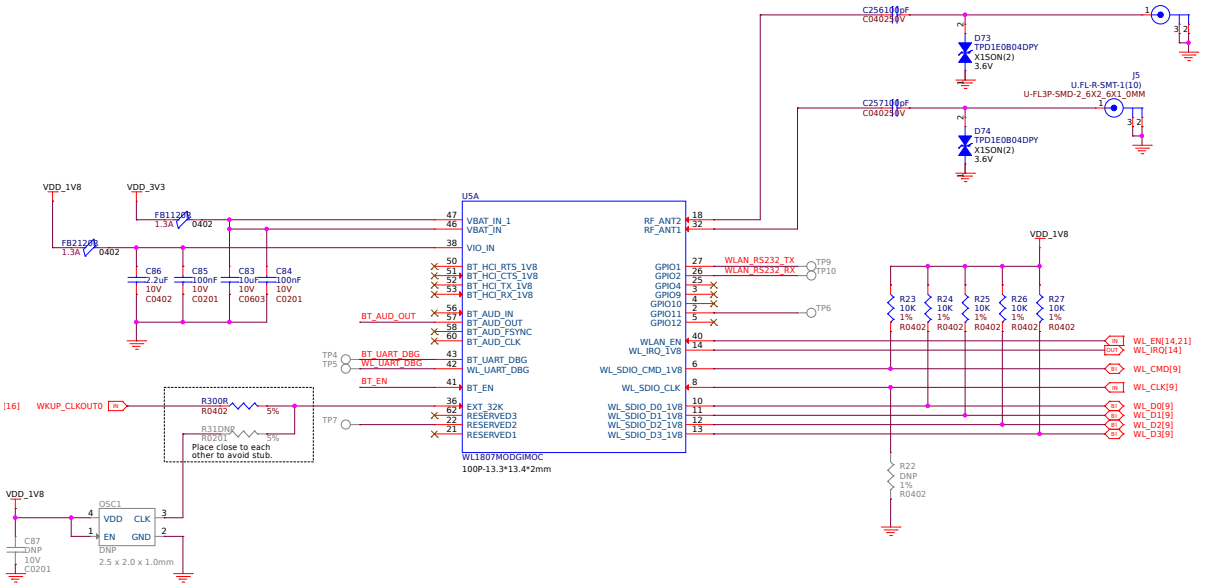


Fig. 2.17: WL1807MOD dual-band (2.4G/5G) WiFi

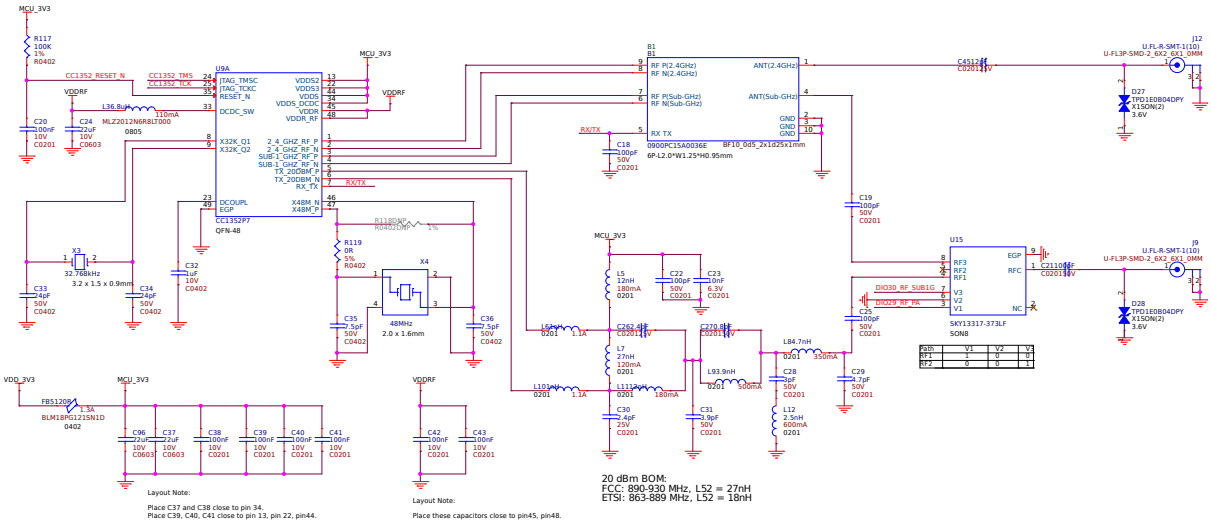


Fig. 2.18: CC1352P7 Bluetooth Low Energy (BLW) and SubGHz connectivity

2.3.7 Memory, Media and Data storage

DDR4

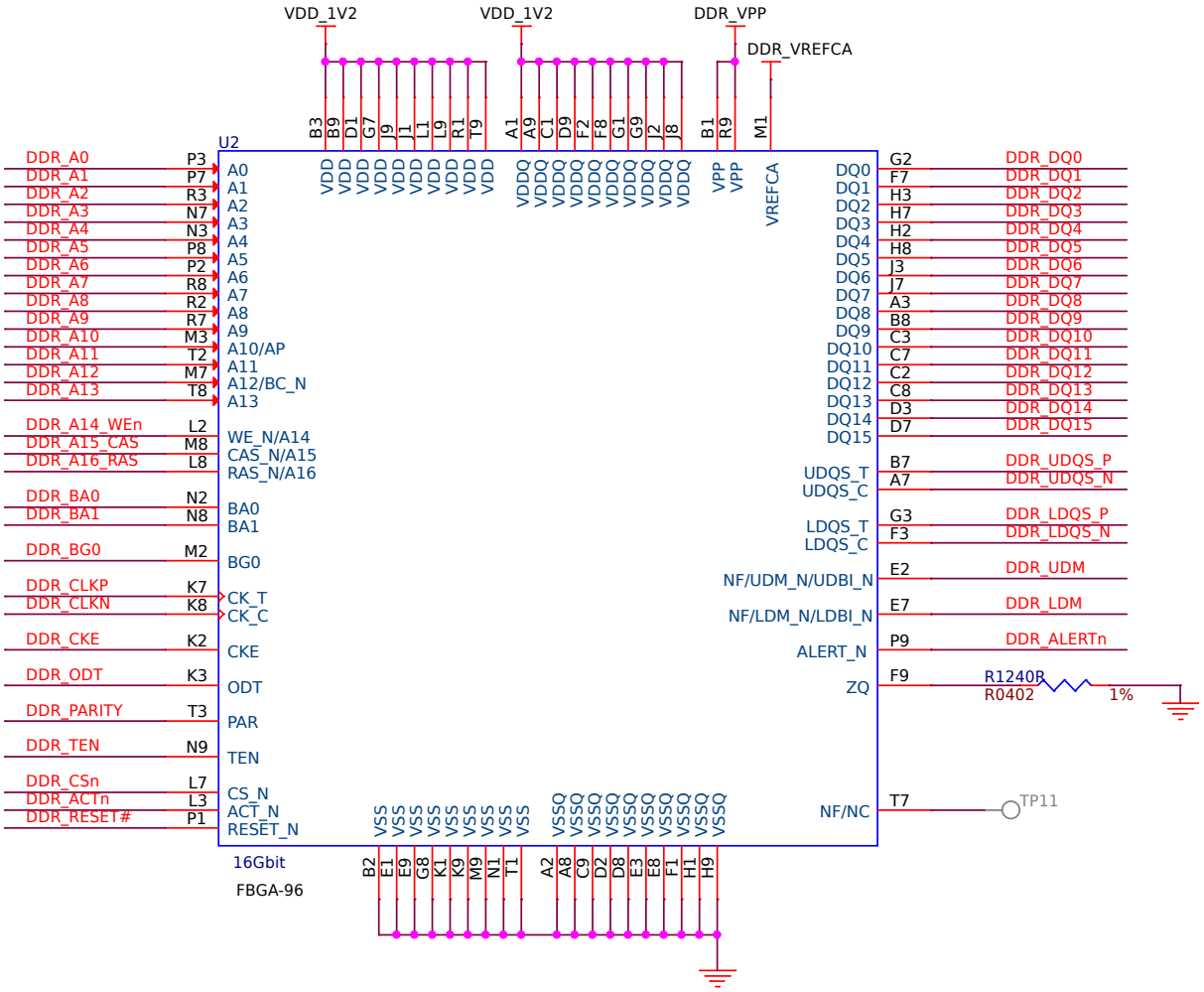


Fig. 2.19: DDR4 Memory

eMMC/SD

microSD Card

Board EEPROM

2.3.8 Multimedia I/O

HDMI

OLDI

CSI

2.3.9 RTC & Debug

RTC

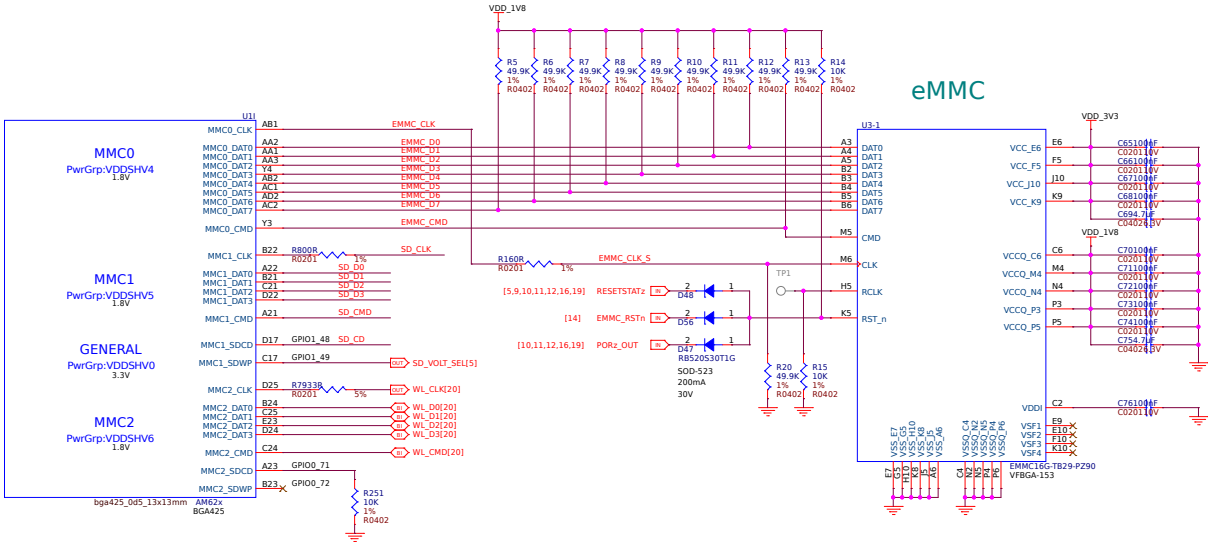


Fig. 2.20: eMMC/SD storage

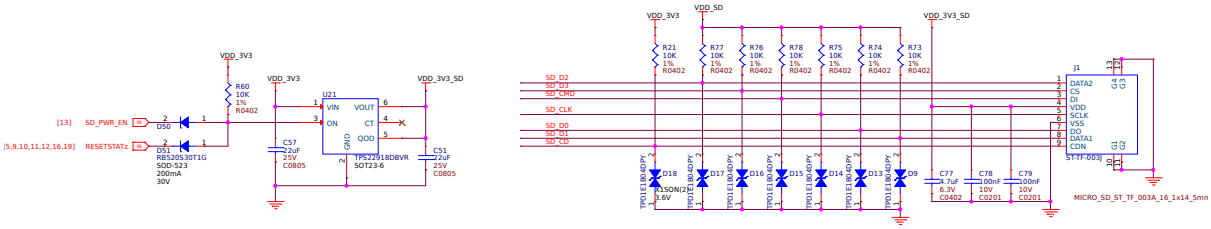


Fig. 2.21: microSD Card storage slot

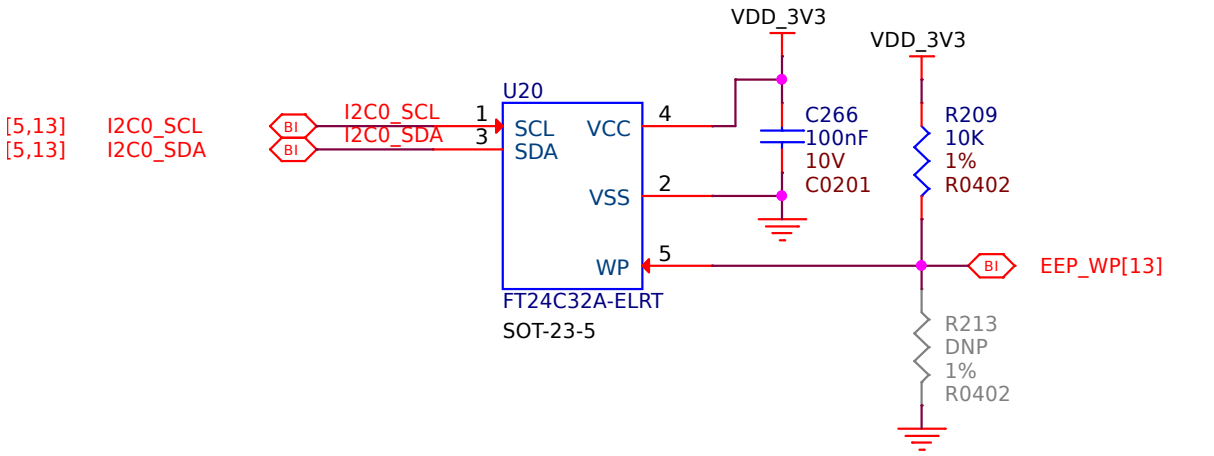


Fig. 2.22: Board EEPROM ID

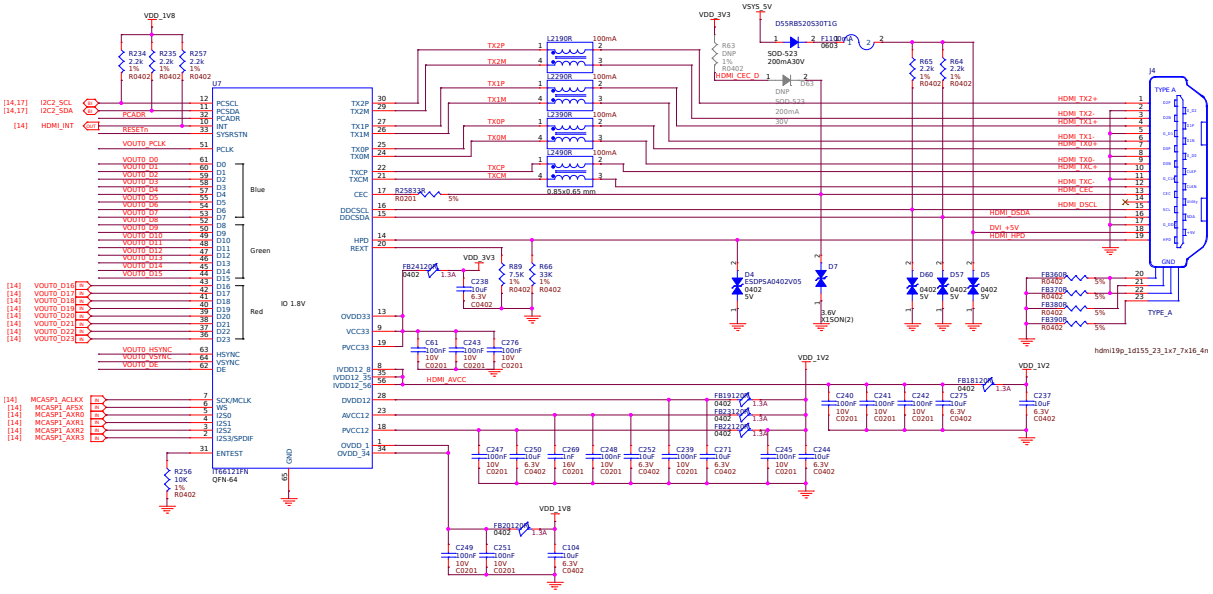


Fig. 2.23: HDMI output

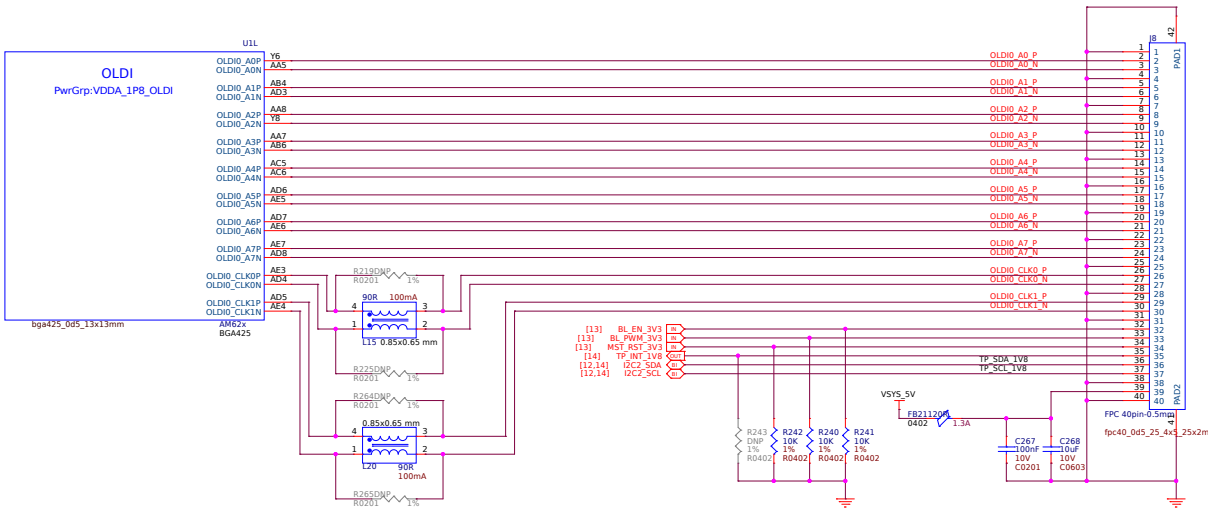


Fig. 2.24: OLDI display interface

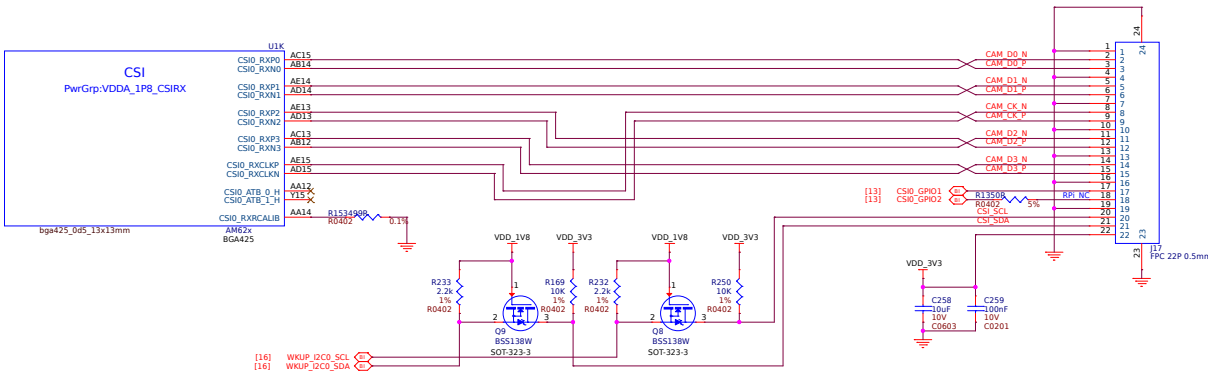


Fig. 2.25: CSI camera interface

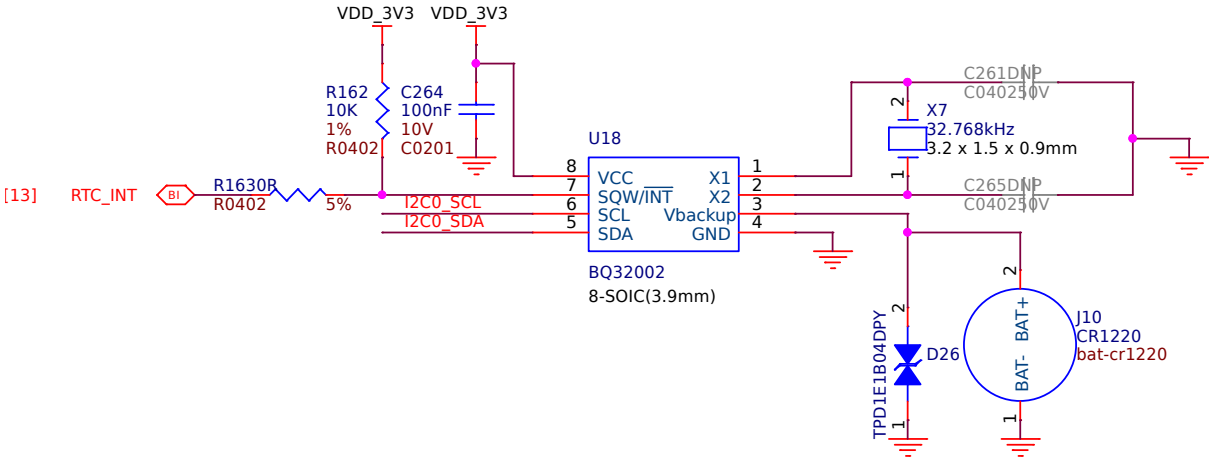


Fig. 2.26: Real Time Clock (RTC)

UART Debug Port

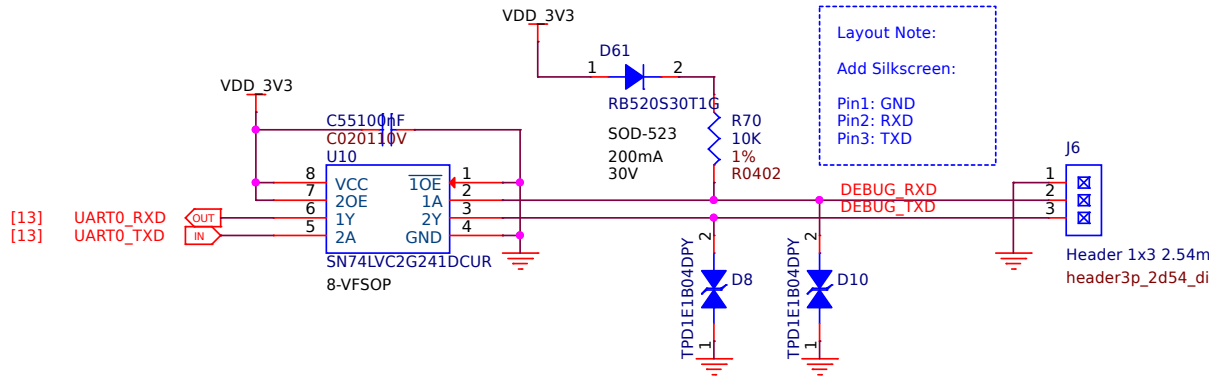


Fig. 2.27: UART debug port

AM62x JTAG & TagConnect

CC1352 JTAG & TagConnect

2.3.10 Mechanical Specifications

Dimensions & Weight

Table 2.5: Dimensions & weight

Parameter	Value
Size	82.5x80x20mm
Max heigh	20mm
PCB Size	80x80mm
PCB Layers	8 layers
PCB Thickness	1.6mm
RoHS compliant	Yes
Weight	55.3g

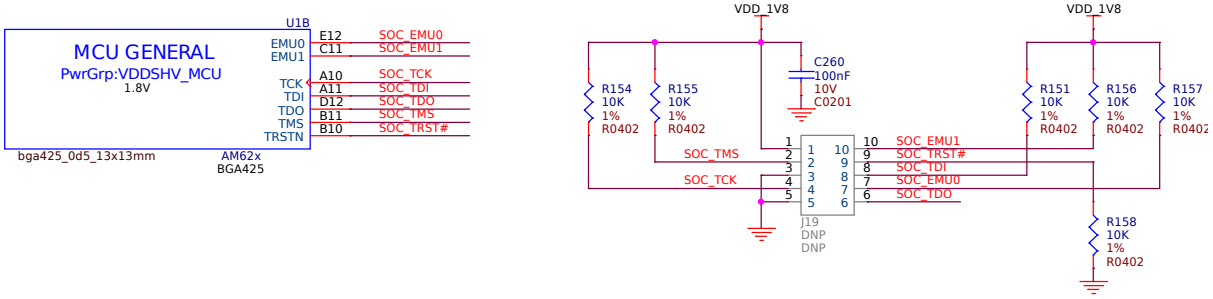


Fig. 2.28: AM62 JTAG debug port and TagConnect interface

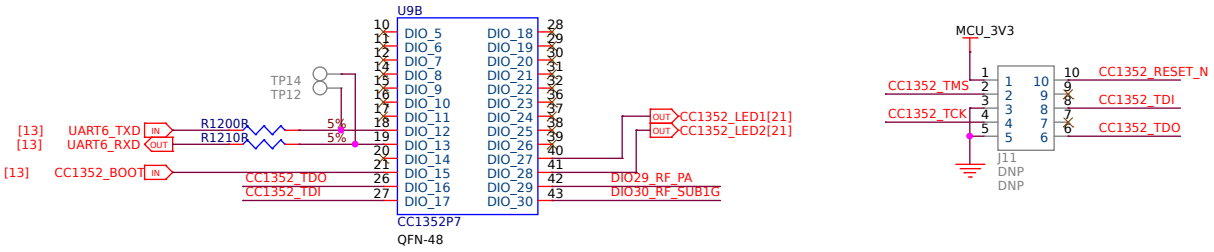


Fig. 2.29: CC1352 JTAG debug port and TagConnect interface

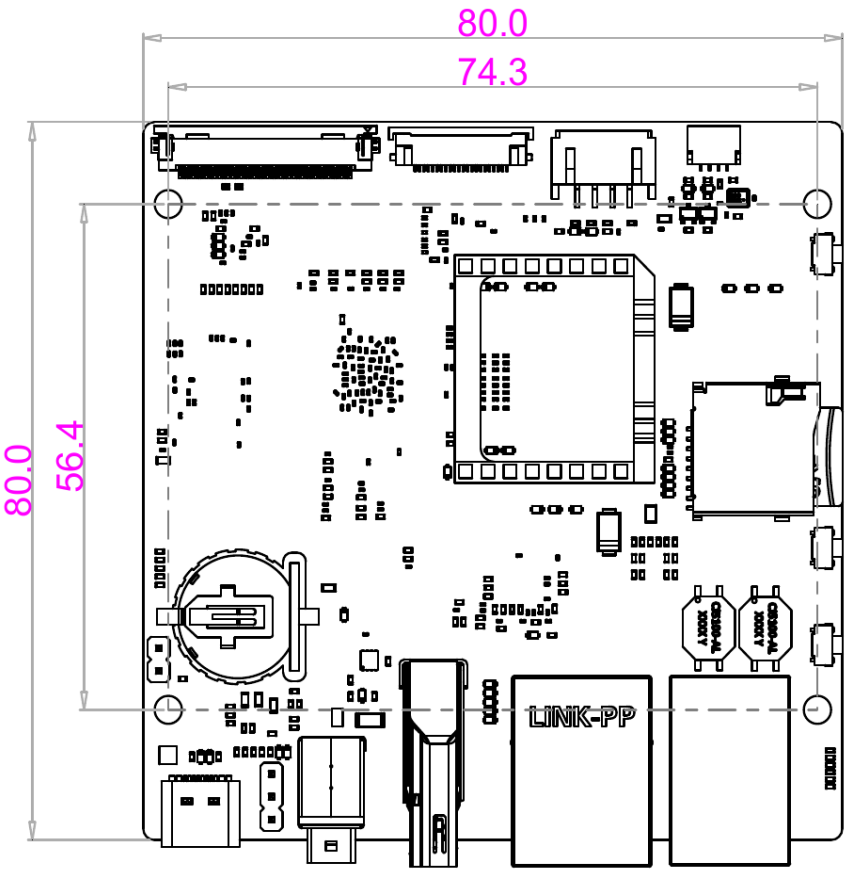


Fig. 2.30: BeaglePlay board dimensions

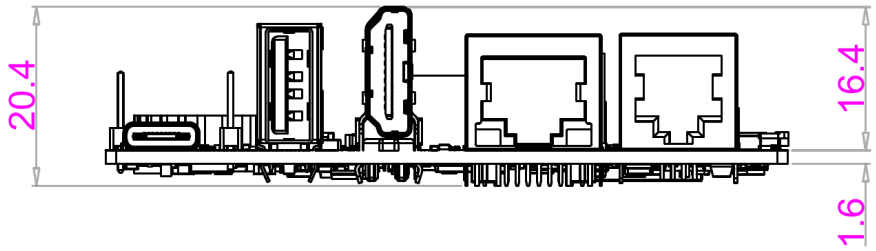


Fig. 2.31: BeaglePlay board side dimensions

2.4 Expansion

Note: This chapter is a work in progress and will include information on building expansion hardware for BeaglePlay.

2.4.1 mikroBUS

The mikroBUS header provides several GPIO pins as well as UART, I2C, SPI, PWM and an Analog Input.

By default, the port is controlled by a mikroBUS driver that helps with auto-detecting MikroE Click Board that feature [ClickID](https://www.mikroe.com/clickid). This does however mean that if you want to manually control the port, you may need to first disable the driver.

To disable the driver, do the following - TODO

2.4.2 Grove

The Grove port on BeaglePlay exposes one of the SoC I2C Ports as well as an analog input.

It maps directly in linux as /dev/I2C-TODO or as the following alias /dev/play/grove

2.4.3 QWIIC

The QWIIC port on BeaglePlay exposes one of the SoC I2C Ports.

It maps directly in linux as */dev/I2C-2* or as the following alias */dev/play/qwiic*

2.4.4 CSI

The AM62x SoC (and by extension BeaglePlay) does not feature on-board ISP (Image Signal Processor) hardware, and as such, Raw-Bayer CSI Sensors must be pre-processed into normal images by the A53 cores.

To avoid performance penalties related to the approach above, it is recommended to use a sensor with a built-in ISP, such as the OV5640 which is supported out of box.

The PCam5C from Digilent is one CSI camera that features this sensor.

Note: Since BeaglePlay uses a 22-pin CSI connector, a 15 pin to 22 pin CSI adapter may also be required such as this one

Once installed, there are some software changes required to load the device driver at boot for the OV5640.

We will need to modify the following file: /boot/firmware/extlinux/extlinux.conf

We will add the following line to load the OV5640 DTBO:

```
fdtoverlays /overlays/k3-am625-beagleplay-csi2-ov5640.dtbo
```

Then you can reboot: sudo reboot

Camera should now work, you can use mplayer to test.

```
sudo apt-get install -y mplayer

mplayer tv: // -tv driver=v4l2:device=/dev/
↪video0:width=640:height=480:fps=30:outfmt=yuy2
```

2.4.5 OLDI

BeaglePlay brings out two OLDI (LVDS) channels, each with up to four data lanes and one clock lane to support 21/28-bit serialized RGB pixel data and synchronization transmissions. The first port, OLDI0, consists of OLDI0_A0-3/CLK0 and corresponds to odd pixels, while the second port, OLDI1, consists of OLDI0_A4-7/CLK1 and corresponds to even pixels.

It is pin compatible with the following two displays from Lincoln Technology Solutions:

Both displays have the following features and only differ in bezzle type:

- **Resolution** - 1920x1200 (16:10)
- **LCD Size (diagonal)** - 10.1”
- **Refresh Rate** - 60Hz
- **Brightness** - 1000nit
- **Pannel Type** - Edge-lit IPS
- **Touch Enabled** - Yes, Capacitive
- **Connector** - 40 pin FFC ribbon cable

A “Flush Coverglass” Version A “Oversized Cover Glass” Version - similar in style to a Tablet Display

To enable OLDI display support, modify the following file: /boot/firmware/extlinux/extlinux.conf

Then, add the following line to load the Lincoln LCD185 OLDI DTBO:

```
fdtoverlays /overlays/k3-am625-beagleplay-lt-lcd185.dtbo
```

Your /boot/firmware/extlinux/extlinux.conf file should look something like this:

```
label Linux eMMC
   kernel /Image
   append root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait net.ifnames=0␣
↪systemd.unified_cgroup_hierarchy=false quiet
   fdtdir /
   fdtoverlays /overlays/k3-am625-beagleplay-lt-lcd185.dtbo
   initrd /initrd.img
```

2.5 Demos and tutorials

2.5.1 Using Serial Console

To see the board boot log and access your BeaglePlay's console you can connect a USB-UART cable as depicted in the image below and use applications like `tio` to access the console.

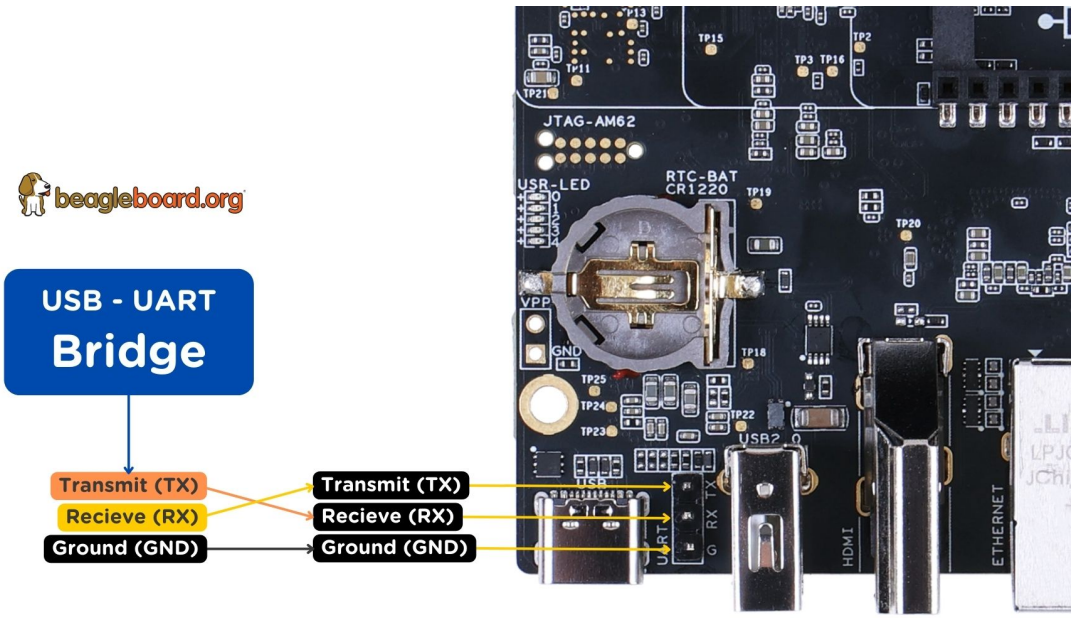


Fig. 2.32: Serial debug (USB-UART) cable connection.

If you are using Linux your USB to UART converter may appear as `/dev/ttyUSB`. It will be different for Mac and Windows operating systems.

```
[lorforlinux@fedora ~] $ tio /dev/ttyUSB0
tio v2.5
Press ctrl-t q to quit
Connected
```

Tip: For more information on `USB to UART` cables, you can checkout *Serial Debug Cables* section.

2.5.2 Connect WiFi

Note: A common issue experienced by users when connecting to Wireless networks are network names that include special characters such as spaces, apostrophes etc, this may make connecting to your network more difficult. It is thus recommended to rename your Wireless AP to something simpler. For Example - renaming "Boris's Wireless Network" to "BorisNet". This avoids having to add special "escape" characters in the name. This shows up especially if you try connecting to iPhone/iOS HotSpots, where the network name is the device name, which by default is something like "Dan's iPhone". Also see this potential solution..

If you have a monitor and keyboard/mouse combo connected, the easiest way is to use the *wpa_gui*.

Alternatively, you can use `wpa_cli` over a shell connection through:

- the *serial console*,

- VSCode or `ssh` over a USB network connection,
- VSCode or `ssh` over an Ethernet connection,
- VSCode or `ssh` over *BeaglePlay WiFi access point*, or
- *a local Terminal Emulator session*.

Once you have a shell connection, follow the *wpa_cli instructions*.

BeaglePlay WiFi Access Point

Running the default image, your BeaglePlay should be hosting a WiFi access point with the SSID "BeaglePlay-XXXX", where *XXXX* is selected based on a hardware identifier on your board to try to increase the chances it will be unique.

Tip: The "XXXX" will be a combination of numbers and the letters A through F.

Note: At some point, we plan to introduce a captive portal design that will enable using your smartphone to provide BeaglePlay local WiFi login information. For now, you'll need to use a computer and

Step 1. Connect to BeaglePlay-XXXX

Tip: The password is either "BeaglePlay" or "BeagleBone" and the IP address will be 192.168.8.1.

Whatever your computer provides as a mechanism for searching for WiFi access points and connecting to them, just use that. You will want to have DHCP enabled, but that is the typical default. Connect to the "BeaglePlay-XXXX" access point and use the password "BeaglePlay" or "BeagleBone".

Note: The configuration for the access point is in the file system at `/etc/hostapd/hostapd.conf`.

Once your are connected to the access point, BeaglePlay should provide your computer an IP address and use 192.168.8.1 for itself. It should also be broadcasting the mDNS name "beagleplay.local".

Step 2. Browse to 192.168.8.1 Once you have connected to the access point, you can simply open VSCode by browsing to https://192.168.8.1:3000.

Within VSCode, you can press "CTRL-'" to open a terminal session to get access to a shell connection.

You could also choose to *ssh* into your board via *ssh debian@192.168.8.1* and use the password *temppwd*.

Important: Once logged in, you should change the default password using the *passwd* command.

wpa_gui

Simplest way to connect to WiFi is to use `wpa_gui` tool pre-installed on your BeaglePlay. Follow simple steps below to connect to any WiFi access point.

Step 1: Starting wpa_gui You can start `wpa_gui` either from `Applications > Internet > wpa_gui` or double click on the `wpa_gui` desktop application shortcut.

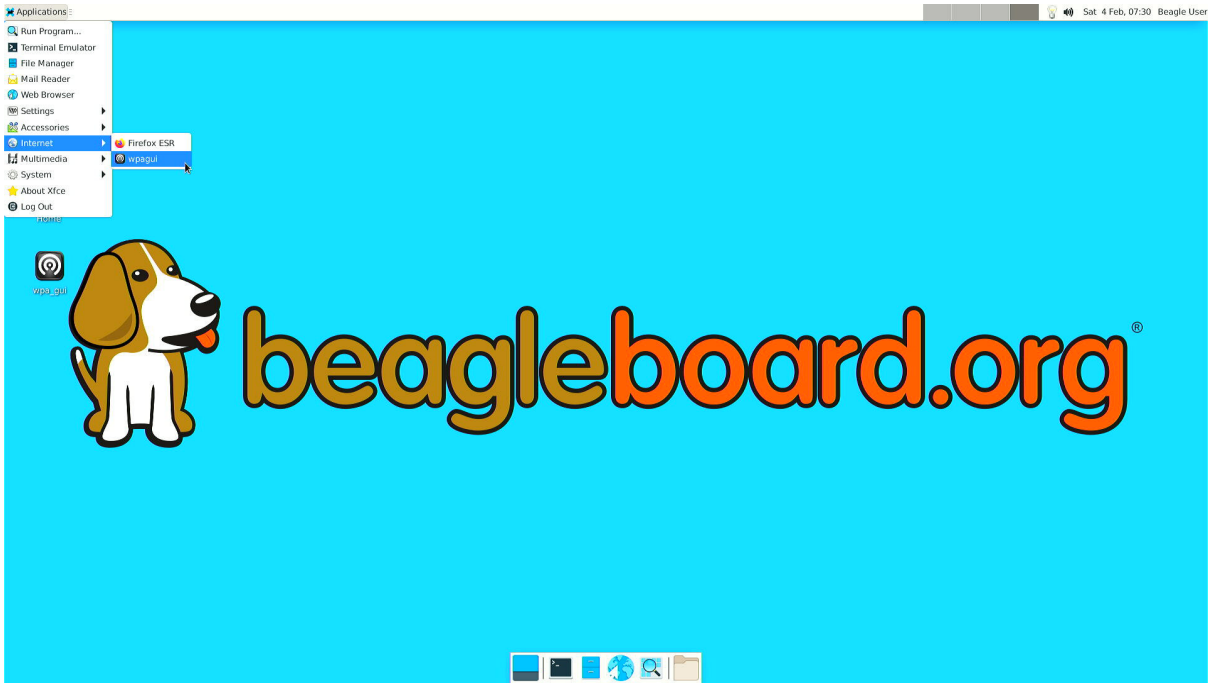


Fig. 2.33: Starting wpa_gui from Applications > Internet > wpa_gui

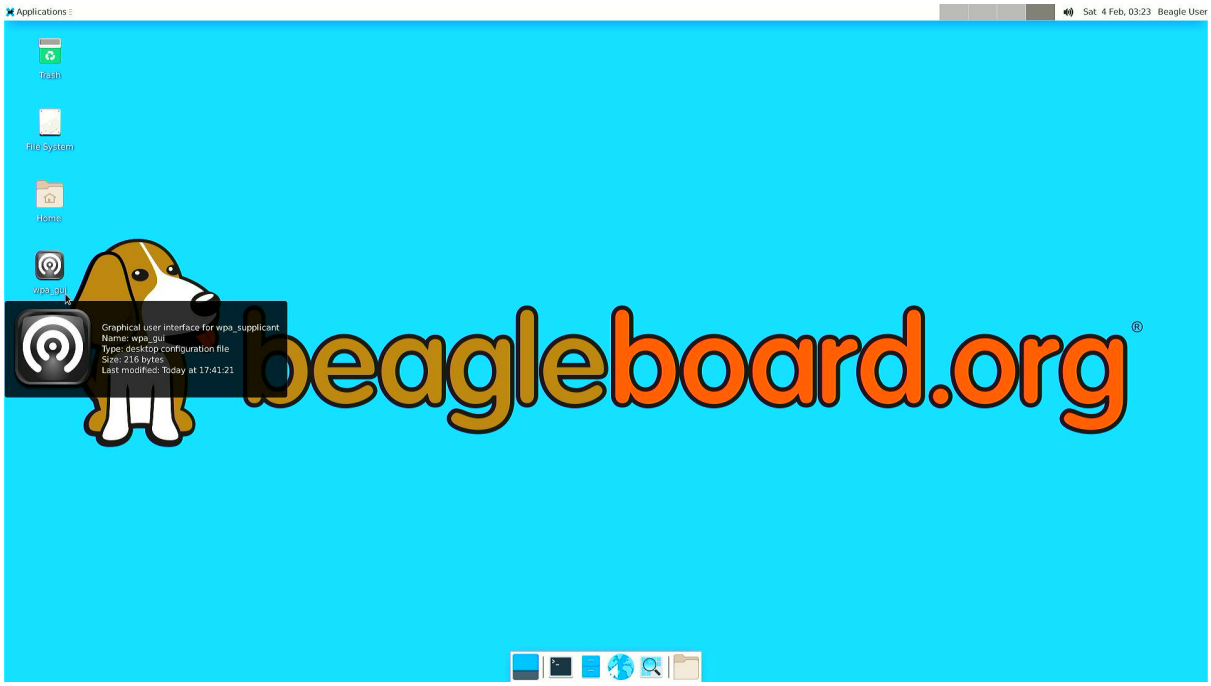


Fig. 2.34: Starting wpa_gui from Desktop application shortcut

Step 2: Understanding wpa_gui interface Let's see the `wpa_gui` interface in detail,

1. `Adapter` is the WiFi interface device, it should be `wlan0` (on-board WiFi) by default.
2. `Network` shows the WiFi access point `SSID` if you are connected to that network.
3. **`Current Status` tab shows you network information if you are connected to any network.**

    - Click on `Connect` to connect if not automatically done.
    - Click on `Disconnect` to disconnect/reset the connection.
    - Click on `Scan` to scan nearby WiFi access points.

4. `Manage Network` tab shows you all the saved networks and options to manage those.

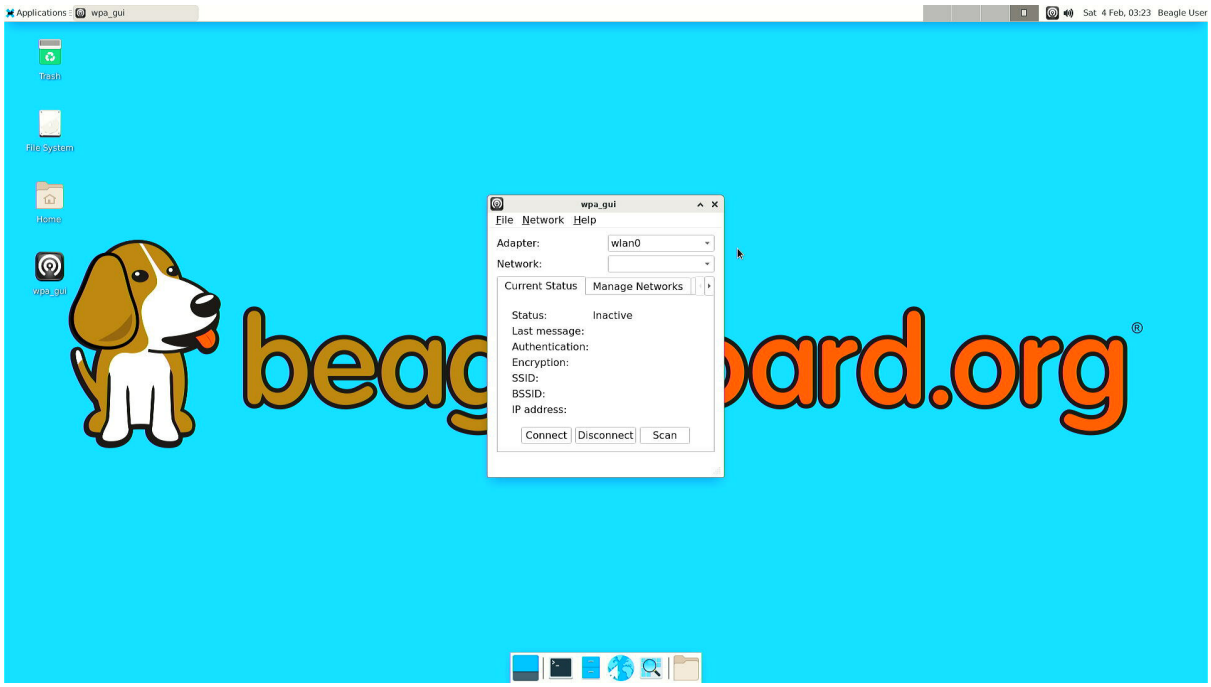


Fig. 2.35: wpa_gui interface

Step 3: Scanning & Connecting to WiFi access points To scan the WiFi access points around you, just click on `Scan` button availale under `wpa_gui > Current Status > Scan`.

A new window will open up with,

1. SSID (WiFi name)
2. BSSID
3. Frequency
4. Signal strength
5. flags

Now, you just have to double click on the Network you want to connect to as shown below.

Note: SSIDs and BSSIDs are not fully visible in screenshot below but you can change the column length to see the WiFi names better.

Final step is to type your WiFi access point password under `PSK` input field and click on `Add` (as shown in screenshot below) which will automatically connect your board to WiFi (if password is correct).

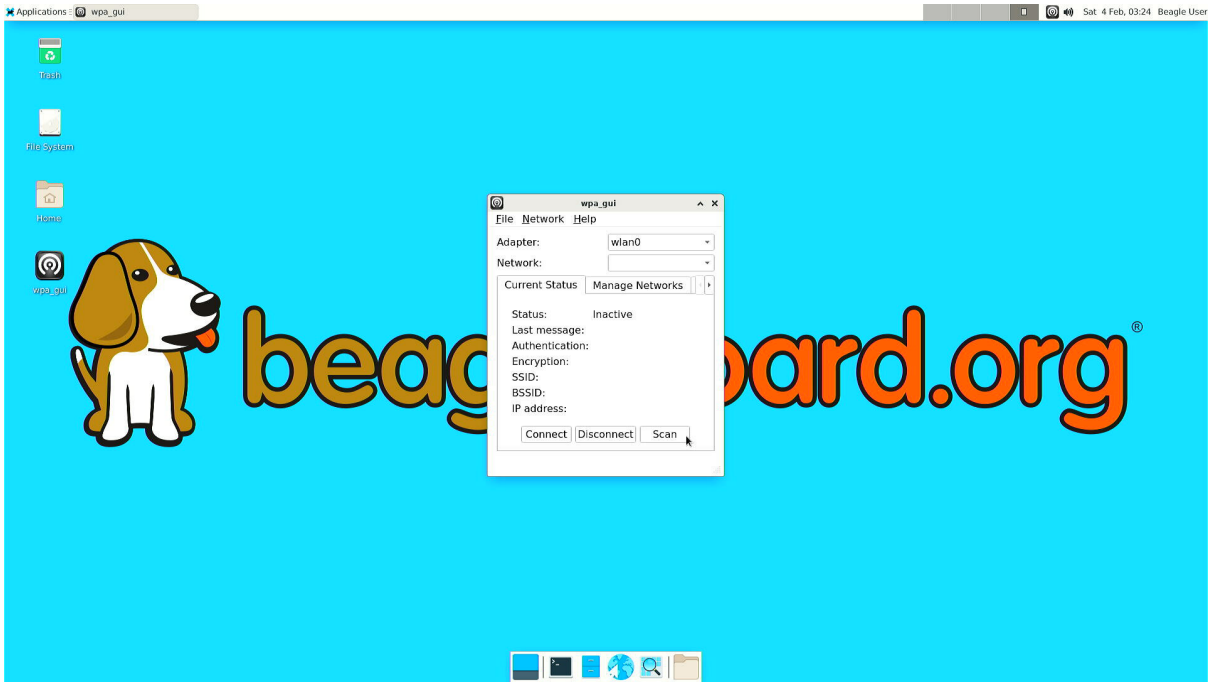


Fig. 2.36: Scanning WiFi access points

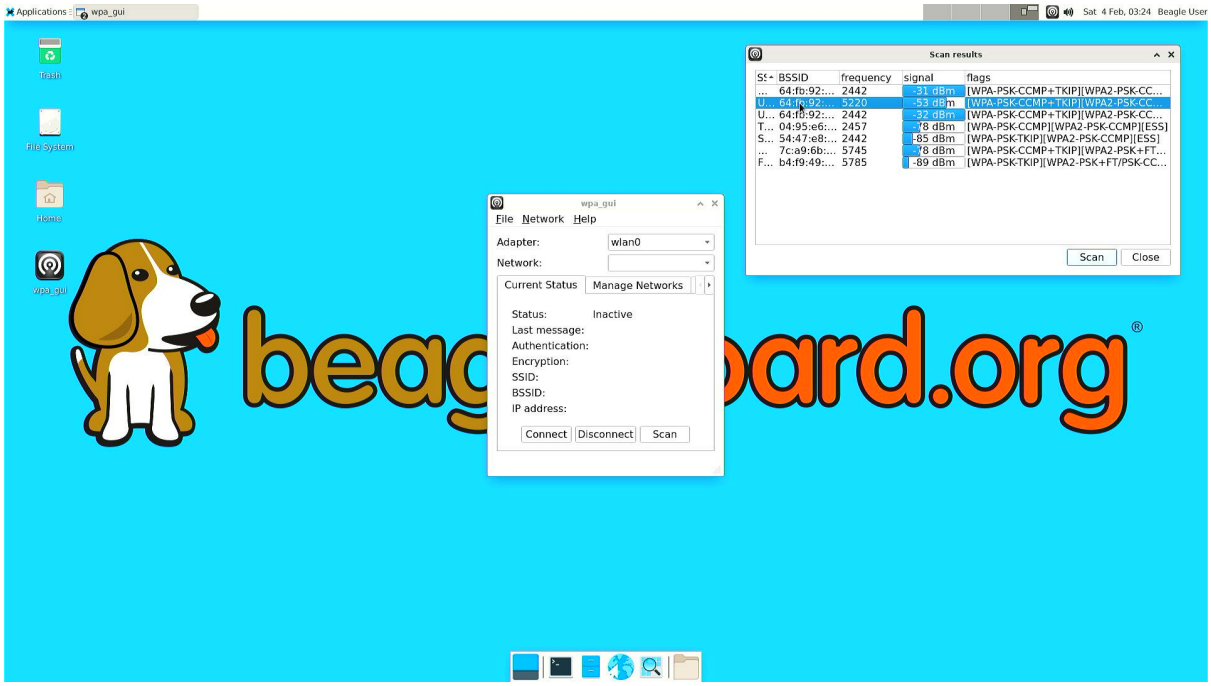


Fig. 2.37: Selecting WiFi access point

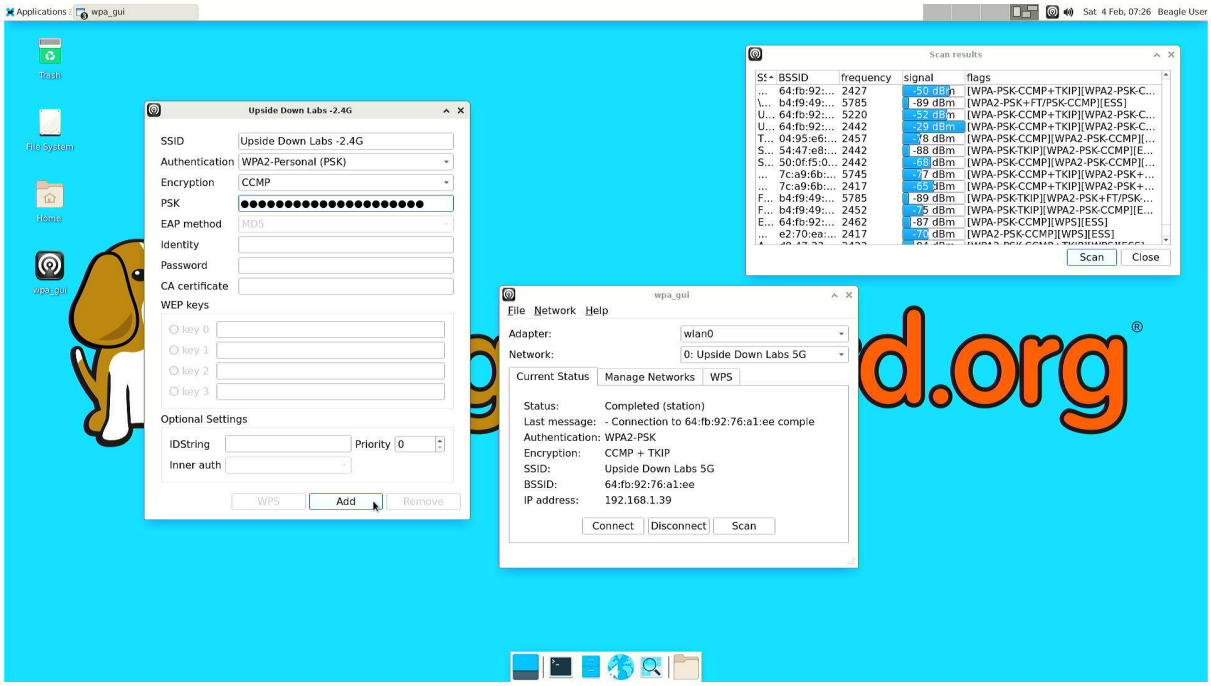


Fig. 2.38: Connecting to WiFi access point

wpa_cli (shell)

Swap out “68:ff:7b:03:0a:8a” and “mypassword” with your network BSSID and password, respectively.

```
debian@BeaglePlay:~$ wpa_cli scan
Selected interface 'wlan0'
OK
debian@BeaglePlay:~$ wpa_cli scan_results
Selected interface 'wlan0'
bssid / frequency / signal level / flags / ssid
68:ff:7b:03:0a:8a    5805    -49     [WPA2-PSK-CCMP][WPS][ESS]       mywifi
debian@BeaglePlay:~$ wpa_cli add_network
Selected interface 'wlan0'
1
debian@BeaglePlay:~$ wpa_cli set_network 1 bssid 68:ff:7b:03:0a:8a
Selected interface 'wlan0'
OK
debian@BeaglePlay:~$ wpa_cli set_network 1 psk '"mypassword"'
Selected interface 'wlan0'
OK
debian@BeaglePlay:~$ wpa_cli enable_network 1
Selected interface 'wlan0'
OK
debian@BeaglePlay:~$ ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.245  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::6e30:2aff:fe29:757d  prefixlen 64  scopeid 0x20<link>
        inet6 2601:408:c083:b6c0::e074  prefixlen 128  scopeid 0x0<global>
        ether 6c:30:2a:29:75:7d  txqueuelen 1000  (Ethernet)
        RX packets 985  bytes 144667 (141.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 10826 (10.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Important: The single quotes around the double quotes are needed to make sure the double quotes are

given to `wpa_cli`. It expects to see them.

Note: For more information about `wpa_cli`, see https://w1.fi/wpa_supplicant/

To make these changes persistent, you need to edit */etc/wpa_supplicant/wpa_supplicant-wlan0.conf*. This is described in *wpa_cli (XFCE)*.

wpa_cli (XFCE)

Another way of connecting to a WiFi access point is to edit the `wpa_supplicant` configuration file.

Step 1: Open up terminal Open up a terminal window either from `Applications > Terminal Emulator` Or from Task Manager.

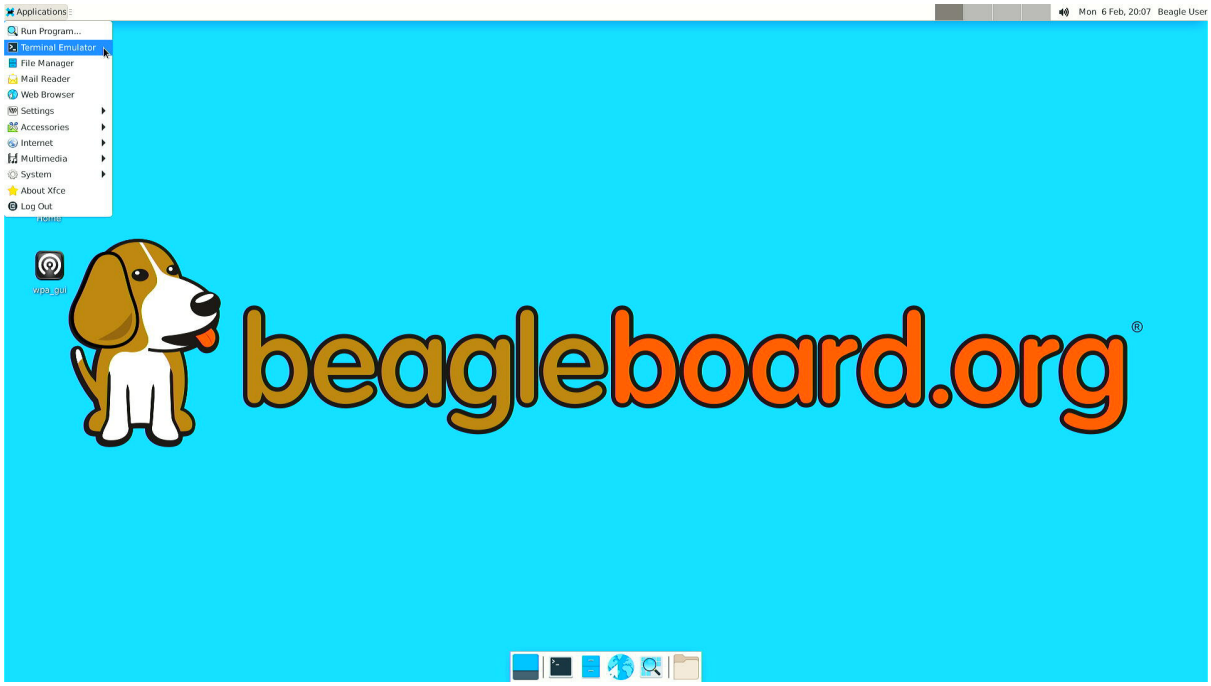


Fig. 2.39: Open terminal from Applications > Terminal Emulator

Step 2: Setup credentials To setup credentials of your WiFi access point follow these steps,

1. Execute `sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan0.conf`, which will open up `wpa_supplicant-wlan0.conf` inside `nano` (terminal based) text editor. 2. Edit `wpa_supplicant-wlan0.conf` to add SSID (WiFi name) & PSK (WiFi password) of your WiFi access point.

```
....
network={
        ssid="WiFi Name"
        psk="WiFi Password"
        ....
}
```

3. Now save the details using `ctrl + O` then enter.
4. To exit out of the `nano` text editor use `ctrl + X`.

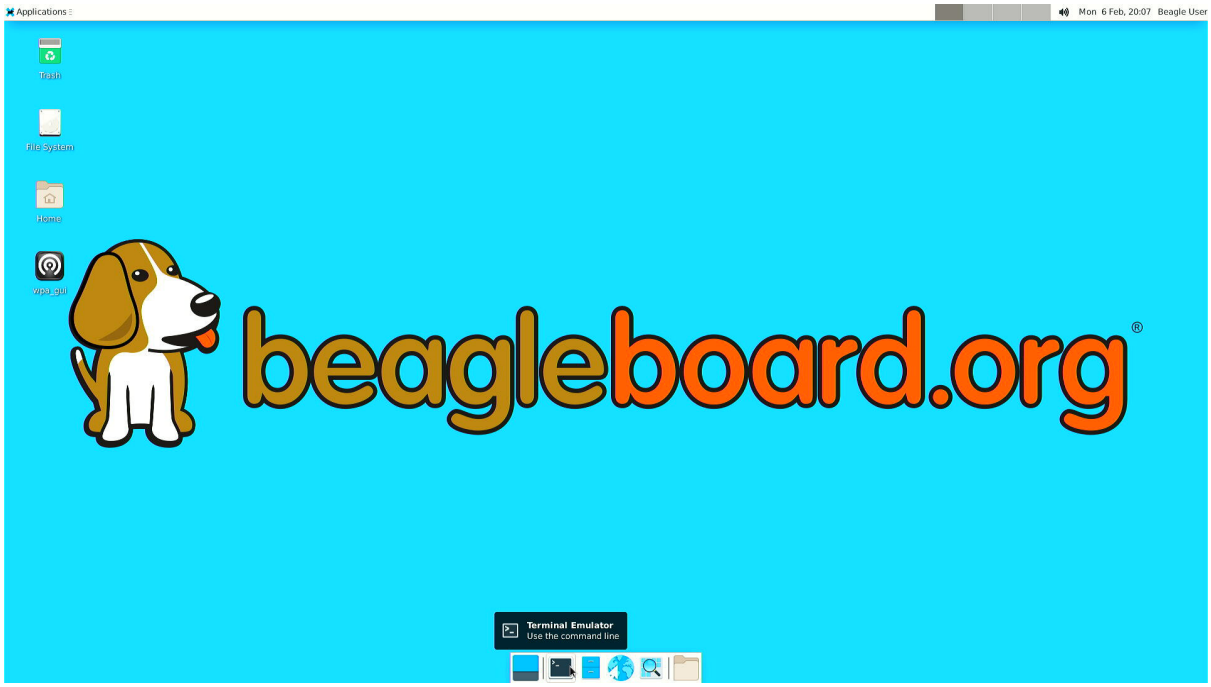


Fig. 2.40: Open terminal from Task Manager

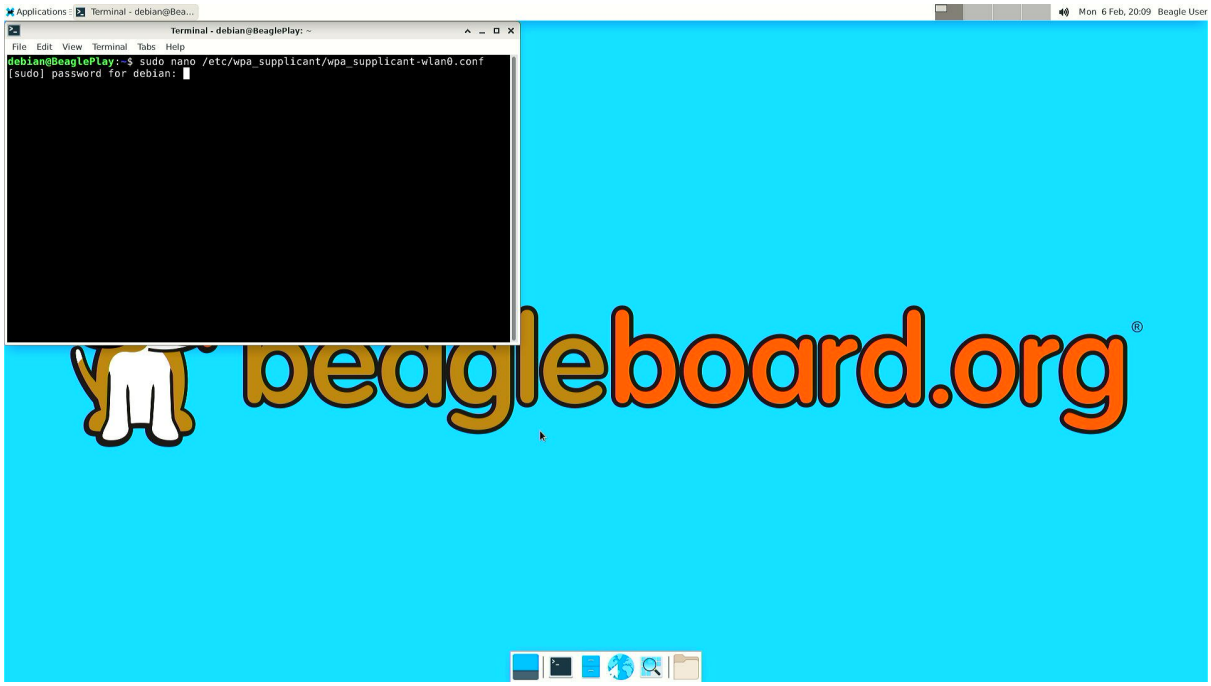


Fig. 2.41: Run: $ sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan0.conf

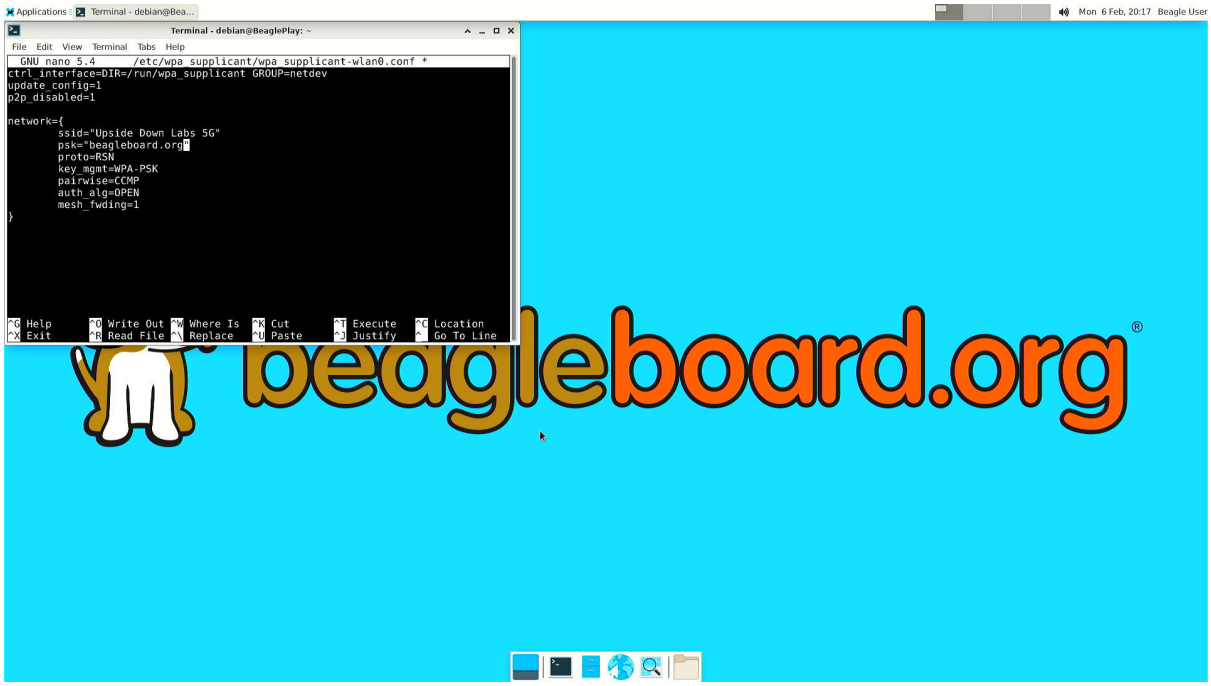


Fig. 2.42: Add SSID and PSK

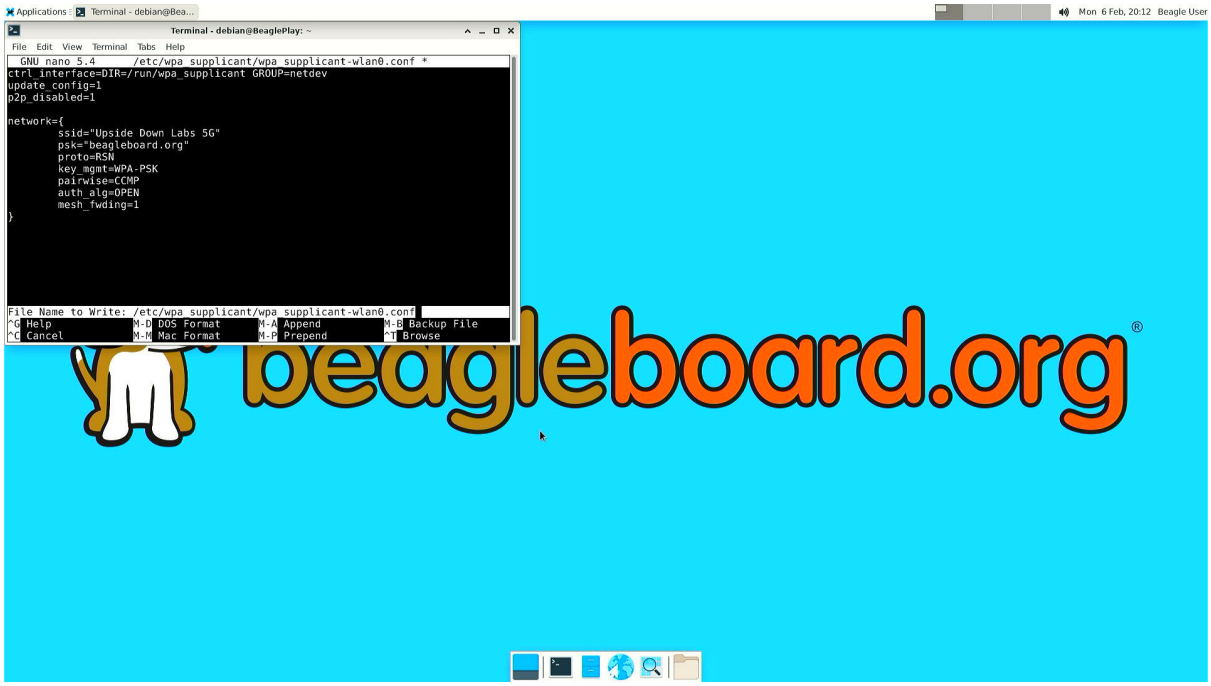


Fig. 2.43: Save credentials (ctrl + O) and Exit (ctrl + X)

Step 3: Reconfigure wlan0 The WiFi doesn't automatically connect to your WiFi access point after you add the credentials to `wpa_supplicant-wlan0.conf`.

1. To connect you can either execute `sudo wpa_cli -i wlan0 reconfigure`
2. Or Reboot your device by executing `reboot` inside your terminal window.
3. Execute `ping 8.8.8.8` to check your connection. Use `ctrl + C` to quit.

```
debian@BeaglePlay:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=5.83 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=7.27 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=5.30 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=5.28 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=118 time=9.04 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=118 time=7.52 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=118 time=5.39 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=118 time=5.94 ms
^C
--- 8.8.8.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7008ms
rtt min/avg/max/mdev = 5.281/6.445/9.043/1.274 ms
```

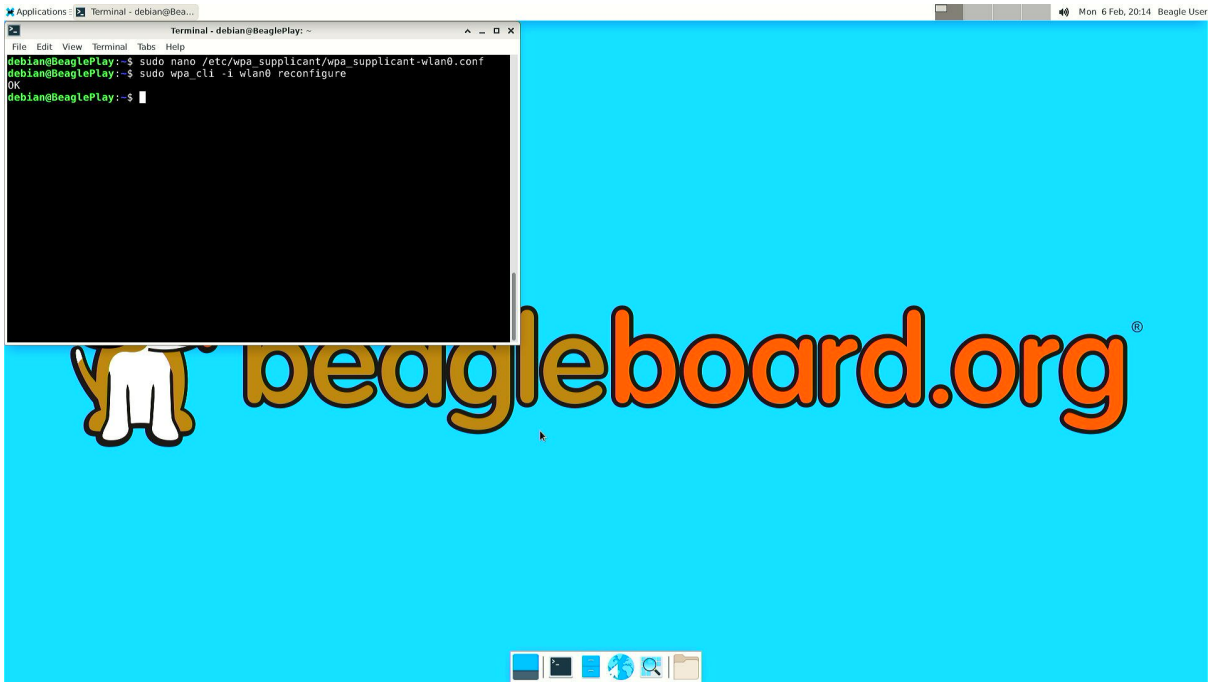


Fig. 2.44: Connect to WiFi by running $ sudo wpa_cli -i wlan0 reconfigure

Disabling the WIFI Access Point

In certain situations, such as running HomeAssistant, you may chose to connect your BeaglePlay to the internet via Ethernet. In this case, it may be desireable to disable it's Wifi access point so that users outside the local network aren't able to connect to it.

The Wifi Access Point that BeaglePlay provides is started using uDev rules. created by the *bb-wlan0-defaults* package

You can simply remove the *bb-wlan0-defaults* package:

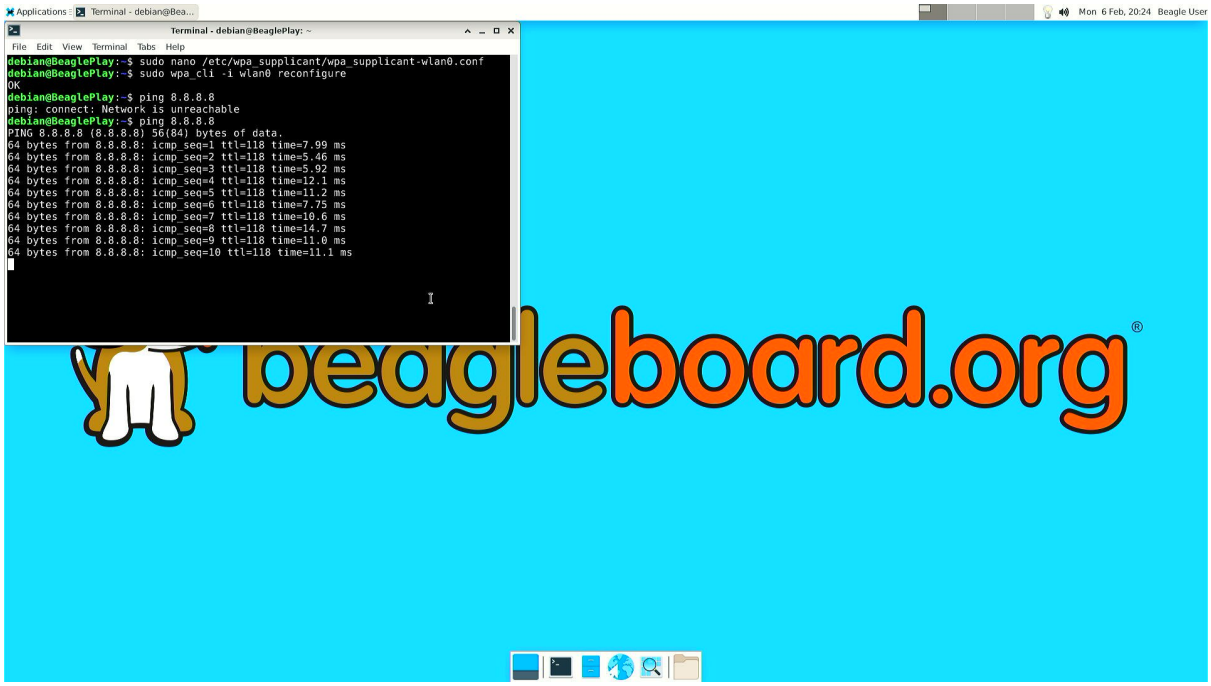


Fig. 2.45: To check connection try running $ ping 8.8.8.8

```
sudo apt remove bb-wlan0-defaults
```

Now just reboot and the Wifi Access point should no longer start.

You can also disable it by removing the two following udev rule files:

```
rm /etc/udev/rules.d/81-add-SoftAp0-interface.rules
rm /etc/udev/rules.d/82-SoftAp0-start-hostpad.rules
```

The issue with doing this latter option is that if you later update your OS, the bb-wlan0-defaults may get updated as well and re-add the rules.

Re-Enabling the WIFI Access Point

Conversely, you can re-enable the access point by re-installing the *bb-wlan0-default* package.

```
sudo apt install bb-wlan0-defaults --reinstall
```

Now just reboot.

–TODO Add notes on changing SSID/Password

2.5.3 Using Grove

See *QWIIC, STEMMA and Grove Add-ons in Linux*.

A link to the appropriate I2C controller can be found at `/dev/play/grove/i2c`.

2.5.4 Using mikroBUS

Steps:

1. Identify if mikroBUS add-on includes a ClickID with `manifest`. If not, `manifest` must be supplied.

2. Identify if mikroBUS add-on is supported by the kernel. If not, kernel module must be added.
3. Identify how driver exposes the data: IIO, net, etc.
4. Connect and power
5. Verify and utilize

Using boards with ClickID

What is mikroBUS? mikroBUS is an open standard for add-on boards for sensors, connectivity, displays, storage and more with over 1,400 available from just a single source, MikroE. With the flexibility of all of the most common embedded serial busses, UART, I2C and SPI, along with ADC, PWM and GPIO functions, it is a great solution for connecting all sorts of electronics.

Note: Learn more at https://www.mikroe.com/mikrobus

What is ClickID? ClickID enables mikroBUS add-on boards to be identified along with the configuration required to use it with the mikroBUS Linux driver. The configuration portion is called a `manifest`.

Note: Learn more at https://github.com/MikroElektronika/click_id

BeaglePlay's Linux kernel is patched with a mikrobus driver that automatically reads the ClickID and loads a driver, greatly simplifying usage.

Does my add-on have ClickID? Look for the "ID" logo on the board. It should be on the side with the pins sticking out, near the AN pin.

Todo: Need an image of the logo

If your add-on has ClickID, simply connect it while BeaglePlay is powered off and then apply power.

Example of examining boot log to see a ClickID was detected.

```
debian@BeaglePlay:~$ dmesg | grep mikrobus
[    2.096254] mikrobus:mikrobus_port_register: registering port mikrobus-0
[    2.096325] mikrobus mikrobus-0: mikrobus port 0 eeprom empty probing␣
↪default eeprom
[    2.663698] mikrobus_manifest:mikrobus_manifest_attach_device: parsed␣
↪device 1, driver=opt3001, protocol=3, reg=44
[    2.663711] mikrobus_manifest:mikrobus_manifest_parse:  Ambient 2 Click␣
↪manifest parsed with 1 devices
[    2.663783] mikrobus mikrobus-0: registering device : opt3001
```

To use the add-on, see *Using boards with Linux drivers*.

Note: Not all Click boards with ClickID have valid `manifest` entries.

What if my add-on doesn't have ClickID?

It is still possible a `manifest` has been created for your add-on as we have created over 100 of them. You can install the existing manifest files onto your BeaglePlay.

First, make sure you have the latest manifests installed in your system.

```
sudo apt update
sudo apt install bbb.io-clickid-manifests
```

Take a look at the list of `manifest` files to see if the Click or other mikrobus add-on board `manifest` is installed.

```
debian@BeaglePlay:~$ ls /lib/firmware/mikrobus/
10DOF-CLICK.mnfb          COMPASS-2-CLICK.mnfb       I2C-2-SPI-CLICK.mnfb   ␣
↪    PWM-CLICK.mnfb
13DOF-2-CLICK.mnfb        COMPASS-CLICK.mnfb         I2C-MUX-CLICK.mnfb     ␣
↪    RFID-CLICK.mnfb
3D-HALL-3-CLICK.mnfb      CURRENT-CLICK.mnfb         ILLUMINANCE-CLICK.mnfb ␣
↪    RF-METER-CLICK.mnfb
3D-HALL-6-CLICK.mnfb      DAC-7-CLICK.mnfb           IR-GESTURE-CLICK.mnfb  ␣
↪    RMS-TO-DC-CLICK.mnfb
6DOF-IMU-2-CLICK.mnfb     DAC-CLICK.mnfb             IR-THERMO-2-CLICK.mnfb ␣
↪    RTC-6-CLICK.mnfb
6DOF-IMU-4-CLICK.mnfb     DIGIPOT-3-CLICK.mnfb       LED-DRIVER-7-CLICK.mnfb␣
↪    SHT1x-CLICK.mnfb
6DOF-IMU-6-CLICK.mnfb     DIGIPOT-CLICK.mnfb         LIGHTRANGER-2-CLICK.
↪mnfb    SHT-CLICK.mnfb
6DOF-IMU-8-CLICK.mnfb     EEPROM-2-CLICK.mnfb        LIGHTRANGER-3-CLICK.
↪mnfb    SMOKE-CLICK.mnfb
9DOF-CLICK.mnfb           EEPROM-3-CLICK.mnfb        LIGHTRANGER-CLICK.mnfb ␣
↪    TEMP-HUM-11-CLICK.mnfb
ACCEL-3-CLICK.mnfb        EEPROM-CLICK.mnfb          LPS22HB-CLICK.mnfb     ␣
↪    TEMP-HUM-12-CLICK.mnfb
ACCEL-5-CLICK.mnfb        ENVIRONMENT-CLICK.mnfb     LSM303AGR-CLICK.mnfb   ␣
↪    TEMP-HUM-3-CLICK.mnfb
ACCEL-6-CLICK.mnfb        ETH-CLICK.mnfb             LSM6DSL-CLICK.mnfb     ␣
↪    TEMP-HUM-4-CLICK.mnfb
ACCEL-8-CLICK.mnfb        ETH-WIZ-CLICK.mnfb         MAGNETIC-LINEAR-CLICK.
↪mnfb  TEMP-HUM-7-CLICK.mnfb
ACCEL-CLICK.mnfb          FLASH-2-CLICK.mnfb         MAGNETIC-ROTARY-CLICK.
↪mnfb  TEMP-HUM-9-CLICK.mnfb
ADC-2-CLICK.mnfb          FLASH-CLICK.mnfb           MICROSD-CLICK.mnfb     ␣
↪    TEMP-HUM-CLICK.mnfb
ADC-3-CLICK.mnfb          GENERIC-SPI-CLICK.mnfb     MPU-9DOF-CLICK.mnfb    ␣
↪    TEMP-LOG-3-CLICK.mnfb
ADC-5-CLICK.mnfb          GEOMAGNETIC-CLICK.mnfb     MPU-IMU-CLICK.mnfb     ␣
↪    TEMP-LOG-4-CLICK.mnfb
ADC-8-CLICK.mnfb          GNSS-4-CLICK.mnfb          NO2-2-CLICK.mnfb       ␣
↪    TEMP-LOG-6-CLICK.mnfb
ADC-CLICK.mnfb            GNSS-7-CLICK.mnfb          NO2-CLICK.mnfb         ␣
↪    THERMO-12-CLICK.mnfb
AIR-QUALITY-2-CLICK.mnfb  GNSS-ZOE-CLICK.mnfb        OLEDB-CLICK.mnfb       ␣
↪    THERMO-15-CLICK.mnfb
AIR-QUALITY-3-CLICK.mnfb  GSR-CLICK.mnfb             OLEDC-CLICK.mnfb       ␣
↪    THERMO-17-CLICK.mnfb
AIR-QUALITY-5-CLICK.mnfb  GYRO-2-CLICK.mnfb          OLEDW-CLICK.mnfb       ␣
↪    THERMO-3-CLICK.mnfb
ALCOHOL-2-CLICK.mnfb      GYRO-CLICK.mnfb            OZONE-2-CLICK.mnfb     ␣
↪    THERMO-4-CLICK.mnfb
ALCOHOL-3-CLICK.mnfb      HALL-CURRENT-2-CLICK.mnfb  PRESSURE-11-CLICK.mnfb ␣
↪    THERMO-7-CLICK.mnfb
ALTITUDE-3-CLICK.mnfb     HALL-CURRENT-3-CLICK.mnfb  PRESSURE-3-CLICK.mnfb  ␣
↪    THERMO-8-CLICK.mnfb
ALTITUDE-CLICK.mnfb       HALL-CURRENT-4-CLICK.mnfb  PRESSURE-4-CLICK.mnfb  ␣
↪    THERMO-CLICK.mnfb
AMBIENT-2-CLICK.mnfb      HDC1000-CLICK.mnfb         PRESSURE-CLICK.mnfb    ␣
↪    THERMOSTAT-3-CLICK.mnfb
AMBIENT-4-CLICK.mnfb      HEART-RATE-3-CLICK.mnfb    PROXIMITY-10-CLICK.mnfb␣
↪    UV-3-CLICK.mnfb
```

(continues on next page)

(continued from previous page)

```
AMBIENT-5-CLICK.mnfb       HEART-RATE-4-CLICK.mnfb    PROXIMITY-2-CLICK.mnfb ␣
↪    VACUUM-CLICK.mnfb
AMMETER-CLICK.mnfb         HEART-RATE-5-CLICK.mnfb    PROXIMITY-5-CLICK.mnfb ␣
↪    VOLTMETER-CLICK.mnfb
COLOR-2-CLICK.mnfb         HEART-RATE-7-CLICK.mnfb    PROXIMITY-9-CLICK.mnfb ␣
↪    WAVEFORM-CLICK.mnfb
COLOR-7-CLICK.mnfb         HEART-RATE-CLICK.mnfb      PROXIMITY-CLICK.mnfb   ␣
↪    WEATHER-CLICK.mnfb
```

Then, load the appropriate `manifest` using the `mikrobus` bus driver. For example, with the Ambient 2 Click, you can write that `manifest` to the `mikrobus-0` `new_device` entry.

```
cat /lib/firmware/mikrobus/AMBIENT-2-CLICK.mnfb > /sys/bus/mikrobus/devices/
↪mikrobus-0/new_device
```

Note: We will be adding a link to the `mikrobus-0` device at `/dev/play/mikrobus` in the near future, but you can find it for now at `/sys/bus/mikrobus/devices/mikrobus-0`. If you need to supply an ID (manifest), this is the directory where you will do it.

Manifesto: https://git.beagleboard.org/beagleconnect/manifesto

Patched Linux with out-of-tree Mikrobus driver: https://git.beagleboard.org/beagleboard/linux

---

Note: It'll forget on reboot... need to have a boot service.

Todo: To make it stick, ...

To use the add-on, see *Using boards with Linux drivers*.

Using boards with Linux drivers

Depending on the type of mikrobus add-on board, the Linux driver could be of various different types. For sensors, the most common is *IIO driver*.

IIO driver Per https://docs.kernel.org/driver-api/iio/intro.html,

> The main purpose of the Industrial I/O subsystem (IIO) is to provide support for devices that in some sense perform either analog-to-digital conversion (ADC) or digital-to-analog conversion (DAC) or both. The aim is to fill the gap between the somewhat similar hwmon and input subsystems. Hwmon is directed at low sample rate sensors used to monitor and control the system itself, like fan speed control or temperature measurement. Input is, as its name suggests, focused on human interaction input devices (keyboard, mouse, touchscreen). In some cases there is considerable overlap between these and IIO.
>
> Devices that fall into this category include:
>
> - analog to digital converters (ADCs)
> - accelerometers
> - capacitance to digital converters (CDCs)
> - digital to analog converters (DACs)
> - gyroscopes
> - inertial measurement units (IMUs)

- color and light sensors
- magnetometers
- pressure sensors
- proximity sensors
- temperature sensors

See also https://wiki.analog.com/software/linux/docs/iio/iio.

To discover IIO driver enabled devices, use the `iio_info` command.

```
debian@BeaglePlay:~$ iio_info
Library version: 0.24 (git tag: v0.24)
Compiled with backends: local xml ip usb
IIO context created with local backend.
Backend version: 0.24 (git tag: v0.24)
Backend description string: Linux BeaglePlay 5.10.168-ti-arm64-r104
↪#1bullseye SMP Thu Jun 8 23:07:22 UTC 2023 aarch64
IIO context has 2 attributes:
        local,kernel: 5.10.168-ti-arm64-r104
        uri: local:
IIO context has 2 devices:
        iio:device0: opt3001
                1 channels found:
                    illuminance:  (input)
                        2 channel-specific attributes found:
                                attr  0: input value: 163.680000
                                attr  1: integration_time value: 0.800000
                2 device-specific attributes found:
                                attr  0: current_timestamp_clock value:␣
↪realtime
                                attr  1: integration_time_available value: 0.
↪1 0.8
                No trigger on this device
        iio:device1: adc102s051
                2 channels found:
                        voltage1:  (input)
                        2 channel-specific attributes found:
                                attr  0: raw value: 4084
                                attr  1: scale value: 0.805664062
                        voltage0:  (input)
                        2 channel-specific attributes found:
                                attr  0: raw value: 2440
                                attr  1: scale value: 0.805664062
                No trigger on this device
```

Note that the units are standardized for the IIO interface based on the device type. If raw values are provided, a scale must be applied to get to the standardized units.

Storage driver

Network driver

How does ClickID work?

Disabling the mikroBUS driver

If you'd like to use other means to control the mikroBUS connector, you might want to disable the mikroBUS driver. This is most easily done by enabling a deivce tree overlay at boot.

Todo: Document kernel version that integrates this overlay and where to get update instructions.

Note: To utilize the overlay with these instructions, make sure to have TBD version of kernel, modules and firmware installed. Use *uname -a* to determine the currently running kernel version. See TBD for information on how to update.

Apply overlay to disable mikrobus0 instance.

```
echo "    fdtoverlays /overlays/k3-am625-beagleplay-release-mikrobus.dtbo" |␣
↪sudo tee -a /boot/firmware/extlinux/extlinux.conf
sudo shutdown -r now
```

Log back in after reboot and verify the device driver did not capture the busses.

```
debian@BeaglePlay:~$ ls /dev/play
grove  mikrobus  qwiic
debian@BeaglePlay:~$ ls /dev/play/mikrobus/
i2c
debian@BeaglePlay:~$ ls /sys/bus/mikrobus/devices/
debian@BeaglePlay:~$ ls /proc/device-tree/chosen/overlays/
k3-am625-beagleplay-release-mikrobus  name
debian@BeaglePlay:~$
```

To re-enable.

```
sudo sed -e '/release-mikrobus/ s/^#*/#/' -i /boot/firmware/extlinux/
↪extlinux.conf
sudo shutdown -r now
```

Verify driver is enabled again.

```
debian@BeaglePlay:~$ ls /sys/bus/mikrobus/devices/
mikrobus-0
debian@BeaglePlay:~$ ls /proc/device-tree/chosen/overlays/
ls: cannot access '/proc/device-tree/chosen/overlays/': No such file or␣
↪directory
debian@BeaglePlay:~$
```

Todo:

- How do turn off the driver?
- How do turn on spidev?
- How do I enable GPIO?
- How do a provide a manifest?

Todo:

- Needs udev
- Needs live description

2.5.5 Using QWIIC

See *QWIIC, STEMMA and Grove Add-ons in Linux*.

A link to the appropriate I2C controller can be found at `/dev/play/qwiic/i2c`.

2.5.6 Using OLDI Displays

2.5.7 Using CSI Cameras

2.5.8 Wireless MCU Zephyr Development

BeaglePlay includes a Texas Instruments CC1352P7 wireless microcontroller (MCU) that can be programmed using the Linux Foundation Zephyr RTOS.

Developing directly in Zephyr will not be ultimately required for end-users who won't touch the firmware running on the CC1352 on BeaglePlay™ and will instead use the provided wireless functionality. However, it is important for early adopters as well as people looking to extend the functionality of the open source design. If you are one of those people, this is a good place to get started.

Further, BeaglePlay is a reasonable development platform for creating Zephyr-based applications for *BeagleConnect Freedom*. The same Zephyr development environment setup here is also described for targeting applications on that board.

Install the latest software image for BeaglePlay

Note: These instructions should be generic for BeaglePlay and other boards and only the specifics of which image was used to test these instructions need be included here moving forward and the detailed instructions can be referenced elsewhere.

You may want to download and install the latest Debian Linux operating system image for BeaglePlay.

Note: These instructions were validated with the BeagleBoard.org Debian image BeaglePlay Debian 11.6 Flasher 2023-03-10.

---

1. Load this image to a microSD card using a tool like Etcher.
2. Insert the microSD card into BeaglePlay.
3. Power BeaglePlay via the USB-C connector.
4. Wait for the LEDs to start blinking, then turn off.
5. Remove power from BeaglePlay.
6. *IMPORTANT* Remove microSD card from BeaglePlay.
7. Apply power to BeaglePlay.

Note: This will flash the CC1352 as well as the eMMC flash on BeaglePlay.

Todo: Describe how to know it is working

Log into BeaglePlay

Please either plug in a keyboard, monitor and mouse or `ssh` into the board. We can point somewhere else for instructions on this. You can also point your web browser to the board to log into the Visual Studio Code IDE environment.

Todo: A big part of what is missing here is to put your BeaglePlay on the Internet such that we can download things in later steps. That has been initially brushed over.

Flash existing IEEE 802.15.4 radio bridge (WPANUSB) firmware

If you've recieved a board fresh from the factory, this is already done and not necessary, unless you want to restore the contents back to the factory condition.

Background This *WPANUSB* application was originally developed for radio devices with a USB interface. The CC1352P7 does not have a USB device, so the application was modified to communicate over a UART serial interface.

For the *BeagleConnect Freedom*, a USB-to-UART bridge device was used and the USB endpoints were made compatible with the WPANUSB linux driver which we augmented to support this board. To utilize the existing *WPANUSB* Zephyr application and this Linux driver, we chose to encode our UART traffic with HDLC. This has the advantage of enabing a serial console interface to the Zephyr shell while WPANUSB-specific traffic is directed to other USB endpoints.

For BeaglePlay, the USB-to-UART bridge is not used, but we largely kept the same *WPANUSB* application, including the HDLC encoding.

Note: Now you know why this WPAN bridge application is called *WPANUSB*, even though USB isn't used!

Steps

1. Ensure the *bcfserial* driver isn't blocking the serial port.

```
echo "    fdtoverlays /overlays/k3-am625-beagleplay-bcfserial-no-
↪firmware.dtbo" | sudo tee -a /boot/firmware/extlinux/extlinux.
↪conf
sudo shutdown -r now
```

Note: The default password is *temppwd*.

2. Download and flash the *WPANUSB* Zephyr application firmware onto the CC1352P7 on BeaglePlay from the releases on git.beagleboard.org or distros on www.beagleboard.org/distros.

```
cd
wget https://files.beagle.cc/file/beagleboard-public-2021/images/
↪download
unzip download
build/play/cc2538-bsl.py build/play/wpanusb
```

Note: I got a *File Not Found* error on the `wget` command above. If it doesn't work for you try:

```
bone:~$ wget https://files.beagle.cc/file/beagleboard-public-2021/
↪images/zephyr-beagle-cc1352-0.2.2.zip
bone:~$ unzip zephyr-beagle-cc1352-0.2.2.zip
bone:~$ build/play/cc2538-bsl.py build/play/wpanusb
```

3. Ensure the *bcfserial* driver is set to load.

```
sudo sed -e '/bcfserial-no-firmware/ s/^#*/#/' -i /boot/firmware/
↪extlinux/extlinux.conf
sudo shutdown -r now
```

4. Verify the the 6LoWPAN network is up.

```
debian@BeaglePlay:~$ lsmod | grep bcfserial
bcfserial              24576  0 ①
mac802154              77824  2 wpanusb,bcfserial
debian@BeaglePlay:~$ ifconfig
SoftAp0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.8.1  netmask 255.255.255.0  broadcast 192.
↪168.8.255
        inet6 fe80::3ee4:b0ff:fe7e:b5f7  prefixlen 64  scopeid␣
↪0x20<link>
        ether 3c:e4:b0:7e:b5:f7  txqueuelen 1000  (Ethernet)
        RX packets 4046  bytes 576780 (563.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4953  bytes 5116336 (4.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.
↪255.255
        ether 02:42:f8:29:41:69  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether f4:84:4c:fc:5d:13  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 246239  bytes 19948296 (19.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 246239  bytes 19948296 (19.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

lowpan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1280 ②
        inet6 fe80::200:0:0:0  prefixlen 64  scopeid 0x20<link> ③
        inet6 2001:db8::2  prefixlen 64  scopeid 0x0<global> ④
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 ␣
↪txqueuelen 1000  (UNSPEC)
        RX packets 107947  bytes 6629290 (6.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2882  bytes 179511 (175.3 KiB) ⑤
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

(continues on next page)

(continued from previous page)

```
        inet 192.168.7.2  netmask 255.255.255.0  broadcast 192.
↪168.7.255
        inet6 fe80::1eba:8cff:fea2:ed6b  prefixlen 64  scopeid␣
↪0x20<link>
        ether 1c:ba:8c:a2:ed:6b  txqueuelen 1000  (Ethernet)
        RX packets 9858  bytes 2638440 (2.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4155  bytes 1454082 (1.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

usb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.6.2  netmask 255.255.255.0  broadcast 192.
↪168.6.255
        inet6 fe80::1eba:8cff:fea2:ed6d  prefixlen 64  scopeid␣
↪0x20<link>
        ether 1c:ba:8c:a2:ed:6d  txqueuelen 1000  (Ethernet)
        RX packets 469614  bytes 35385636 (33.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 365548  bytes 66523708 (63.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.161  netmask 255.255.255.0  broadcast 192.
↪168.0.255
        inet6 fe80::3ee4:b0ff:fe7e:b5f6  prefixlen 64  scopeid␣
↪0x20<link>
        inet6 2601:408:c083:b6c0::d00d  prefixlen 128  scopeid␣
↪0x0<global>
        ether 3c:e4:b0:7e:b5:f6  txqueuelen 1000  (Ethernet)
        RX packets 3188898  bytes 678154090 (646.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1162074  bytes 293237366 (279.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

wpan0: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 123 ⑥
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 ␣
↪txqueuelen 300  (UNSPEC)
        RX packets 108495  bytes 2539160 (2.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2888  bytes 140523 (137.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0
```

① You'll want to see that the *bcfserial* driver has been loaded.

② There should be a *lowpan0* interface.

③ There should be a link-local address for *lowpan0*.

④ There should be a global address for *lowpan0*.

⑤ Seeing some packets have been transmitted can give you some confidence.

⑥ The *wpan0* interface should be there, but we have a 6LoWPAN adapter on top of it.

Note: You may find Linux-WPAN.org useful.

Setup Zephyr development on BeaglePlay

1. Download and setup Zephyr for BeaglePlay

```
cd
sudo apt update
sudo apt install --no-install-recommends -y \
    gperf \
    ccache dfu-util \
    libsdl2-dev \
    libxml2-dev libxslt1-dev libssl-dev libjpeg62-turbo-dev␣
↪libmagic1 \
    libtool-bin autoconf automake libusb-1.0-0-dev \
    python3-tk python3-virtualenv
wget https://github.com/zephyrproject-rtos/sdk-ng/releases/
↪download/v0.15.1/zephyr-sdk-0.15.1_linux-aarch64_minimal.tar.gz
tar xf zephyr-sdk-0.15.1_linux-aarch64_minimal.tar.gz
rm zephyr-sdk-0.15.1_linux-aarch64_minimal.tar.gz
./zephyr-sdk-0.15.1/setup.sh -t arm-zephyr-eabi -c
west init -m https://git.beagleboard.org/beagleconnect/zephyr/
↪zephyr --mr sdk zephyr-beagle-cc1352-sdk
cd $HOME/zephyr-beagle-cc1352-sdk
python3 -m virtualenv zephyr-beagle-cc1352-env
echo "export ZEPHYR_TOOLCHAIN_VARIANT=zephyr" >> $HOME/zephyr-
↪beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/activate
echo "export ZEPHYR_SDK_INSTALL_DIR=$HOME/zephyr-sdk-0.15.1" >>
↪$HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
echo "export ZEPHYR_BASE=$HOME/zephyr-beagle-cc1352-sdk/zephyr" >
↪> $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
echo 'export PATH=$HOME/zephyr-beagle-cc1352-sdk/zephyr/scripts:
↪$PATH' >> $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-
↪env/bin/activate
echo "export BOARD=beagleplay" >> $HOME/zephyr-beagle-cc1352-sdk/
↪zephyr-beagle-cc1352-env/bin/activate
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/
↪bin/activate
west update
west zephyr-export
pip3 install -r zephyr/scripts/requirements-base.txt
```

2. Activate the Zephyr build environment

   If you exit and come back, you'll need to reactivate your Zephyr build environment.

```
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/
↪bin/activate
```

3. Verify Zephyr setup for BeaglePlay

```
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ cmake --version
cmake version 3.22.1

CMake suite maintained and supported by Kitware (kitware.com/
↪cmake).
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ python3 --version
Python 3.9.2
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ dtc --version
Version: DTC 1.6.0
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ west --version
West version: v0.14.0
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ ./zephyr-sdk-0.15.1/
↪arm-zephyr-eabi/bin/arm-zephyr-eabi-gcc --version
```

(continues on next page)

(continued from previous page)

```
arm-zephyr-eabi-gcc (Zephyr SDK 0.15.1) 12.1.0
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. ␣
↪There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A␣
↪PARTICULAR PURPOSE.
```

Build applications for BeaglePlay CC1352

Now you can build various Zephyr applications

1. Build and flash Blinky example

```
cd $HOME/zephyr-beagle-cc1352-sdk/zephyr
west build -d build/play_blinky samples/basic/blinky
west flash -d build/play_blinky
```

2. Try out Micropython

```
cd
git clone -b beagleplay-cc1352 https://git.beagleboard.org/
↪beagleplay/micropython
cd micropython
west build -d play ports/zephyr
west flash -d play
tio /dev/ttyS4
```

Build applications for BeagleConnect Freedom

1. Build and flash Blinky example

```
cd $HOME/zephyr-beagle-cc1352-sdk/zephyr
west build -d build/freedom_blinky -b beagleconnect_freedom␣
↪samples/basic/blinky
west flash -d build/freedom_blinky
```

2. Try out Micropython

```
cd
git clone -b beagleplay-cc1352 https://git.beagleboard.org/
↪beagleplay/micropython
cd micropython
west build -d freedom -b beagleconnect_freedom ports/zephyr
west flash -d freedom
tio /dev/ttyACM0
```

Important: Nothing below here is tested

---

1. TODO

```
west build -d build/sensortest zephyr/samples/boards/beagle_bcf/
↪sensortest -- -DOVERLAY_CONFIG=overlay-subghz.conf
```

2. TODO

```
west build -d build/wpanusb modules/lib/wpanusb_bc -- -DOVERLAY_
↪CONFIG=overlay-subghz.conf
```

3. TODO

```
west build -d build/bcfserial modules/lib/wpanusb_bc -- -
↪DOVERLAY_CONFIG=overlay-bcfserial.conf -DDTC_OVERLAY_
↪FILE=bcfserial.overlay
```

4. TODO

```
west build -d build/greybus modules/lib/greybus/samples/subsys/
↪greybus/net -- -DOVERLAY_CONFIG=overlay-802154-subg.conf
```

Flash applications to BeagleConnect Freedom And then you can flash the BeagleConnect Freedom boards over USB

1. **Make sure you are in Zephyr directory**

```
cd $HOME/bcf-zephyr
```

2. **Flash Blinky**

```
cc2538-bsl.py build/blinky
```

Debug applications over the serial terminal

Todo: Describe how to handle the serial connection

2.5.9 Introduction

This guide is for all those who want to kick start their kernel development journey on the TI AM625x SoC Based BeaglePlay.

Getting the Kernel Source Code

The Linux kernel is hosted on a number of servers around the world. The main repository is hosted on the kernel.org website, but there are also mirrors hosted by other organizations, such as GitHub and Bootlin.

The Linux Torvalds tree is the most up-to-date source of the Linux kernel. It is used by Linux distributions and other projects to build their own kernels. The tree is also a popular destination for kernel developers who want to contribute to the kernel.

Kernel sources can directly be fetched from GIT:

```
git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

Preparing to Build

Todo: Add Compiler download link and env setup instructions

Configuring the Kernel

Todo: Add steps to config kernel for play

Sources

This documentation is heavily based on the official TI-SDK documentation for AM62X

2.5.10 BeagleConnect™ Greybus demo using BeagleConnect™ Freedom and BeaglePlay

BeaglePlay CC1352 Firmware

Build (Download and Setup Zephyr for BeaglePlay)

1. Install prerequisites

```
cd
sudo apt update
sudo apt install --no-install-recommends -y \
    gperf \
    ccache dfu-util \
    libsdl2-dev \
    libxml2-dev libxslt1-dev libssl-dev libjpeg62-turbo-dev␣
↪libmagic1 \
    libtool-bin autoconf automake libusb-1.0-0-dev \
    python3-tk python3-virtualenv
```

2. Download the latest Zephyr Release, extract it and cleanup

```
sudo wget https://github.com/zephyrproject-rtos/sdk-ng/releases/
↪download/v0.16.3/zephyr-sdk-0.16.3_linux-aarch64_minimal.tar.xz
tar xf zephyr-sdk-0.16.3_linux-aarch64_minimal.tar.xz
rm zephyr-sdk-0.16.3_linux-aarch64_minimal.tar.xz
```

3. Run the Zephyr SDK Setup Script

```
./zephyr-sdk-0.16.3/setup.sh -t arm-zephyr-eabi -c
```

4. Download and Initialize West. (Zephyr's meta-tool)

Note: You may want to add */home/debian/.local/bin* to your *.bashrc* file to make the West command available after a reboot

---

```
pip3 install --user -U west
export PATH="/home/debian/.local/bin:$PATH"
west init -m https://git.beagleboard.org/beagleconnect/zephyr/
↪zephyr --mr sdk-next zephyr-beagle-cc1352-sdk
cd $HOME/zephyr-beagle-cc1352-sdk
```

5. Setup a Python Virtual Environment and add our PATH Variables

```
virtualenv zephyr-beagle-cc1352-env
echo "export ZEPHYR_TOOLCHAIN_VARIANT=zephyr" >> $HOME/zephyr-
↪beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/activate
echo "export ZEPHYR_SDK_INSTALL_DIR=$HOME/zephyr-sdk-0.16.3" >>
↪$HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
echo "export ZEPHYR_BASE=$HOME/zephyr-beagle-cc1352-sdk/zephyr" >
↪> $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
echo 'export PATH=$HOME/zephyr-beagle-cc1352-sdk/zephyr/scripts:
↪$PATH' >> $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-
↪env/bin/activate
```

(continues on next page)

(continued from previous page)

```
echo "export BOARD=beagleplay_cc1352" >> $HOME/zephyr-beagle-
↪cc1352-sdk/zephyr-beagle-cc1352-env/bin/activate
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/
↪bin/activate
```

6. Update West

```
west update
west zephyr-export
```

7. Install Python Prerequisites

```
pip3 install -r zephyr/scripts/requirements-base.txt
```

8. Activate the Zephyr build environment

   NOTE - If you exit and come back, you'll need to reactivate your Zephyr build environment.

```
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/
↪bin/activate
```

9. Verify Zephyr setup for BeaglePlay

```
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ cmake --version
cmake version 3.22.1

CMake suite maintained and supported by Kitware (kitware.com/
↪cmake).

(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ python3 --version
Python 3.9.2

(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ dtc --version
Version: DTC 1.6.0

(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ west --version
West version: v0.14.0

(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ ./zephyr-sdk-0.16.3/
↪arm-zephyr-eabi/bin/arm-zephyr-eabi-gcc --version
arm-zephyr-eabi-gcc (Zephyr SDK 0.16.3) 12.1.0

Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. ␣
↪There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A␣
↪PARTICULAR PURPOSE.
```

10. Clone CC1352 Firmware at top level: https://git.beagleboard.org/gsoc/greybus/cc1352-firmware

```
cd ~
git clone https://git.beagleboard.org/gsoc/greybus/cc1352-
↪firmware
```

11. Build the Firmware

```
west build -b beagleplay_cc1352 -p always cc1352-firmware
```

12. You can now find the built firmware at *build/zephyr/zephyr.bin*

Flash

1. Ensure the *gb-beagleplay* driver isn't blocking the serial port.

```
debian@BeaglePlay:~$ echo "    fdtoverlays /overlays/k3-am625-
↪beagleplay-bcfserial-no-firmware.dtbo" | sudo tee -a /boot/
↪firmware/extlinux/extlinux.conf
debian@BeaglePlay:~$ sudo shutdown -r now
```

Note: The default password is *temppwd*.

2. Clone cc1352-flasher

```
cd
git clone https://git.beagleboard.org/beagleconnect/cc1352-
↪flasher.git
```

3. Flash Firmware

```
python $HOME/cc1352-flasher --beagleplay $HOME/zephyr-beagle-
↪cc1352-sdk/build/zephyr/zephyr.bin
```

4. Ensure the *gb-beagleplay* driver is set to load.

```
sudo sed -e '/bcfserial-no-firmware/ s/^#*/#/' -i /boot/firmware/
↪extlinux/extlinux.conf
sudo shutdown -r now
```

Building gb-beagleplay Kernel Module

Note: *gb-beagleplay* is still not merged upstream and thus needs to be built seperately. This should not be required in the future.

1. Disable bcfserial driver. Add `quiet module_blacklist=bcfserial` to kernel parameters at `/boot/firmware/extlinux/extlinux.conf` (line 2) as shown below.

```
label Linux eMMC
    kernel /Image
    append root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait net.
↪ifnames=0 quiet module_blacklist=bcfserial ①
    fdtdir /
    #fdtoverlays /overlays/<file>.dtbo
    #fdtoverlays /overlays/k3-am625-beagleplay-bcfserial-no-firmware.
↪dtbo
    fdtoverlays /overlays/k3-am625-beagleplay-release-mikrobus.dtbo
    initrd /initrd.img
```

① `quiet module_blacklist=bcfserial` has been added to this line

1. Reboot

```
debian@BeaglePlay:~$ sudo shutdown -r now
```

2. Download the upstream module

```
debian@BeaglePlay:~$ git clone https://git.beagleboard.org/gsoc/
↪greybus/beagleplay-greybus-driver.git
debian@BeaglePlay:~$ cd beagleplay-greybus-driver
```

3. Install dependencies

```
debian@BeaglePlay:~$ sudo apt install linux-headers-$(uname -r)
```

4. Build Kernel moudle

```
debian@BeaglePlay:~/beagleplay-greybus-driver$ make
make -C /lib/modules/5.10.168-ti-arm64-r111/build M=/home/debian/
↪beagleplay-greybus-driver modules
make[1]: Entering directory '/usr/src/linux-headers-5.10.168-ti-
↪arm64-r111'
  CC [M]  /home/debian/beagleplay-greybus-driver/gb-beagleplay.o
  MODPOST /home/debian/beagleplay-greybus-driver/Module.symvers
  CC [M]  /home/debian/beagleplay-greybus-driver/gb-beagleplay.
↪mod.o
  LD [M]  /home/debian/beagleplay-greybus-driver/gb-beagleplay.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.10.168-ti-
↪arm64-r111'
```

Flashing BeagleConnect Freedom Greybus Firmware

1. Connect BeagleConnect Freedom to BeaglePlay
2. Build the BeagleConnect Freedom firmware

```
west build -b beagleconnect_freedom modules/greybus/samples/
↪subsys/greybus/net/ -p -- -DOVERLAY_CONFIG=overlay-802154-subg.
↪conf
```

3. Flash the BeagleConnect Freedom

```
west flash
```

Run the Demo

1. Connect BeagleConnect Freedom.
2. See shell output using *tio*

```
tio /dev/ACM0
```

3. Press the Reset button on BeagleConnect Freedom
4. Verify that greybus is working by checking the *tio* output. It should look as follows:

```
[00:00:00.000,976] <dbg> greybus_platform_bus: greybus_init:␣
↪probed greybus: 0 major: 0 minor: 1
[00:00:00.001,068] <dbg> greybus_platform_string: greybus_string_
↪init: probed greybus string 4: hdc2010
[00:00:00.001,129] <dbg> greybus_platform_string: greybus_string_
↪init: probed greybus string 3: opt3001
[00:00:00.001,190] <dbg> greybus_platform_string: greybus_string_
↪init: probed greybus string 2: Greybus Service Sample␣
↪Application
[00:00:00.001,251] <dbg> greybus_platform_string: greybus_string_
↪init: probed greybus string 1: Zephyr Project RTOS
[00:00:00.001,251] <dbg> greybus_platform_interface: greybus_
↪interface_init: probed greybus interface 0
[00:00:00.001,281] <dbg> greybus_platform_bundle: greybus_bundle_
↪init: probed greybus bundle 1: class: 10
[00:00:00.001,312] <dbg> greybus_platform_bundle: greybus_bundle_
↪init: probed greybus bundle 0: class: 0
```

(continues on next page)

(continued from previous page)

```
[00:00:00.001,342] <dbg> greybus_platform_control: greybus_
↪control_init: probed cport 0: bundle: 0 protocol: 0
[00:00:00.001,434] <dbg> greybus_platform: gb_add_cport_device_
↪mapping: added mapping between cport 1 and device gpio@40022000
[00:00:00.001,464] <dbg> greybus_platform_gpio_control: greybus_
↪gpio_control_init: probed cport 1: bundle: 1 protocol: 2
[00:00:00.001,556] <dbg> greybus_platform: gb_add_cport_device_
↪mapping: added mapping between cport 2 and device sensor-switch
[00:00:00.001,556] <dbg> greybus_platform_i2c_control: greybus_
↪i2c_control_init: probed cport 2: bundle: 1 protocol: 3
*** Booting Zephyr OS build bcf-sdk-0.2.1-3384-ge76584f824c8 ***
[00:00:00.009,704] <dbg> greybus_service: greybus_service_init:␣
↪Greybus initializing..
[00:00:00.009,765] <dbg> greybus_manifest: identify_descriptor:␣
↪cport_id = 0
[00:00:00.009,796] <dbg> greybus_manifest: identify_descriptor:␣
↪cport_id = 1
[00:00:00.009,826] <dbg> greybus_manifest: identify_descriptor:␣
↪cport_id = 2
[00:00:00.009,857] <dbg> greybus_transport_tcpip: gb_transport_
↪backend_init: Greybus TCP/IP Transport initializing..
[00:00:00.010,101] <inf> greybus_transport_tcpip: CPort 0 mapped␣
↪to TCP/IP port 4242
[00:00:00.014,709] <inf> greybus_transport_tcpip: CPort 1 mapped␣
↪to TCP/IP port 4243
[00:00:00.014,953] <inf> greybus_transport_tcpip: CPort 2 mapped␣
↪to TCP/IP port 4244
[00:00:00.015,075] <inf> greybus_transport_tcpip: Greybus TCP/IP␣
↪Transport initialized
[00:00:00.015,136] <inf> greybus_manifest: Registering CONTROL␣
↪greybus driver.
[00:00:00.015,167] <dbg> greybus: _gb_register_driver:␣
↪Registering Greybus driver on CP0
[00:00:00.015,411] <inf> greybus_manifest: Registering GPIO␣
↪greybus driver.
[00:00:00.015,411] <dbg> greybus: _gb_register_driver:␣
↪Registering Greybus driver on CP1
[00:00:00.015,625] <inf> greybus_manifest: Registering I2C␣
↪greybus driver.
[00:00:00.015,625] <dbg> greybus: _gb_register_driver:␣
↪Registering Greybus driver on CP2
[00:00:00.015,777] <inf> greybus_service: Greybus is active
```

5. Load gb-beagleplay

```
debian@BeaglePlay:~$ sudo insmod $HOME/beagleplay-greybus-driver/
↪gb-beagleplay.ko
```

6. Check *iio_device* to verify that greybus node has been detected:

```
debian@BeaglePlay:~$ iio_info
Library version: 0.24 (git tag: v0.24)
Compiled with backends: local xml ip usb
IIO context created with local backend.
Backend version: 0.24 (git tag: v0.24)
Backend description string: Linux BeaglePlay 5.10.168-ti-arm64-
↪r111 #1bullseye SMP Tue Sep 26 14:22:20 UTC 2023 aarch64
IIO context has 2 attributes:
        local,kernel: 5.10.168-ti-arm64-r111
        uri: local:
IIO context has 2 devices:
```

(continues on next page)

(continued from previous page)

```
        iio:device0: adc102s051
                2 channels found:
                        voltage1:  (input)
                        2 channel-specific attributes found:
                                attr  0: raw value: 4068
                                attr  1: scale value: 0.805664062
                        voltage0:  (input)
                        2 channel-specific attributes found:
                                attr  0: raw value: 0
                                attr  1: scale value: 0.805664062
                No trigger on this device
        iio:device1: hdc2010
                3 channels found:
                        temp:  (input)
                        4 channel-specific attributes found:
                                attr  0: offset value: -15887.
↪515151
                                attr  1: peak_raw value: 28928
                                attr  2: raw value: 28990
                                attr  3: scale value: 2.517700195
                        humidityrelative:  (input)
                        3 channel-specific attributes found:
                                attr  0: peak_raw value: 43264
                                attr  1: raw value: 41892
                                attr  2: scale value: 1.525878906
                        current:  (output)
                        2 channel-specific attributes found:
                                attr  0: heater_raw value: 0
                                attr  1: heater_raw_available␣
↪value: 0 1
                No trigger on this device
```

2.6 Support

2.6.1 Certifications and export control

Export designations

- HS: 8471504090
- US HS: 8473301180
- EU HS: 8471707000

Size and weight

- Bare board dimensions: 82.5 x 80 x 20 mm
- Bare board weight: 55.3 g
- Full package dimensions: 140 x 100 x 40 mm
- Full package weight: 125.3 g

2.6.2 Additional documentation

Hardware docs

For any hardware document like schematic diagram PDF, EDA files, issue tracker, and more you can checkout the BeaglePlay design repository.

Software docs

For BeaglePlay specific software projects you can checkout all the BeaglePlay project repositories group.

Support forum

For any additional support you can submit your queries on our forum, https://forum.beagleboard.org/tag/play

Pictures

2.6.3 Change History

Note: This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

Document Changes

For all changes, see https://git.beagleboard.org/docs/docs.beagleboard.io. Frozen releases tested against specific hardware and software revisions are noted below.

Table 2.6: BeaglePlay document change history

Rev	Changes	Date	By

Board Changes

For all changes, see https://git.beagleboard.org/beagleplay/beagleplay. Versions released into production are noted below.

Table 2.7: BeaglePlay board change history

Rev	Changes	Date	By
A2	Initial production version	2023-03-08	JK

Chapter 3

BeagleBone AI-64

BeagleBone® AI-64 brings a complete system for developing artificial intelligence (AI) and machine learning solutions with the convenience and expandability of the BeagleBone® platform and the peripherals on board to get started right away learning and building applications. With locally hosted, ready-to-use, open-source focused tool chains and development environment, a simple web browser, power source and network connection are all that need to be added to start building performance-optimized embedded applications. Industry-leading expansion possibilities are enabled through familiar BeagleBone® cape headers, with hundreds of open-source hardware examples and dozens of readily available embedded expansion options available off-the-shelf.



License Terms

- This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page
- For export, emissions and other compliance, see *Additional Support Information*



3.1 Introduction

BeagleBone AI-64 like its predecessors (BeagleBone AI), is designed to address the open-source Community, early adopters, and anyone interested in a low cost 64-bit Dual Arm® Cortex®-A72 processor based Single Board Computer (SBC). It also offers access to many of the interfaces and allows for the use of add-on boards called capes, to add many different combinations of features. A user may also develop their own board or add their own circuitry.

Note: AI-64 has been equipped with a minimum set of features to allow the user to experience the power of the processor and is not intended as a full development platform as many of the features and interfaces supplied by the processor are not accessible from BeagleBone AI-64 via onboard support of some interfaces. It is not a complete product designed to do any particular function. It is a foundation for experimentation and learning how to program the processor and to access the peripherals by the creation of your own software and hardware.

3.1.1 BeagleBone Compatibility

The board is intended to provide functionality well beyond BeagleBone Black or BeagleBone AI, while still providing compatibility with BeagleBone Black's expansion headers as much as possible. There are several significant differences between the three designs.

Table 3.1: Table: BeagleBone Compatibility

Feature	AI-64	AI	Black
SoC	TDA4VM	AM5729	AM3358
Arm CPU	Cortex-A72 (64-bit)	Cortex-A15 (32-bit)	Cortex-A8 (32-bit)
Arm cores/MHz	2x 2GHz	2x 1.5GHz	1x 1GHz
RAM	4GB	1GB	512MB
eMMC flash	16GB	16GB	4GB
Size	4" x 3.1"	3.4" x 2.1"	.4" x 2.1"
Display	miniDP + DSI	microHDMI	microHDMI
USB host (Type-A)	2x 5Gbps	1x 480Mbps	1x 480Mbps
USB dual-role	Type-C 5Gbps	Type-C 5Gbps	mini-AB 480Mbps
Ethernet	10/100/1000M	10/100/1000M	10/100M
M.2	E-key	-	-
WiFi/ Bluetooth	-	AzureWave AW‑CM256SM	-

Todo: add cape compatibility details

3.1.2 BeagleBone AI-64 Features and Specification

This section covers the specifications and features of the board and provides a high level description of the major components and interfaces that make up the board.

Table 3.2: Table: BeagleBone AI-64 Features and Specification

	Feature
Processor	Texas Instruments TDA4VM
Graphics Engine	PowerVR® Series8XE GE8430
SDRAM Memory	LPDDR4 3.2GHz (4GB) Kingston Q3222PM1WDGTK-U
Onboard Flash	eMMC (16GB) Kingston EMMC16G-TB29-PZ90
PMIC	TPS65941213 and TPS65941111 PMICs regulator and one additional LDO.
Debug Support	**2x 3 pin 3.3V TTL header** 1. WKUP_UART0: Wake-up domain serial port 2. UART0: Main domain serial port 10-pin JTAG TAG-CONNECT footprint
Power Source	USB C or DC Jack (5V @ >3A)
PCB	4” x 3.1”
Indicators	1x Power & 5x User Controllable LEDs
USB-3.0 Client Port	Access to USB0 SuperSpeed dual-role mode via USB-C (no power output)
USB-3.0 Host Port	TUSB8041 4-port SuperSpeed hub 1x on USB1, 2x Type A Socket up-to 2.8A total depending on power input
Ethernet	Gigabit RJ45 link indicator speed indicator
SD/MMC Connector	microSD (1.8/3.3V)
User Input	1. Reset Button 2. Boot Button 3. Power Button
Video Out	miniDP
Audio	via miniDP (stereo)
Weight	192gm (with heatsink)
Power	Refer to *Main Board Power* section

3.1.3 Board Component Locations

This section describes the key components on the board. It provides information on their location and function. Familiarize yourself with the various components on the board.

3.1.4 Board components

BeagleBone AI-64 board components below shows the locations of the connectors, LEDs, and switches on the PCB layout of the board.

- **DC Power** is the main DC input that accepts 5V power.
- **Power Button** alerts the processor to initiate the power down sequence and is used to power down the board.
- **GigaBit Ethernet** is the connection to the LAN.
- **Serial Debug ports** WKUP_UART0 for early boot from the management MCU and UART0 is for the main processor.
- **USB Client** is a USB-C connection to a PC that can also power the board.
- **BOOT switch** can be used to force a boot from the microSD card if the power is cycled on the board, removing power and reapplying the power to the board.
- There are five green **LEDs** that can be used by the user.

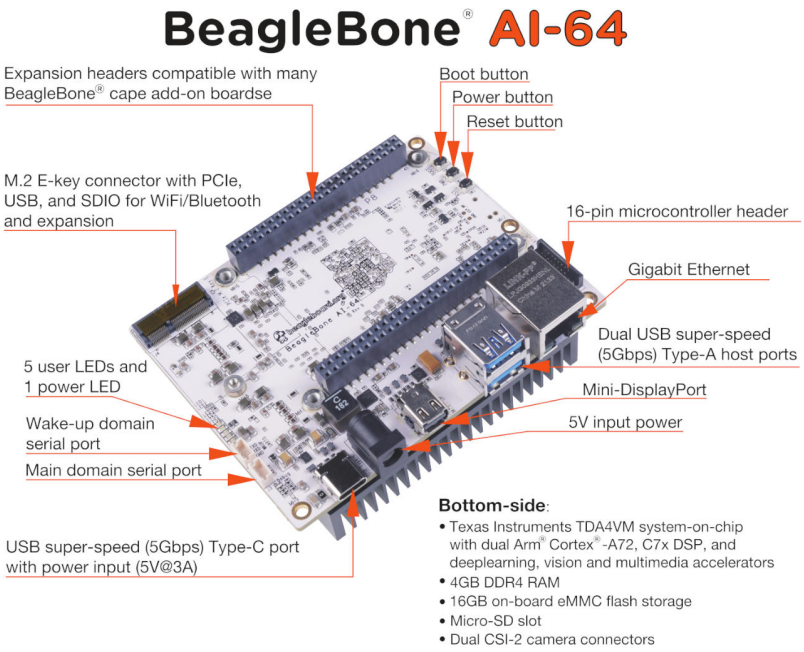


Fig. 3.1: BeagleBone AI-64 board components

- **Reset Button** allows the user to reset the processor.
- **microSD** slot is where a microSD card can be installed.
- **miniDP** connector is where the display is connected to.
- **USB Host** can be connected different USB interfaces such as Wi-Fi, Bluetooth, Keyboard, etc.

On bottom side we have,

- **TI TDA4VM** processor.
- **4GB LPDDR4** Dual Data Rate RAM memory.
- **Ethernet PHY** physical interface to the network.
- **eMMC** onboard MMC chip that holds up to 16GB of data.

3.2 Quick Start Guide

This section provides instructions on how to hook up your board. This beagle requires a 5V > 3A power supply to work properly via either USB Type-C power adapter or a barrel jack power adapter.

Recommended adapters:

- 5V @ 3A USB C power supply adapter for SBCs.
- 5V > 3A laptop/mobile adapter with USB-C cable.

All the *BeagleBone AI-64 connections ports* we will use in this chapter are shown in the figure below.

3.2.1 What's In the Box

In the box you will find two main items as shown in *BeagleBone AI-64 box content*.

- BeagleBone AI-64
- Instruction card

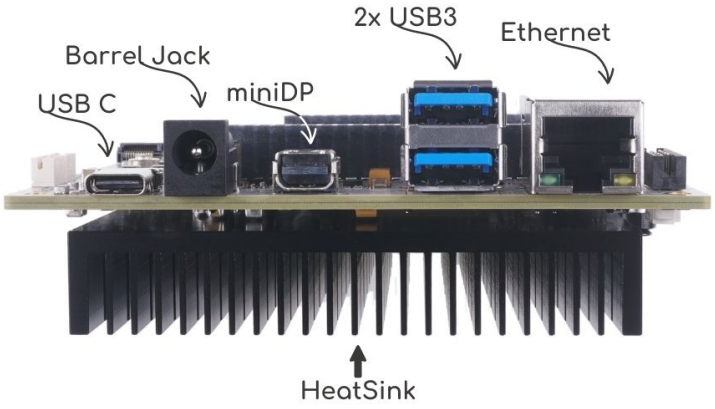


Fig. 3.2: BeagleBone AI-64 connections ports

A USB-C to USB-C cable is not included, but recommended for the tethered scenario and creates a developer experience where the board can be used immediately with no other equipment needed.



Fig. 3.3: BeagleBone AI-64 box content

3.2.2 Methods of operation

1. Tethered to a PC
2. Standalone development platform in a PC configuration using external peripherals

Main Connection Scenarios

This section describes how to connect and power the board and serves as a slightly more detailed description of the Quick Start Guide included in the box. The board can be configured in several different ways, but we will discuss the two most common scenarios.

- Tethered to a PC via the USB cable
  - Board is accessed as a storage drive and virtual Ethernet connection.
- Standalone Desktop
  - Display

- Keyboard and Mouse
- External 5V > 3A power supply

Each of these configurations is discussed in general terms in the following sections.

Tethered To A PC In this configuration, the board is powered by the PC via a single USB cable. The board is accessed either as a USB storage drive or via the browser on the connected PC. You need to use either Firefox or Chrome on the PC, Internet Explorer will not work properly.

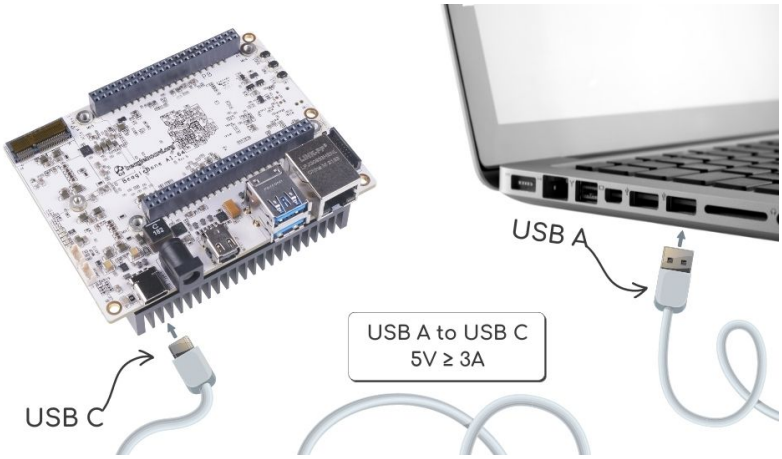


Fig. 3.4: Tethered Configuration

At least 5V @ 3A is required to power the board, In most cases the PC may not be able to supply sufficient power for the board unless the connection is made over a Type-C to Type-C cable. You should always use an external 5V > 3A DC power supply connected to the barrel jack if you are unsure that the system can provide the required power or are otherwise using a USB-A to Type-C cable which will always require power from the DC barrel jack.

Connect the Cable to the Board

1. Connect the type C USB cable to the board as shown in *USB Connection to the Board*. The connector is on the top side of the board near barrel jack.

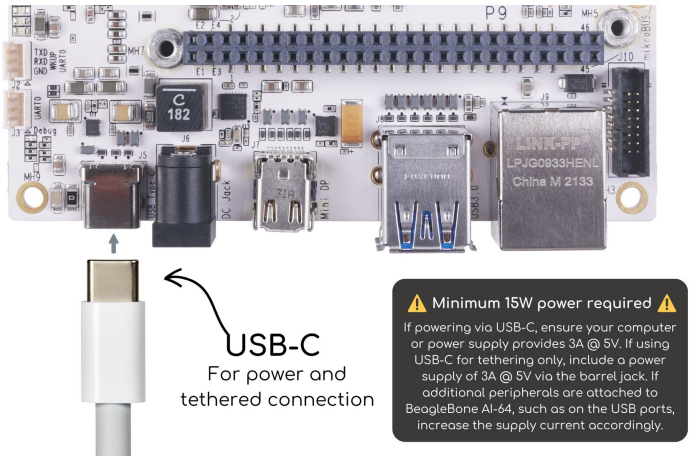


Fig. 3.5: USB Connection to the Board

2. Connect the USB-A end of the cable to your PC or laptop USB port as shown in the *USB Connection to the PC/Laptop* below.
3. The board will power on and the power LED will be on as shown in *Board Power LED* below.

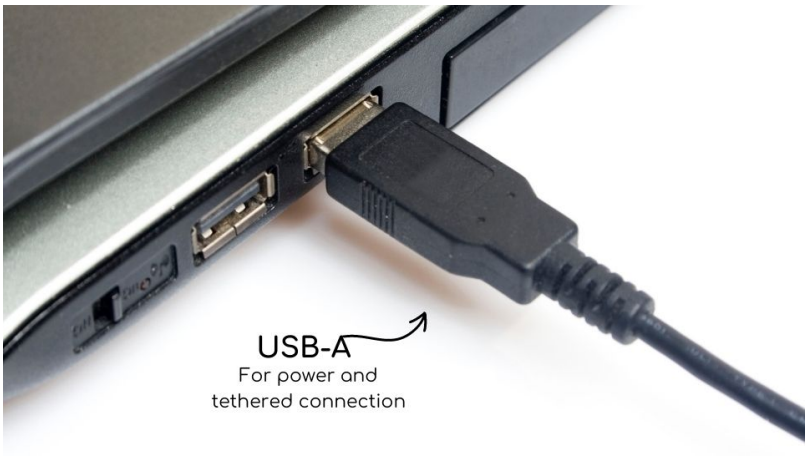


Fig. 3.6: USB Connection to the PC/Laptop

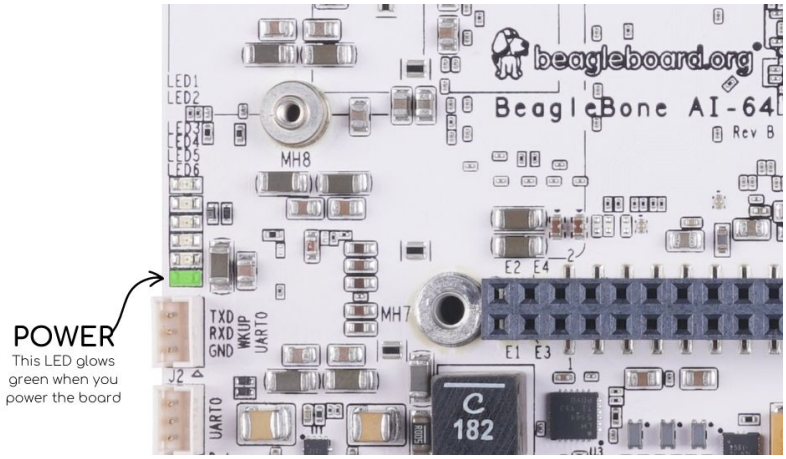


Fig. 3.7: Board Power LED

4. When the board starts to the booting process started by the process of applying power, the LEDs will come on in sequence as shown in *Board Boot Status* below. It will take a few seconds for the status LEDs to come on, so be patient. The LEDs will be flashing in an erratic manner as it begins to boot the Linux kernel.

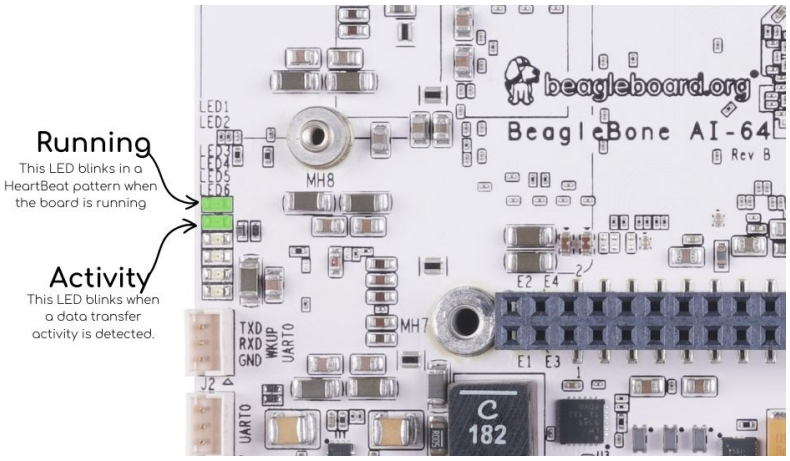


Fig. 3.8: Board Boot Status

Accessing the Board as a Storage Drive The board will appear around a USB Storage drive on your PC after thekernel has booted, which will take a round 10 seconds. The kernel on the board needs to boot before the port gets enumerated. Once the board appears as a storage drive, do the following:

1. Open the USB Drive folder.
2. Click on the file named **start.htm**
3. The file will be opened by your browser on the PC and you should get a display showing the Quick Start Guide.
4. Your board is now operational! Follow the instructions on your PC screen.

Standalone w/Display and Keyboard/Mouse In this configuration, the board works more like a PC, totally free from any connection to a PC as shown in *Desktop Configuration*. It allows you to create your code to make the board do whatever you need it to do. It will however require certain common PC accessories. These accessories and instructions are described in the following section.

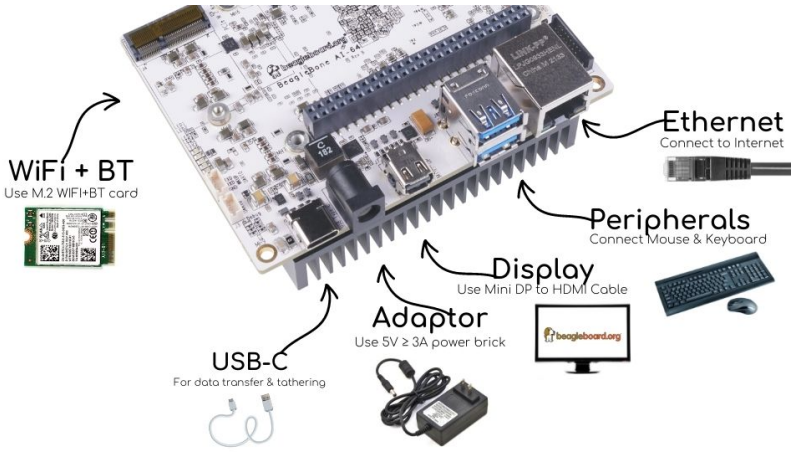


Fig. 3.9: Desktop Configuration

Ethernet cable and M.2 WiFi + Bluetooth card are optional. They can be used if network access required.

Required Accessories In order to use the board in this configuration, you will need the following accessories:

- 5V > 3A power supply.
- Display Port or HDMI monitor.
- miniDP-DP or active miniDP-HDMI cable (or a recommended **miniDP-DP or active miniDP-HDMI adapter** https://www.amazon.com/dp/B089GF8M87 has been tested and worked beautifully).
- USB wired/wireless keyboard and mouse.
- powered USB HUB (OPTIONAL). The board has only two USB Type-A host ports, so you may need to use a powered USB Hub if you wish to add additional USB devices, such as a USB WiFi adapter.
- M.2 Bluetooth & WiFi module (OPTIONAL). For wireless connections, a USB WiFi adapter or a recommended M.2 WiFi module can provide wireless networking.

Connecting Up the Board

1. Connect the miniDP to DP or active miniDP to HDMI cable from your BeagleBone AI-64 to your monitor.

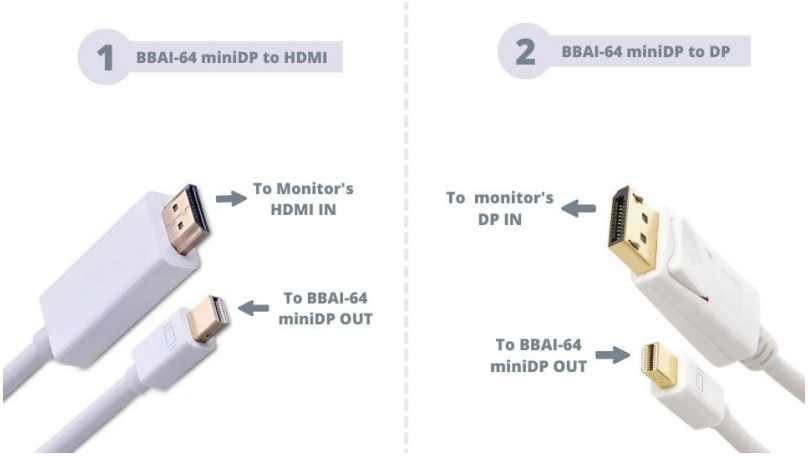


Fig. 3.10: Connect miniDP-DP or active miniDP-HDMI cable to BeagleBone AI-64

2. If you have an Display Port or HDMI monitor with HDMI-HDMI or DP-DP cable you can use adapters as shown in. *Display adapters*.

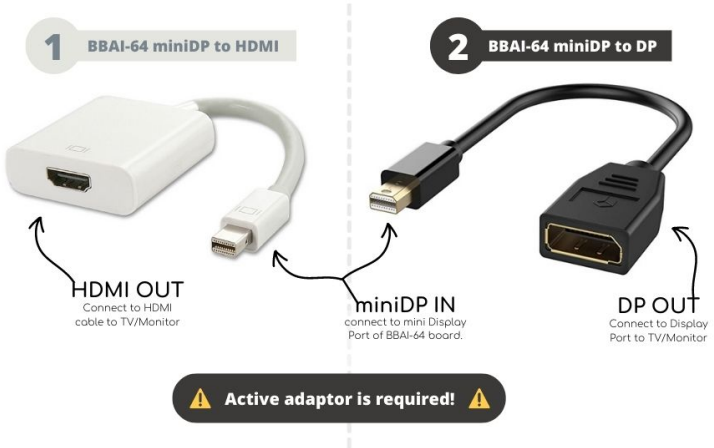


Fig. 3.11: Display adapters

3. If you have wired/wireless USB keyboard and mouse such as seen in *Keyboard and Mouse* below, you need to plug the receiver in the USB host port of the board as shown in *Keyboard and Mouse*.

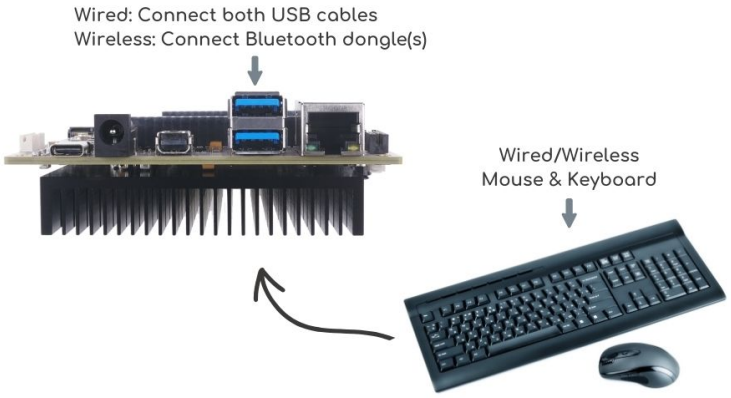


Fig. 3.12: Keyboard and Mouse

4. Connect the Ethernet Cable

If you decide you want to connect to your local area network, an Ethernet cable can be used. Connect the Ethernet Cable to the Ethernet port as shown in *Ethernet Cable Connection*. Any standard 100M Ethernet cable should work.

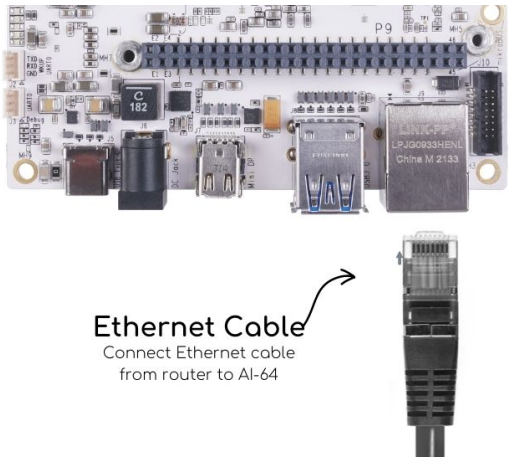


Fig. 3.13: Ethernet Cable Connection

5. The final step is to plug in the DC power supply to the DC power jack as shown in *External DC Power* below.
6. The cable needed to connect to your display is a miniDP-DP or active miniDP-HDMI. Connect the miniDP connector end to the board at this time. The connector is on the top side of the board as shown in *Connect miniDP to DP or active miniDP to HDMI Cable to the Board* below.

The connector is fairly robust, but we suggest that you not use the cable as a leash for your Beagle. Take proper care not to put too much stress on the connector or cable.

7. Booting the Board

As soon as the power is applied to the board, it will start the booting up process. When the board starts to boot the LEDs will come on. It will take a few seconds for the status LEDs to come on, so be patient. The LEDs will be flashing in an erratic manner as it boots the Linux kernel.

While the four user LEDS can be over written and used as desired, they do have specific meanings in the image that is shipped with the board once the Linux kernel has booted.

- **USR0** is the heartbeat indicator from the Linux kernel.

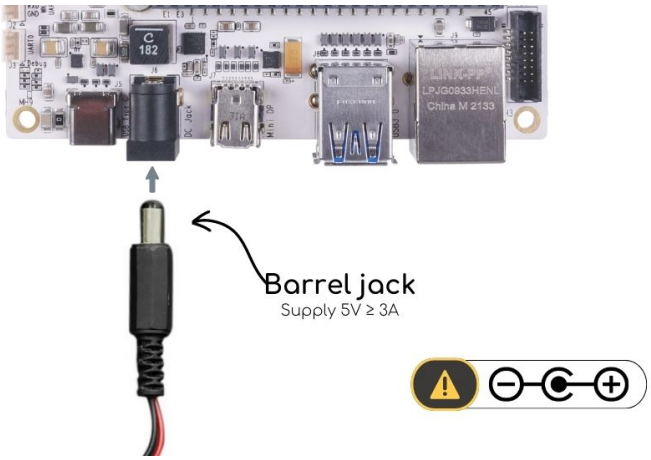


Fig. 3.14: External DC Power

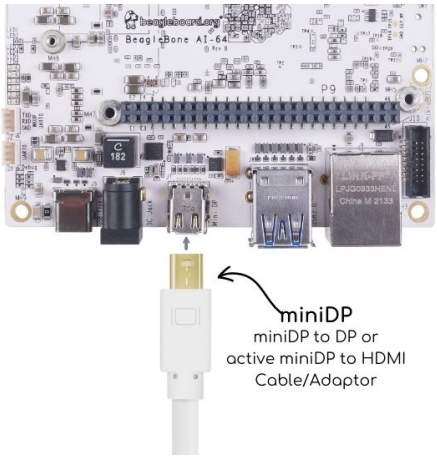


Fig. 3.15: Connect miniDP to DP or active miniDP to HDMI Cable to the Board

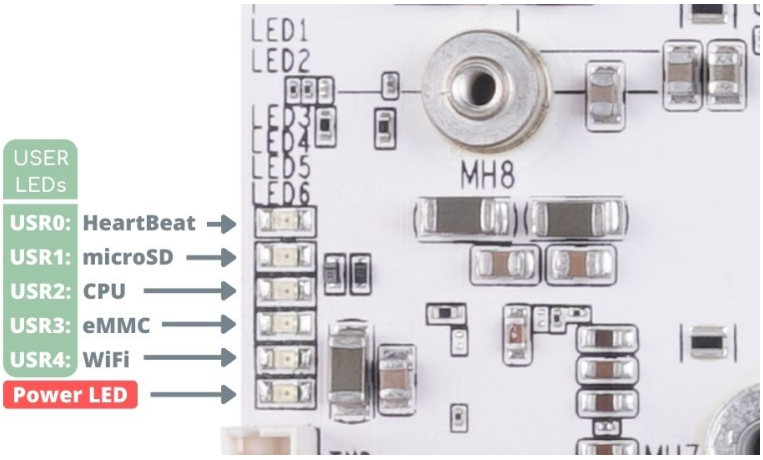


Fig. 3.16: BeagleBone AI-64 LEDs

- **USR1** turns on when the microSD card is being accessed
- **USR2** is an activity indicator. It turns on when the kernel is not in the idle loop.
- **USR3** turns on when the onboard eMMC is being accessed.
- **USR4** is an activity indicator for WiFi.

8. A Booted System

    a. The board will have a mouse pointer appear on the screen as it enters the Linux boot step. You may have to move the physical mouse to get the mouse pointer to appear. The system can come up in the suspend mode with the monitor in a sleep mode.

    b. After a minute or two a login screen will appear. You do not have to do anything at this point.

    c. After a minute or two the desktop will appear. It should be similar to the one shown in *BeagleBone XFCE Desktop Screen*. HOWEVER, it will change from one release to the next, so do not expect your system to look exactly like the one in the figure, but it will be very similar.

    d. And at this point you are ready to go! *BeagleBone XFCE Desktop Screen* shows the desktop after booting.

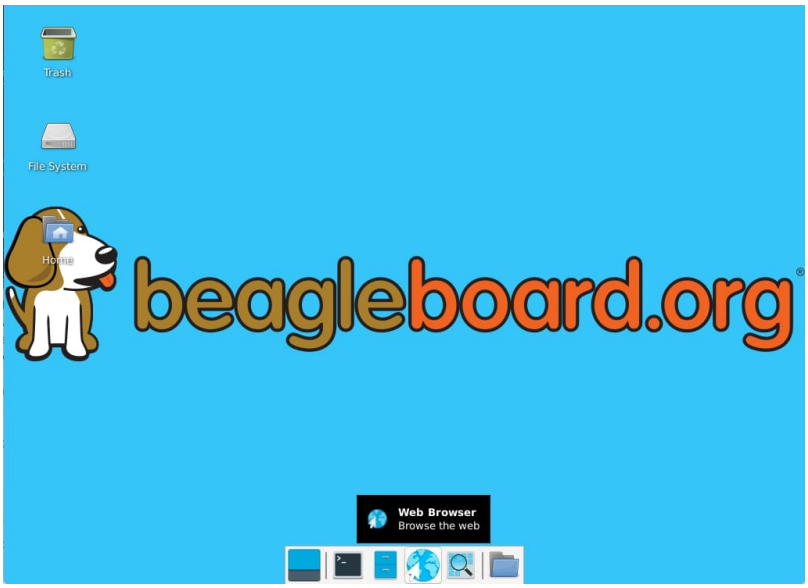


Fig. 3.17: BeagleBone XFCE Desktop Screen

3.2.3 Update software

Production boards currently ship with the factory-installed 2022-01-14-8GB image. To upgrade from the software image on your BeagleBone AI-64 to the latest, you don't need to completely reflash the board. If you do want to reflash it, visit the flashing instructions on the getting started page. Factory Image update (without reflashing)...

```
sudo apt update
sudo apt install --only-upgrade bb-j721e-evm-firmware generic-sys-mods
sudo apt upgrade
```

Update U-Boot:

to ensure only tiboot3.bin is in boot0, the pre-production image we tried to do more in boot0, but failed...

```
sudo /opt/u-boot/bb-u-boot-beagleboneai64/install-emmc.sh
sudo /opt/u-boot/bb-u-boot-beagleboneai64/install-microsd.sh
sudo reboot
```

Update Kernel and SGX modules:

```
sudo apt install bbb.io-kernel-5.10-ti-k3-j721e
```

Update xfce:

```
sudo apt install bbb.io-xfce4-desktop
```

Update ti-edge-ai 8.2 examples

```
sudo apt install ti-edgeai-8.2-base ti-vision-apps-8.2 ti-vision-apps-eaik-
↪firmware-8.2
```

Cleanup:

```
sudo apt autoremove --purge
```

3.2.4 Next steps

- *Edge AI*

3.3 Design and Specifications

If you want to know how BeagleBone AI-64 is designed and the detailed specifications, then this chapter is for you. We are going to attept to provide you a short and crisp overview followed by discussing each hardware design element in detail.

3.3.1 Block Diagram and Overview

BeagleBone AI-64 Key Components below shows the high level block diagram of BeagleBone AI-64 board surrounding TDA4VM SoC.

Processor

BeagleBone AI-64 uses TI J721E-family TDA4VM system-on-chip (SoC) which is part of the K3 Multicore SoC architecture platform and it is targeted for the reliability and low-latency needs of the automotive market provide for a great general purpose platform suitable for industrial automation, mobile robotics, building automation and numerous hobby projects.

The SoC designed as a low power, high performance and highly integrated device architecture, adding significant enhancement on processing power, graphics capability, video and imaging processing, virtualization and coherent memory support. In addition, these SoCs support state of the art security and functional safety features. For the remaining of this section device, SoC, and processor will be used interchangeably.

Some of the main distinguished characteristics of the device are:

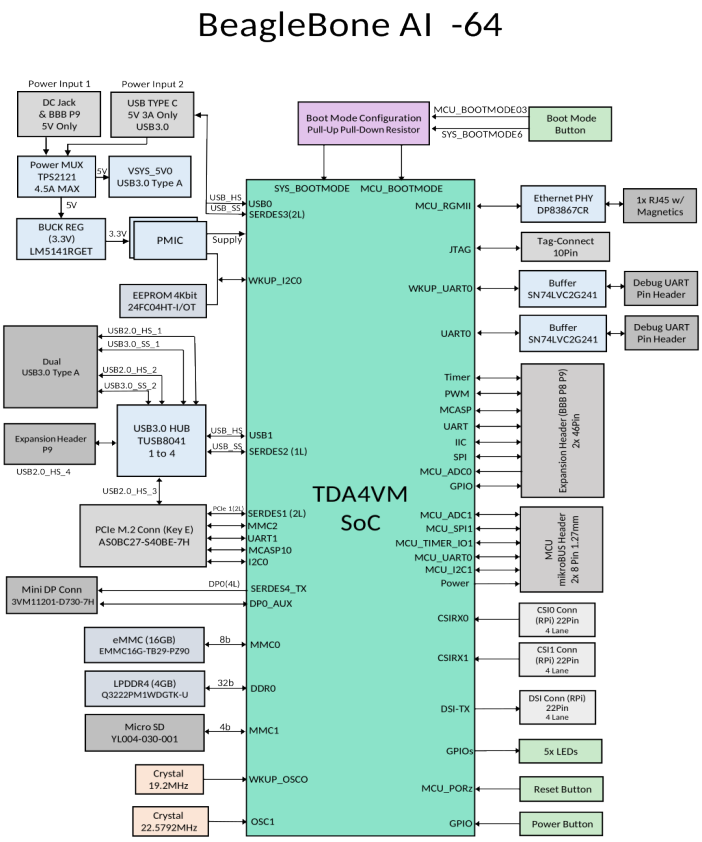


Fig. 3.18: BeagleBone AI-64 Key Components

- 64-bit architecture with virtualization and coherent memory support, which leverages full processing capability of 64-bit Arm® Cortex®-A72
- Fully programmable industrial communication subsystems to enable future-proof designs for customers that need to adopt the new Gigabit Time-sensitive Networks (TSN) standards, but still need full support on legacy protocols and continuous system optimization over the product deployment
- Integration of vision hardware processing accelerators to facilitate extensive processing requirements in low power budget for automotive ADAS and machine vision applications
- Integration of a general-purpose microcontroller unit (MCU) with a dual Arm® Cortex®-R5F MCU subsystem, available for general purpose use as two cores or in lockstep, intended to help customers achieve functional safety goals for their end products
- Integration of a next-generation fixed and floating-point C71x Digital Signal Processor (DSP) that significantly boosts power over a broad range of general signal processing tasks for both general applications and automotive functions which also incorporates advanced techniques to improve control code efficiency and ease of programming such as branch prediction, protected pipeline, precise exception and virtual memory management
- Tightly coupled Matrix Multiplication Accelerator (MMA) that extends the C71x DSP architecture's scalar and vector facilities enabling deep learning and enhance vision, analytics and wide range of general applications. The achieved total TOPS (Tera Operations Per Second) performance significantly differentiates the device for single board computer in machine vision and deep learning applications
- Key display features including flexibility to interface with different panel types (eDP, DSI, DPI) with multi-layer hardware composition
- Integration of hardware features that help applications to achieve functional safety mechanisms
- Robust security architecture with sandboxed DMSC controller managing all secure configurations with high performance client-server messaging scheme between secure DMSC and all cores
- Simplified solution for power supply management, enabling lower cost system solution (on-die bias LDOs and power good comparators for minimal power sequencing requirements consistent with low cost supply design)

The device is composed of the following main subsystems, across different domains of the SoC, among others:

- One dual-core 64-bit Arm Cortex-A72 microprocessor subsystem at up to 2.0 GHz and up to 24K DMIPS (Dhrystone Million Instructions per Second)
- Up to three Microcontroller Units (MCU), based on dual-core Arm Cortex-R5F processor running at up to 1.0 GHz, up to 12K DMIPS
- Up to two TMS320C66x DSP CorePac modules running at up to 1.35 GHz, up to 40 GFLOPS
- One C71x floating point, vector DSP running at up to 1.0 GHz, up to 80 GFLOPS
- One deep-learning MMA, up to 8 TOPS (8b) at 1.0 GHz
- Up to two gigabit dual-core Programmable Real-Time Unit and Industrial Communication Subsystems (PRU_ICSSG)
- Two Navigator Subsystems (NAVSS) for data movement and control
- One multi-pipeline Display Subsystem (DSS) with one MIPI® Display Serial Interface Controller (DSI) and shared MIPI D-PHY Transmitter (DPHY_TX), one Embedded DisplayPort Transmitter (EDP) with shared Serializer/Deserializer (SERDES), and two MIPI Display Pixel Interface (DPI) ports
- Two Camera Streaming Interface Receivers (CSI_RX_IF) with dedicated MIPI D-PHYs (DPHY_RX)
- One Camera Streaming Interface Transmitter (CSI_TX_IF) with MIPI D-PHY Transmitter (DPHY_TX) shared with DSI
- One Vision Processing Accelerator (VPAC) with image signal processor
- One Depth and Motion Processing Accelerator (DMPAC)

- One dual-core multi-standard HD Video Decoder (DECODER)
- One dual-core multi-standard HD Video Encoder (ENCODER)
- One Graphics Processing Unit (GPU)
- One Device Management and Security Controller (DMSC)

The device provides a rich set of peripherals such as:

- General connectivity peripherals, including:
  - Two 12-bit general purpose Analog-to-Digital Converters (ADC)
  - Ten Inter-Integrated Circuit (I2C) interfaces
  - Three Improved Inter-Integrated Circuit (I3C) controllers
  - Eleven master/slave Multichannel Serial Peripheral Interfaces (MCSPI)
  - Twelve configurable Universal Asynchronous Receiver/Transmitter (UART) interfaces
  - Ten General-Purpose Input/Output (GPIO) modules
- High-speed interfaces, including:
  - Two Gigabit Ethernet Switch (CPSW) modules
  - Two Dual-Role-Device (DRD) Universal Serial Bus Subsystems (USBSS) with integrated PHY
  - Four Peripheral Component Interconnect express (PCIe) Gen3 subsystems
- Flash memory interfaces, including:
  - One Octal SPI (OSPI) interface and one Quad SPI (QSPI) or one QSPI and one HyperBus^TM^
  - One General Purpose Memory Controller (GPMC) with Error Location Module (ELM) and 8- or 16-bit-wide data bus width (supports parallel NOR or NAND FLASH devices)
  - Three Multimedia Card/Secure Digital (MMCSD) controllers
  - One Universal Flash Storage (UFS) interface
- Industrial and control interfaces, including:
  - Sixteen Controller Area Network (MCAN) interfaces with flexible data rate support
  - Three Enhanced Capture (ECAP) modules
  - Six Enhanced Pulse-Width Modulation (EPWM) subsystems
  - Three Enhanced Quadrature Encoder Pulse (EQEP) modules
- Audio peripherals, including:
  - One Audio Tracking Logic (ATL)
  - Twelve Multichannel Audio Serial Port (MCASP) modules supporting up to 16 channels with independent TX/RX clock/sync domain
- One Video Processing Front End (VPFE) interface module

The device also integrates:

- Power distribution, reset controls and clock management components
- Power-management techniques for device power consumption minimization:

  - Adaptive Voltage Scaling (AVS)
  - Dynamic Frequency Scaling (DFS)
  - Gated clocks
  - Multiple voltage domains
  - Independently controlled power domains for major modules
  - Voltage and Temperature Management (VTM) module
  - Power-on Reset Generators (PRG)
  - Power Sleep Controllers (PSC)
- Optimized interconnect (CBASS) architecture to enable latency-critical real time network and IO applications
- Control modules (CTRL_MMRs) mainly associated with device top-level configurations such as:
  - IO Pad and pin multiplexing configuration
  - PLL control and associated High-Speed Dividers (HSDIV)
  - Clock selection
  - Analog function controls
- Multicore Shared Memory Controller (MSMC)
- DDR Subsystem (DDRSS) with Error Correcting Code (ECC), supporting LPDDR4
- 1KB RAM with ECC support for C71x boot vectors
- 2KB RAM with ECC support for A72 and R5F boot vectors
- 512KB On-Chip SRAM protected by ECC
- One Global Time Counter (GTC) module
- Thirty 32-bit counter timers with compare and capture modes
- Debug and trace capabilities

The device includes different modules for functional safety requirements support:

- MCU island with dual lock step Arm Cortex-R5F
- Safety enabled interconnect with implemented features to help with Freedom From Interference (FFI)
- Twelve Real Time Interrupt (RTI) modules with Windowed Watchdog Timer (WWDT) functionality to monitor processor cores
- Sixteen Dual-Clock Comparators (DCC) to monitor clocking sources during run-time
- Three Error Signaling Modules (ESM) to enable error monitoring
- Temperature monitoring sensors
- ECC on all critical memories
- Dedicated hardware Memory Cyclic Redundancy Check (MCRC) blocks

The device supports the following main security functionalities among others:

- Secure Boot Management
- Public Key Accelerator (PKA) for large vector math operation
- Cryptographic acceleration (AES, 3DES, MD5, SHA1, SHA2-224, 256, 512 operation)
- Trusted Execution Environment (TEE)
- Secure storage support
- On-the-fly encryption and authentication support for OSPI interface

The device is partitioned into three functional domains as shown in *Device Top-level Block Diagram*, each containing specific processing cores and peripherals:

- Wake-up (WKUP) domain
- Microcontroller (MCU) domain with one of the dual Cortex-R5 cluster
- MAIN domain

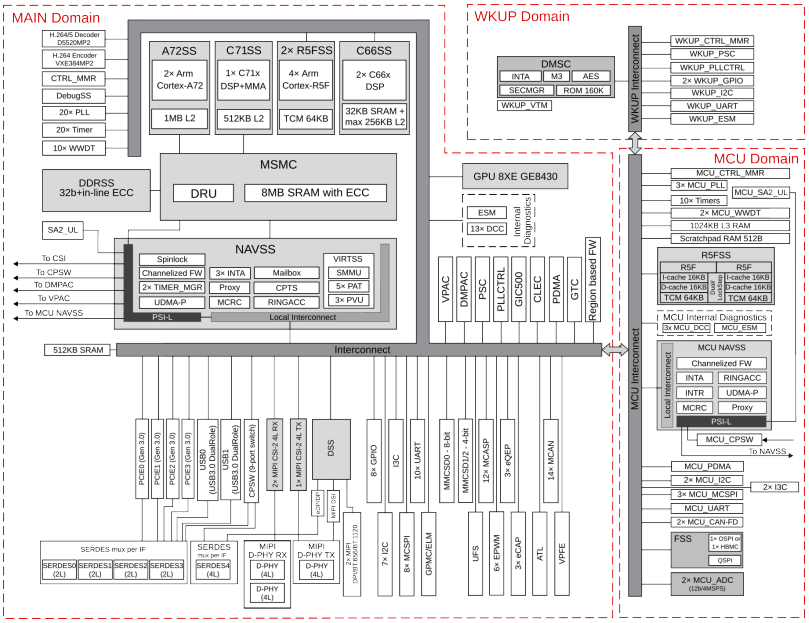


Fig. 3.19: Device Top-level Block Diagram

Memory

Described in the following sections are the three memory devices found on the board.

4GB LPDDR4 A single (1024M x 16bits x 2channels) LPDDR4 4Gb memory device is used. The memory used is:

- Kingston Q3222PM1WDGTK-U

4Kb EEPROM A single 4Kb EEPROM (24FC04HT-I/OT) is provided on I2C0 that holds the board information. This information includes board name, serial number, and revision information.

16GB Embedded MMC A single 16GB embedded MMC (eMMC) device is on the board. The device connects to the MMC1 port of the processor, allowing for 8bit wide access. Default boot mode for the board will be MMC1 with an option to change it to MMC0, the SD card slot, for booting from the SD card as a result of removing and reapplying the power to the board. Simply pressing the reset button will not change the boot mode. MMC0 cannot be used in 8Bit mode because the lower data pins are located on the pins used by the Ethernet port. This does not interfere with SD card operation but it does make it unsuitable for use as an eMMC port if the 8 bit feature is needed.

MicroSD Connector The board is equipped with a single microSD connector to act as the secondary boot source for the board and, if selected as such, can be the primary boot source. The connector will support larger capacity microSD cards. The microSD card is not provided with the board. Booting from MMC0 will be used to flash the eMMC in the production environment or can be used by the user to update the SW as needed.

Boot Modes

As mentioned earlier, there are two boot modes:

- **eMMC Boot:** This is the default boot mode and will allow for the fastest boot time and will enable the board to boot out of the box using the pre-flashed OS image without having to purchase an microSD card or an microSD card writer.
- **SD Boot:** This mode will boot from the microSD slot. This mode can be used to override what is on the eMMC device and can be used to program the eMMC when used in the manufacturing process or for field updates.

Todo: This section needs more work and references to greater detail. Other boot modes are possible. Software to support USB and serial boot modes is not provided by beagleboard.org._Please contact TI for support of this feature.

A switch is provided to allow switching between the modes.

- Holding the boot switch down during a removal and reapplication of power without a microSD card inserted will force the boot source to be the USB port and if nothing is detected on the USB client port, it will go to the serial port for download.
- Without holding the switch, the board will boot try to boot from the eMMC. If it is empty, then it will try booting from the microSD slot, followed by the serial port, and then the USB port.
- If you hold the boot switch down during the removal and reapplication of power to the board, and you have a microSD card inserted with a bootable image, the board will boot from the microSD card.

Note: Pressing the RESET button on the board will NOT result in a change of the boot mode. You MUST remove power and reapply power to change the boot mode. The boot pins are sampled during power on reset from the PMIC to the processor.The reset button on the board is a warm reset only and will not force a boot mode change.

Power Management

The *TPS65941213 and TPS65941111* power management device is used along with a separate LDO to provide power to the system.

PC USB Interface

The board has a USB type-C connector that connects to USB0 port of the processor.

Serial Debug Ports

Two serial debug ports are provided on board via 3pin micro headers,

1. WKUP_UART0: Wake-up domain serial port
2. UART0: Main domain serial port

In order to use the interfaces a 3pin micro to 6pin dupont adaptor header is required with a 6 pin USB to TTL adapter. The header is compatible with the one provided by FTDI and can be purchased for about $$12 to $$20 from various sources. Signals supported are TX and RX. None of the handshake signals are supported.

USB Host Ports

On the board is a stacked dual USB 3.0 Type A female connector with full LS/FS/HS/SS host support. The ports can provide power on/off control and up to 1.5A of current at 5V. Under USB power, the board will not be able to supply the full 1.5A.

Power Sources

The board can be powered from three different sources:

- 5V > 3A power supply plugged into the barrel jack
- 5V > 3A capable device plugged into the USB Type-C connector
- The cape header pins

The power supply is not provided with the board but can be easily obtained from numerous sources. A 5V > 3A supply is mandatory to have with the board, but if there is a cape plugged into the board or you have a power hungry device or hub plugged into the host port, then more current may needed from the DC supply.

Reset Button

When pressed and released, causes a reset of the board.

Power Button

This button takes advantage of the input to the PMIC for power down features.

Indicators

There are a total of six green LEDs on the board.

- One green power LED indicates that power is applied and the power management IC is up.
- Five blue LEDs that can be controlled via the SW by setting GPIO pins.

3.4 Expansion

3.4.1 Cape Header Connectors

The cape expansion interface on the board is comprised of two headers P8 (46 pin) & P9 (50 pin). All signals on the expansion headers are **3.3V** unless otherwise indicated.

Important: Do not connect 5V logic level signals to these pins or the board will be damaged.

Important: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

Important: **NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.**

Connector P8

The following tables show the pinout of the **P8** expansion header. The SW is responsible for setting the default function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **GPIO** row is the expected gpio identifier number in the Linux kernel.

Each row includes the gpiochipX and pinY in the format of *X Y*. You can use these values to directly control the GPIO pins with the commands shown below.

```
# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset X Y=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset X Y=0

For Example:

+---------+----------+
| Pin     | P8.03    |
+=========+==========+
| GPIO    | 1 20     |
+---------+----------+

Use the commands below for controlling this pin (P8.03) where X = 1 and Y =␣
↪20

# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset 1 20=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset 1 20=0
```

The **BALL** row is the pin number on the processor.

The **REG** row is the offset of the control register for the processor pin.

The **MODE #** rows are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

Important: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

P8.01-P8.02

P8.01	P8.02
GND	GND

P8.03-P8.05

Pin	P8.03	P8.04	P8.05
GPIO	1 20	1 48	1 33
BALL	AH21	AC29	AH25
REG	0x00011C054	0x00011C0C4	0x00011C088
Page	46	30	50
MODE 0	PRG1_PRU0_GPO19	PRG0_PRU0_GPO5	PRG1_PRU1_GPO12
1	PRG1_PRU0_GPI19	PRG0_PRU0_GPI5	PRG1_PRU1_GPI12
2	PRG1_IEP0_EDC_SYNC_OUT0	~	PRG1_RGMII2_TD1
3	PRG1_PWM0_TZ_OUT	PRG0_PWM3_B2	PRG1_PWM1_A0
4	~	~	RGMII2_TD1
5	RMII5_TXD0	RMII3_TXD0	~
6	MCAN6_TX	~	MCAN7_TX
7	GPIO0_20	GPIO0_48	GPIO0_33
8	~	GPMC0_AD0	RGMII8_TD1
9	~	~	~
10	VOUT0_EXTPCLKIN	~	VOUT0_DATA12
11	VPFE0_PCLK	~	~
12	MCASP4_AFSX	MCASP0_AXR3	MCASP9_AFSX
13	~	~	~
14	~	~	~
Bootstrap	~	BOOTMODE2	~

P8.06-P8.09

Pin	P8.06	P8.07	P8.08	P8.09
GPIO	1 34	1 15	1 14	1 17
BALL	AG25	AD24	AG24	AE24
REG	0x00011C08C	0x00011C03C	0x00011C038	0x00011C044
Page	51	44	44	45
MODE 0	PRG1_PRU1_GPO13	PRG1_PRU0_GPO14	PRG1_PRU0_GPO13	PRG1_PRU0_GPO16
1	PRG1_PRU1_GPI13	PRG1_PRU0_GPI14	PRG1_PRU0_GPI13	PRG1_PRU0_GPI16
2	PRG1_RGMII2_TD2	PRG1_RGMII1_TD3	PRG1_RGMII1_TD2	PRG1_RGMII1_TXC
3	PRG1_PWM1_B0	PRG1_PWM0_A1	PRG1_PWM0_B0	PRG1_PWM0_A2
4	RGMII2_TD2	RGMII1_TD3	RGMII1_TD2	RGMII1_TXC
5	~	~	~	~
6	MCAN7_RX	MCAN5_RX	MCAN5_TX	MCAN6_RX
7	GPIO0_34	GPIO0_15	GPIO0_14	GPIO0_17
8	RGMII8_TD2	~	~	~
9	~	RGMII7_TD3	RGMII7_TD2	RGMII7_TXC
10	VOUT0_DATA13	VOUT0_DATA19	VOUT0_DATA18	VOUT0_DATA21
11	VPFE0_DATA8	VPFE0_DATA3	VPFE0_DATA2	VPFE0_DATA5
12	MCASP9_AXR0	MCASP7_AXR1	MCASP7_AXR0	MCASP7_AXR3
13	MCASP4_ACLKR	~	~	MCASP7_AFSR
14	~	~	~	~
Bootstrap	~	~	~	~

P8.10-P8.13

Pin	P8.10	P8.11	P8.12	P8.13
GPIO	1 16	1 60	1 59	1 89
BALL	AC24	AB24	AH28	V27
REG	0x00011C040	0x00011C0F4	0x00011C0F0	0x00011C168
Page	44	33	33	56
MODE 0	PRG1_PRU0_GPO15	PRG0_PRU0_GPO17	PRG0_PRU0_GPO16	RGMII5_TD1
1	PRG1_PRU0_GPI15	PRG0_PRU0_GPI17	PRG0_PRU0_GPI16	RMII7_TXD1
2	PRG1_RGMII1_TX_CTL	PRG0_IEP0_EDC_SYNC_OUT1	PRG0_RGMII1_TXC	I2C3_SCL
3	PRG1_PWM0_B1	PRG0_PWM0_B2	PRG0_PWM0_A2	~
4	RGMII1_TX_CTL	PRG0_ECAP0_SYNC_OUT	RGMII3_TXC	VOUT1_DATA4
5	~	~	~	TRC_DATA2
6	MCAN6_TX	~	~	EHRPWM0_B
7	GPIO0_16	GPIO0_60	GPIO0_59	GPIO0_89
8	~	GPMC0_AD5	~	GPMC0_A5
9	RGMII7_TX_CTL	OBSCLK1	~	~
10	VOUT0_DATA20	~	DSS_FSYNC1	~
11	VPFE0_DATA4	~	~	~
12	MCASP7_AXR2	MCASP0_AXR13	MCASP0_AXR12	MCASP11_ACLKX
13	MCASP7_ACLKR	~	~	~
14	~	~	~	~
Bootstrap	~	BOOTMODE7	~	~

P8.14-P8.16

Pin	P8.14	P8.15	P8.16
GPIO	1 75	1 61	1 62
BALL	AF27	AB29	AB28
REG	0x00011C130	0x00011C0F8	0x00011C0FC
Page	37	33	34
MODE 0	PRG0_PRU1_GPO12	PRG0_PRU0_GPO18	PRG0_PRU0_GPO19
1	PRG0_PRU1_GPI12	PRG0_PRU0_GPI18	PRG0_PRU0_GPI19
2	PRG0_RGMII2_TD1	PRG0_IEP0_EDC_LATCH_IN0	PRG0_IEP0_EDC_SYNC_OUT0
3	PRG0_PWM1_A0	PRG0_PWM0_TZ_IN	PRG0_PWM0_TZ_OUT
4	RGMII4_TD1	PRG0_ECAP0_IN_APWM_OUT	~
5	~	~	~
6	~	~	~
7	GPIO0_75	GPIO0_61	GPIO0_62
8	~	GPMC0_AD6	GPMC0_AD7
9	~	~	~
10	~	~	~
11	~	~	~
12	MCASP1_AXR8	MCASP0_AXR14	MCASP0_AXR15
13	~	~	~
14	UART8_CTSn	~	~
Bootstrap	~	~	~

P8.17-P8.19

Pin	P8.17	P8.18	P8.19
GPIO	1 3	1 4	1 88
BALL	AF22	AJ23	V29
REG	0x00011C00C	0x00011C010	0x00011C164
Page	40	40	57
MODE 0	PRG1_PRU0_GPO2	PRG1_PRU0_GPO3	RGMII5_TD2
1	PRG1_PRU0_GPI2	PRG1_PRU0_GPI3	UART3_TXD
2	PRG1_RGMII1_RD2	PRG1_RGMII1_RD3	~
3	PRG1_PWM2_A0	PRG1_PWM3_A2	SYNC3_OUT
4	RGMII1_RD2	RGMII1_RD3	VOUT1_DATA3
5	RMII1_CRS_DV	RMII1_RX_ER	TRC_DATA1
6	~	~	EHRPWM0_A
7	GPIO0_3	GPIO0_4	GPIO0_88
8	GPMC0_WAIT1	GPMC0_DIR	GPMC0_A4
9	RGMII7_RD2	RGMII7_RD3	~
10	~	~	~
11	~	~	~
12	MCASP6_AXR0	MCASP6_AXR1	MCASP10_AXR1
13	~	~	~
14	UART1_RXD	UART1_TXD	~
Bootstrap	~	~	~

P8.20-P8.22

Pin	P8.20	P8.21	P8.22
GPIO	1 76	1 30	1 5
BALL	AF26	AF21	AH23
REG	0x00011C134	0x00011C07C	0x00011C014
Page	37	49	41
MODE 0	PRG0_PRU1_GPO13	PRG1_PRU1_GPO9	PRG1_PRU0_GPO4
1	PRG0_PRU1_GPI13	PRG1_PRU1_GPI9	PRG1_PRU0_GPI4
2	PRG0_RGMII2_TD2	PRG1_UART0_RXD	PRG1_RGMII1_RX_CTL
3	PRG0_PWM1_B0	~	PRG1_PWM2_B0
4	RGMII4_TD2	SPI6_CS3	RGMII1_RX_CTL
5	~	RMII6_RXD1	RMII1_TXD0
6	~	MCAN8_TX	~
7	GPIO0_76	GPIO0_30	GPIO0_5
8	~	GPMC0_CSn0	GPMC0_CSn2
9	~	PRG1_IEP0_EDIO_DATA_IN_OUT30	RGMII7_RX_CTL
10	~	VOUT0_DATA9	~
11	~	~	~
12	MCASP1_AXR9	MCASP4_AXR3	MCASP6_AXR2
13	~	~	MCASP6_ACLKR
14	UART8_RTSn	~	UART2_RXD
Bootstrap	~	~	~

P8.23-P8.26

Pin	P8.23	P8.24	P8.25	P8.26
GPIO	1 31	1 6	1 35	1 51
BALL	AB23	AD20	AH26	AC27
REG	0x00011C080	0x00011C018	0x00011C090	0x00011C0D0
Page	50	41	51	31
MODE 0	PRG1_PRU1_GPO10	PRG1_PRU0_GPO5	PRG1_PRU1_GPO14	PRG0_PRU0_GPO8
1	PRG1_PRU1_GPI10	PRG1_PRU0_GPI5	PRG1_PRU1_GPI14	PRG0_PRU0_GPI8
2	PRG1_UART0_TXD	~	PRG1_RGMII2_TD3	~
3	PRG1_PWM2_TZ_IN	PRG1_PWM3_B2	PRG1_PWM1_A1	PRG0_PWM2_A1
4	~	~	RGMII2_TD3	~
5	RMII6_CRS_DV	RMII1_TX_EN	~	~
6	MCAN8_RX	~	MCAN8_TX	MCAN9_RX
7	GPIO0_31	GPIO0_6	GPIO0_35	GPIO0_51
8	GPMC0_CLKOUT	GPMC0_WEn	RGMII8_TD3	GPMC0_AD2
9	PRG1_IEP0_EDIO_DATA_IN_OUT31	~	~	~
10	VOUT0_DATA10	~	VOUT0_DATA14	~
11	GPMC0_FCLK_MUX	~	~	~
12	MCASP5_ACLKX	MCASP3_AXR0	MCASP9_AXR1	MCASP0_AXR6
13	~	~	MCASP4_AFSR	~
14	~	~	~	UART6_RXD
Bootstrap	~	BOOTMODE0	~	~

P8.27-P8.29

Pin	P8.27	P8.28	P8.29
GPIO	1 71	1 72	1 73
BALL	AA28	Y24	AA25
REG	0x00011C120	0x00011C124	0x00011C128
Page	36	36	36
MODE 0	PRG0_PRU1_GPO8	PRG0_PRU1_GPO9	PRG0_PRU1_GPO10
1	PRG0_PRU1_GPI8	PRG0_PRU1_GPI9	PRG0_PRU1_GPI10
2	~	PRG0_UART0_RXD	PRG0_UART0_TXD
3	PRG0_PWM2_TZ_OUT	~	PRG0_PWM2_TZ_IN
4	~	SPI3_CS3	~
5	~	~	~
6	MCAN11_RX	PRG0_IEP0_EDIO_DATA_IN_OUT30	PRG0_IEP0_EDIO_DATA_IN_OUT31
7	GPIO0_71	GPIO0_72	GPIO0_73
8	GPMC0_AD10	GPMC0_AD11	GPMC0_AD12
9	~	~	CLKOUT
10	~	DSS_FSYNC3	~
11	~	~	~
12	MCASP1_AFSX	MCASP1_AXR5	MCASP1_AXR6
13	~	~	~
14	~	UART8_RXD	UART8_TXD
Bootstrap	~	~	~

P8.30-P8.32

Pin	P8.30	P8.31	~	P8.32	~
GPIO	1 74	1 32	1 63	1 26	1 64
BALL	AG26	AJ25	AE29	AG21	AD28
REG	0x00011C12C	0x00011C084	0x00011C100	0x00011C06C	0x00011C104
Page	37	50	34	48	34
MODE 0	PRG0_PRU1_GPO11	PRG1_PRU1_GPO11	PRG0_PRU1_GPO0	PRG1_PRU1_GPO5	PRG0_PRU1_GPO1
1	PRG0_PRU1_GPI11	PRG1_PRU1_GPI11	PRG0_PRU1_GPI0	PRG1_PRU1_GPI5	PRG0_PRU1_GPI1
2	PRG0_RGMII2_TD0	PRG1_RGMII2_TD0	PRG0_RGMII2_RD0	~	PRG0_RGMII2_RD1
3	~	~	~	~	~
4	RGMII4_TD0	RGMII2_TD0	RGMII4_RD0	~	RGMII4_RD1
5	RMII4_TX_EN	RMII2_TX_EN	RMII4_RXD0	RMII5_TX_EN	RMII4_RXD1
6	~	~	~	MCAN6_RX	~
7	GPIO0_74	GPIO0_32	GPIO0_63	GPIO0_26	GPIO0_64
8	GPMC0_A26	RGMII8_TD0	UART4_CTSn	GPMC0_WPn	UART4_RTSn
9	~	EQEP1_I	~	EQEP1_S	~
10	~	VOUT0_DATA11	~	VOUT0_DATA5	~
11	~	~	~	~	~
12	MCASP1_AXR7	MCASP9_ACLKX	MCASP1_AXR0	MCASP4_AXR0	MCASP1_AXR1
13	~	~	~	~	~
14	~	~	UART5_RXD	TIMER_IO4	UART5_TXD
Bootstrap	~	~	~	~	~

P8.33-P8.35

Pin	P8.33	~	P8.34	P8.35	~
GPIO	1 25	1 111	1 7	1 24	1 116
BALL	AH24	AA2	AD22	AD23	Y3
REG	0x00011C068	0x00011C1C0	0x00011C01C	0x00011C064	0x00011C1D4
Page	48	67	41	47	67
MODE 0	PRG1_PRU1_GPO4	SPI0_CS0	PRG1_PRU0_GPO6	PRG1_PRU1_GPO3	SPI1_CS0
1	PRG1_PRU1_GPI4	UART0_RTSn	PRG1_PRU0_GPI6	PRG1_PRU1_GPI3	UART0_CTSn
2	PRG1_RGMII2_RX_CTL	~	PRG1_RGMII1_RXC	PRG1_RGMII2_RD3	~
3	PRG1_PWM2_B2	~	PRG1_PWM3_A1	~	UART5_RXD
4	RGMII2_RX_CTL	~	RGMII1_RXC	RGMII2_RD3	~
5	RMII2_TXD0	~	RMII1_TXD1	RMII2_RX_ER	~
6	~	~	AUDIO_EXT_REFCLK0	~	PRG0_IEP0_EDIO_OUTVALID
7	GPIO0_25	GPIO0_111	GPIO0_7	GPIO0_24	GPIO0_116
8	RGMII8_RX_CTL	~	GPMC0_CSn3	RGMII8_RD3	PRG0_IEP0_EDC_LATCH_IN0
9	EQEP1_B	~	RGMII7_RXC	EQEP1_A	~
10	VOUT0_DATA4	~	~	VOUT0_DATA3	~
11	VPFE0_DATA13	~	~	VPFE0_WEN	~
12	MCASP8_AXR2	~	MCASP6_AXR3	MCASP8_AXR1	~
13	MCASP8_ACLKR	~	MCASP6_AFSR	MCASP3_AFSR	~
14	TIMER_IO3	~	UART2_TXD	TIMER_IO2	~
Bootstrap	~	~	~	~	~

P8.36-P8.38

Pin	P8.36	P8.37	~	P8.38	~
GPIO	1 8	1 106	1 11	1 105	1 9
BALL	AE20	Y27	AD21	Y29	AJ20
REG	0x00011C020	0x00011C1AC	0x00011C02C	0x00011C1A8	0x00011C024
Page	42	58	43	58	42
MODE 0	PRG1_PRU0_GPO7	RGMII6_RD2	PRG1_PRU0_GPO10	RGMII6_RD3	PRG1_PRU0_GPO8
1	PRG1_PRU0_GPI7	UART4_RTSn	PRG1_PRU0_GPI10	UART4_CTSn	PRG1_PRU0_GPI8
2	PRG1_IEP0_EDC_LATCH_IN1	~	PRG1_UART0_RTSn	~	~
3	PRG1_PWM3_B1	UART5_TXD	PRG1_PWM2_B1	UART5_RXD	PRG1_PWM2_A1
4	~	~	SPI6_CS2	CLKOUT	~
5	AUDIO_EXT_REFCLK1	TRC_DATA19	RMII5_CRS_DV	TRC_DATA18	RMII5_RXD0
6	MCAN4_TX	EHRPWM5_A	~	EHRPWM_TZn_IN4	MCAN4_RX
7	GPIO0_8	GPIO0_106	GPIO0_11	GPIO0_105	GPIO0_9
8	~	GPMC0_A22	GPMC0_BE0n_CLE	GPMC0_A21	GPMC0_OEn_REn
9	~	~	PRG1_IEP0_EDIO_DATA_IN_OUT29	~	~
10	~	~	OBSCLK2	~	VOUT0_DATA22
11	~	~	~	~	~
12	MCASP3_AXR1	MCASP11_AXR5	MCASP3_AFSX	MCASP11_AXR4	MCASP3_AXR2
13	~	~	~	~	~
14	~	~	~	~	~
Boot-strap	~	~	~	~	~

P8.39-P8.41

Pin	P8.39	P8.40	P8.41
GPIO	1 69	1 70	1 67
BALL	AC26	AA24	AD29
REG	0x00011C118	0x00011C11C	0x00011C110
Page	35	36	35
MODE 0	PRG0_PRU1_GPO6	PRG0_PRU1_GPO7	PRG0_PRU1_GPO4
1	PRG0_PRU1_GPI6	PRG0_PRU1_GPI7	PRG0_PRU1_GPI4
2	PRG0_RGMII2_RXC	PRG0_IEP1_EDC_LATCH_IN1	PRG0_RGMII2_RX_CTL
3	~	~	PRG0_PWM2_B2
4	RGMII4_RXC	SPI3_CS0	RGMII4_RX_CTL
5	RMII4_TXD0	~	RMII4_TXD1
6	~	MCAN11_TX	~
7	GPIO0_69	GPIO0_70	GPIO0_67
8	GPMC0_A25	GPMC0_AD9	GPMC0_A24
9	~	~	~
10	~	~	~
11	~	~	~
12	MCASP1_AXR3	MCASP1_AXR4	MCASP1_AXR2
13	~	~	~
14	~	UART2_TXD	~
Bootstrap	~	~	~

P8.42-P8.44

Pin	P8.42	P8.43	P8.44
GPIO	1 68	1 65	1 66
BALL	AB27	AD27	AC25
REG	0x00011C114	0x00011C108	0x00011C10C
Page	35	34	35
MODE 0	PRG0_PRU1_GPO5	PRG0_PRU1_GPO2	PRG0_PRU1_GPO3
1	PRG0_PRU1_GPI5	PRG0_PRU1_GPI2	PRG0_PRU1_GPI3
2	~	PRG0_RGMII2_RD2	PRG0_RGMII2_RD3
3	~	PRG0_PWM2_A2	~
4	~	RGMII4_RD2	RGMII4_RD3
5	~	RMII4_CRS_DV	RMII4_RX_ER
6	~	~	~
7	GPIO0_68	GPIO0_65	GPIO0_66
8	GPMC0_AD8	GPMC0_A23	~
9	~	~	~
10	~	~	~
11	~	~	~
12	MCASP1_ACLKX	MCASP1_ACLKR	MCASP1_AFSR
13	~	MCASP1_AXR10	MCASP1_AXR11
14	~	~	~
Bootstrap	BOOTMODE6	~	~

P8.45-P8.46

Pin	P8.45	P8.46
GPIO	1 79	1 80
BALL	AG29	Y25
REG	0x00011C140	0x00011C144
Page	38	38
MODE 0	PRG0_PRU1_GPO16	PRG0_PRU1_GPO17
1	PRG0_PRU1_GPI16	PRG0_PRU1_GPI17
2	PRG0_RGMII2_TXC	PRG0_IEP1_EDC_SYNC_OUT1
3	PRG0_PWM1_A2	PRG0_PWM1_B2
4	RGMII4_TXC	SPI3_CLK
5	~	~
6	~	~
7	GPIO0_79	GPIO0_80
8	~	GPMC0_AD13
9	~	~
10	~	~
11	~	~
12	MCASP2_AXR2	MCASP2_AXR3
13	~	~
14	~	~
Bootstrap	~	BOOTMODE3

Connector P9

The following tables show the pinout of the **P9** expansion header. The SW is responsible for setting the default function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **GPIO** row is the expected gpio identifier number in the Linux kernel.

Each row includes the gpiochipX and pinY in the format of *X Y*. You can use these values to directly control the GPIO pins with the commands shown below.

```
# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset X Y=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset X Y=0
```

(continues on next page)

(continued from previous page)

```
For Example:

+---------+----------+
| Pin     | P9.11    |
+=========+==========+
| GPIO    | 1 1      |
+---------+----------+

Use the commands below for controlling this pin (P9.11) where X = 1 and Y = 1

# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset 1 20=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset 1 20=0
```

The **BALL** row is the pin number on the processor.

The **REG** row is the offset of the control register for the processor pin.

The **MODE #** rows are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

If included, the **2nd BALL** row is the pin number on the processor for a second processor pin connected to the same pin on the expansion header. Similarly, all row headings starting with **2nd** refer to data for this second processor pin.

Important: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

P9.E1-P9.E4

E1	E2	E3	E4
USB1 DP	USB1 DN	VSYS_5V0	GND

P9.01-P9.05

P9.01	P9.02	P9.03	P9.04	P9.05
GND	GND	VOUT_3V3	VOUT_3V3	VIN

P9.06-P9.10

P9.06	P9.07	P9.08	P9.09	P9.10
VIN	VOUT_SYS	VOUT_SYS	RESET#	RESET#

P9.11-P9.13

Pin	P9.11	P9.12	P9.13
GPIO	1 1	1 45	1 2
BALL	AC23	AE27	AG22
REG	0x00011C004	0x00011C0B8	0x00011C008
Page	39	29	40
MODE 0	PRG1_PRU0_GPO0	PRG0_PRU0_GPO2	PRG1_PRU0_GPO1
1	PRG1_PRU0_GPI0	PRG0_PRU0_GPI2	PRG1_PRU0_GPI1
2	PRG1_RGMII1_RD0	PRG0_RGMII1_RD2	PRG1_RGMII1_RD1
3	PRG1_PWM3_A0	PRG0_PWM2_A0	PRG1_PWM3_B0
4	RGMII1_RD0	RGMII3_RD2	RGMII1_RD1
5	RMII1_RXD0	RMII3_CRS_DV	RMII1_RXD1
6	~	~	~
7	GPIO0_1	GPIO0_45	GPIO0_2
8	GPMC0_BE1n	UART3_RXD	GPMC0_WAIT0
9	RGMII7_RD0	~	RGMII7_RD1
10	~	~	~
11	~	~	~
12	MCASP6_ACLKX	MCASP0_ACLKR	MCASP6_AFSX
13	~	~	~
14	UART0_RXD	~	UART0_TXD
Bootstrap	~	~	~

P9.14-P9.16

Pin	P9.14	P9.15	P9.16
GPIO	1 93	1 47	1 94
BALL	U27	AD25	U24
REG	0x00011C178	0x00011C0C0	0x00011C17C
Page	56	30	56
MODE 0	RGMII5_RD3	PRG0_PRU0_GPO4	RGMII5_RD2
1	UART3_CTSn	PRG0_PRU0_GPI4	UART3_RTSn
2	~	PRG0_RGMII1_RX_CTL	~
3	UART6_RXD	PRG0_PWM2_B0	UART6_TXD
4	VOUT1_DATA8	RGMII3_RX_CTL	VOUT1_DATA9
5	TRC_DATA6	RMII3_TXD1	TRC_DATA7
6	EHRPWM2_A	~	EHRPWM2_B
7	GPIO0_93	GPIO0_47	GPIO0_94
8	GPMC0_A9	~	GPMC0_A10
9	~	~	~
10	~	~	~
11	~	~	~
12	MCASP11_AXR0	MCASP0_AXR2	MCASP11_AXR1
13	~	~	~
14	~	~	~
Bootstrap	~	~	~

P9.17-P9.18

Pin	P9.17	~	P9.18	~
GPIO	1 28	1 115	1 40	1 120
BALL	AC21	AA3	AH22	Y2
REG	0x00011C074	0x00011C1D0	0x00011C0A4	0x00011C1E4
Page	49	67	53	68
MODE 0	PRG1_PRU1_GPO7	SPI0_D1	PRG1_PRU1_GPO19	SPI1_D1
1	PRG1_PRU1_GPI7	~	PRG1_PRU1_GPI19	~
2	PRG1_IEP1_EDC_LATCH_IN1	I2C6_SCL	PRG1_IEP1_EDC_SYNC_OUT0	I2C6_SDA
3	~	~	PRG1_PWM1_TZ_OUT	~
4	SPI6_CS0	~	SPI6_D1	~
5	RMII6_RX_ER	~	RMII6_TXD1	~
6	MCAN7_TX	~	PRG1_ECAP0_IN_APWM_OUT	~
7	GPIO0_28	GPIO0_115	GPIO0_40	GPIO0_120
8	~	~	~	PRG0_IEP1_EDC_SYNC_OUT0
9	~	~	~	~
10	VOUT0_DATA7	~	VOUT0_PCLK	~
11	VPFE0_DATA15	~	~	~
12	MCASP4_AXR1	~	MCASP5_AXR1	~
13	~	~	~	~
14	UART3_TXD	~	~	~
Bootstrap	~	~	~	~

P9.19-P9.20

Pin	P9.19	~	P9.20	~
GPIO	2 1	1 78	2 2	1 77
BALL	W5	AF29	W6	AE25
REG	0x00011C208	0x00011C13C	0x00011C20C	0x00011C138
Page	19	38	19	37
MODE 0	MCAN0_RX	PRG0_PRU1_GPO15	MCAN0_TX	PRG0_PRU1_GPO14
1	~	PRG0_PRU1_GPI15	~	PRG0_PRU1_GPI14
2	~	PRG0_RGMII2_TX_CTL	~	PRG0_RGMII2_TD3
3	~	PRG0_PWM1_B1	~	PRG0_PWM1_A1
4	I2C2_SCL	RGMII4_TX_CTL	I2C2_SDA	RGMII4_TD3
5	~	~	~	~
6	~	~	~	~
7	GPIO1_1	GPIO0_78	GPIO1_2	GPIO0_77
8	~	~	~	~
9	~	~	~	~
10	~	~	~	~
11	~	~	~	~
12	~	MCASP2_AXR1	~	MCASP2_AXR0
13	~	~	~	~
14	~	UART2_RTSn	~	UART2_CTSn
Bootstrap	~	~	~	~

P9.21-P9.22

Pin	P9.21	~	P9.22	~
GPIO	1 39	1 90	1 38	1 91
BALL	AJ22	U28	AC22	U29
REG	0x00011C0A0	0x00011C16C	0x00011C09C	0x00011C170
Page	52	56	52	54
MODE 0	PRG1_PRU1_GPO18	RGMII5_TD0	PRG1_PRU1_GPO17	RGMII5_TXC
1	PRG1_PRU1_GPI18	RMII7_TXD0	PRG1_PRU1_GPI17	RMII7_TX_EN
2	PRG1_IEP1_EDC_LATCH_IN0	I2C3_SDA	PRG1_IEP1_EDC_SYNC_OUT1	I2C6_SCL
3	PRG1_PWM1_TZ_IN	~	PRG1_PWM1_B2	~
4	SPI6_D0	VOUT1_DATA5	SPI6_CLK	VOUT1_DATA6
5	RMII6_TXD0	TRC_DATA3	RMII6_TX_EN	TRC_DATA4
6	PRG1_ECAP0_SYNC_IN	EHRPWM1_A	PRG1_ECAP0_SYNC_OUT	EHRPWM1_B
7	GPIO0_39	GPIO0_90	GPIO0_38	GPIO0_91
8	~	GPMC0_A6	~	GPMC0_A7
9	VOUT0_VP2_VSYNC	~	VOUT0_VP2_DE	~
10	VOUT0_VSYNC	~	VOUT0_DE	~
11	~	~	VPFE0_DATA10	~
12	MCASP5_AXR0	MCASP11_AFSX	MCASP5_AFSX	MCASP10_AXR2
13	~	~	~	~
14	VOUT0_VP0_VSYNC	~	VOUT0_VP0_DE	~
Bootstrap	~	~	BOOTMODE1	~

P9.23-P9.25

Pin	P9.23	P9.24	~	P9.25	~
GPIO	1 10	1 119	1 13	1 127	1 104
BALL	AG20	Y5	AJ24	AC4	W26
REG	0x00011C028	0x00011C1E0	0x00011C034	0x00011C200	0x00011C1A4
Page	42	68	43	69	54
MODE 0	PRG1_PRU0_GPO9	SPI1_D0	PRG1_PRU0_GPO12	UART1_CTSn	RGMII6_RXC
1	PRG1_PRU0_GPI9	UART5_RTSn	PRG1_PRU0_GPI12	MCAN3_RX	~
2	PRG1_UART0_CTSn	I2C4_SCL	PRG1_RGMII1_TD1	~	~
3	PRG1_PWM3_TZ_IN	UART2_TXD	PRG1_PWM0_A0	~	AUDIO_EXT_REFCLK2
4	SPI6_CS1	~	RGMII1_TD1	SPI2_D0	VOUT1_DE
5	RMII5_RXD1	~	~	EQEP0_S	TRC_DATA17
6	~	~	MCAN4_RX	~	EHRPWM4_B
7	GPIO0_10	GPIO0_119	GPIO0_13	GPIO0_127	GPIO0_104
8	GPMC0_ADVn_ALE	PRG0_IEP1_EDC_LATCH_IN0	~	~	GPMC0_A20
9	PRG1_IEP0_EDIO_DATA_IN_OUT28	~	RGMII7_TD1	~	VOUT1_VP0_DE
10	VOUT0_DATA23	~	VOUT0_DATA17	~	~
11	~	~	VPFE0_DATA1	~	~
12	MCASP3_ACLKX	~	MCASP7_AFSX	~	MCASP10_AXR7
13	~	~	~	~	~
14	~	~	~	~	~
Bootstrap	~	~	~	~	~

P9.26-P9.27

Pin	P9.26	~	P9.27	~
GPIO	1 118	1 12	1 46	1 124
BALL	Y1	AF24	AD26	AB1
REG	0x00011C1DC	0x00011C030	0x00011C0BC	0x00011C1F4
Page	67	43	30	69
MODE 0	SPI1_CLK	PRG1_PRU0_GPO11	PRG0_PRU0_GPO3	UART0_RTSn
1	UART5_CTSn	PRG1_PRU0_GPI11	PRG0_PRU0_GPI3	TIMER_IO7
2	I2C4_SDA	PRG1_RGMII1_TD0	PRG0_RGMII1_RD3	SPI0_CS3
3	UART2_RXD	PRG1_PWM3_TZ_OUT	PRG0_PWM3_A2	MCAN2_TX
4	~	RGMII1_TD0	RGMII3_RD3	SPI2_CLK
5	~	~	RMII3_RX_ER	EQEP0_B
6	~	MCAN4_TX	~	~
7	GPIO0_118	GPIO0_12	GPIO0_46	GPIO0_124
8	PRG0_IEP0_EDC_SYNC_OUT0	~	UART3_TXD	~
9	~	RGMII7_TD0	~	~
10	~	VOUT0_DATA16	~	~
11	~	VPFE0_DATA0	~	~
12	~	MCASP7_ACLKX	MCASP0_AFSR	~
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	~	~

P9.28-P9.29

Pin	P9.28	~	P9.29	~
GPIO	2 11	1 43	2 14	1 53
BALL	U2	AF28	V5	AB25
REG	0x00011C230	0x00011C0B0	0x00011C23C	0x00011C0D8
Page	18	29	68	31
MODE 0	ECAP0_IN_APWM_OUT	PRG0_PRU0_GPO0	TIMER_IO1	PRG0_PRU0_GPO10
1	SYNC0_OUT	PRG0_PRU0_GPI0	ECAP2_IN_APWM_OUT	PRG0_PRU0_GPI10
2	CPTS0_RFT_CLK	PRG0_RGMII1_RD0	OBSCLK0	PRG0_UART0_RTSn
3	~	PRG0_PWM3_A0	~	PRG0_PWM2_B1
4	SPI2_CS3	RGMII3_RD0	~	SPI3_CS2
5	I3C0_SDAPULLEN	RMII3_RXD1	~	PRG0_IEP0_EDIO_DATA_IN_OUT29
6	SPI7_CS0	~	SPI7_D1	MCAN10_RX
7	GPIO1_11	GPIO0_43	GPIO1_14	GPIO0_53
8	~	~	~	GPMC0_AD4
9	~	~	~	~
10	~	~	~	~
11	~	~	~	~
12	~	MCASP0_AXR0	~	MCASP0_AFSX
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	BOOTMODE5	~

P9.30-P9.31

Pin	P9.30	~	P9.31	~
GPIO	2 13	1 44	2 12	1 52
BALL	V6	AE28	U3	AB26
REG	0x00011C238	0x00011C0B4	0x00011C234	0x00011C0D4
Page	68	29	18	31
MODE 0	TIMER_IO0	PRG0_PRU0_GPO1	EXT_REFCLK1	PRG0_PRU0_GPO9
1	ECAP1_IN_APWM_OUT	PRG0_PRU0_GPI1	SYNC1_OUT	PRG0_PRU0_GPI9
2	SYSCLKOUT0	PRG0_RGMII1_RD1	~	PRG0_UART0_CTSn
3	~	PRG0_PWM3_B0	~	PRG0_PWM3_TZ_IN
4	~	RGMII3_RD1	~	SPI3_CS1
5	~	RMII3_RXD0	~	PRG0_IEP0_EDIO_DATA_IN_OUT28
6	SPI7_D0	~	SPI7_CLK	MCAN10_TX
7	GPIO1_13	GPIO0_44	GPIO1_12	GPIO0_52
8	~	~	~	GPMC0_AD3
9	~	~	~	~
10	~	~	~	~
11	~	~	~	~
12	~	MCASP0_AXR1	~	MCASP0_ACLKX
13	~	~	~	~
14	~	~	~	UART6_TXD
Bootstrap	BOOTMODE4	~	~	~

P9.32-P9.35

P9.32	P9.34
VDD_ADC	GND

Pin	P9.33	~	P9.35	~
GPIO	~	1 50	~	1 55
BALL	K24	AC28	K29	AH27
REG	0x00011C140	0x00011C0CC	0x00011C148	0x00011C0E0
Page	20	31	20	32
MODE 0	MCU_ADC0_AIN4	PRG0_PRU0_GPO7	MCU_ADC0_AIN6	PRG0_PRU0_GPO12
1	~	PRG0_PRU0_GPI7	~	PRG0_PRU0_GPI12
2	~	PRG0_IEP0_EDC_LATCH_IN1	~	PRG0_RGMII1_TD1
3	~	PRG0_PWM3_B1	~	PRG0_PWM0_A0
4	~	PRG0_ECAP0_SYNC_IN	~	RGMII3_TD1
5	~	~	~	~
6	~	MCAN9_TX	~	~
7	~	GPIO0_50	~	GPIO0_55
8	~	GPMC0_AD1	~	~
9	~	~	~	~
10	~	~	~	DSS_FSYNC0
11	~	~	~	~
12	~	MCASP0_AXR5	~	MCASP0_AXR8
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	~	~

P9.36-P9.37

Pin	P9.36	~	P9.37	~
GPIO	~	1 56	~	1 57
BALL	K27	AH29	K28	AG28
REG	0x00011C144	0x00011C0E4	0x00011C138	0x00011C0E8
Page	20	32	20	32
MODE 0	MCU_ADC0_AIN5	PRG0_PRU0_GPO13	MCU_ADC0_AIN2	PRG0_PRU0_GPO14
1	~	PRG0_PRU0_GPI13	~	PRG0_PRU0_GPI14
2	~	PRG0_RGMII1_TD2	~	PRG0_RGMII1_TD3
3	~	PRG0_PWM0_B0	~	PRG0_PWM0_A1
4	~	RGMII3_TD2	~	RGMII3_TD3
5	~	~	~	~
6	~	~	~	~
7	~	GPIO0_56	~	GPIO0_57
8	~	~	~	UART4_RXD
9	~	~	~	~
10	~	DSS_FSYNC2	~	~
11	~	~	~	~
12	~	MCASP0_AXR9	~	MCASP0_AXR10
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	~	~

P9.38-P9.39

Pin	P9.38	~	P9.39	~
GPIO	~	1 58	~	1 54
BALL	L28	AG27	K25	AJ28
REG	0x00011C13C	0x00011C0EC	0x00011C130	0x00011C0DC
Page	~	33	20	32
MODE 0	MCU_ADC0_AIN3	PRG0_PRU0_GPO15	MCU_ADC0_AIN0	PRG0_PRU0_GPO11
1	~	PRG0_PRU0_GPI15	~	PRG0_PRU0_GPI11
2	~	PRG0_RGMII1_TX_CTL	~	PRG0_RGMII1_TD0
3	~	PRG0_PWM0_B1	~	PRG0_PWM3_TZ_OUT
4	~	RGMII3_TX_CTL	~	RGMII3_TD0
5	~	~	~	~
6	~	~	~	~
7	~	GPIO0_58	~	GPIO0_54
8	~	UART4_TXD	~	~
9	~	~	~	CLKOUT
10	~	DSS_FSYNC3	~	~
11	~	~	~	~
12	~	MCASP0_AXR11	~	MCASP0_AXR7
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	~	~

P9.40-P9.42

Pin	P9.40	~	P9.41	P9.42	~
GPIO	~	1 81	2 0	1 123	1 18
BALL	K26	AA26	AD5	AC2	AJ21
REG	0x00011C134	0x00011C148	0x00011C204	0x00011C1F0	0x00011C04C
Page	20	38	69	68	45
MODE 0	MCU_ADC0_AIN1	PRG0_PRU1_GPO18	UART1_RTSn	UART0_CTSn	PRG1_PRU0_GPO17
1	~	PRG0_PRU1_GPI18	MCAN3_TX	TIMER_IO6	PRG1_PRU0_GPI17
2	~	PRG0_IEP1_EDC_LATCH_IN0	~	SPI0_CS2	PRG1_IEP0_EDC_SYNC_OUT1
3	~	PRG0_PWM1_TZ_IN	~	MCAN2_RX	PRG1_PWM0_B2
4	~	SPI3_D0	SPI2_D1	SPI2_CS0	~
5	~	~	EQEP0_I	EQEP0_A	RMII5_TXD1
6	~	MCAN12_TX	~	~	MCAN5_TX
7	~	GPIO0_81	GPIO1_0	GPIO0_123	GPIO0_18
8	~	GPMC0_AD14	~	~	~
9	~	~	~	~	~
10	~	~	~	~	~
11	~	~	~	~	VPFE0_DATA6
12	~	MCASP2_AFSX	~	~	MCASP3_AXR3
13	~	~	~	~	~
14	~	UART2_RXD	~	~	~
Bootstrap	~	~	~	~	~

P9.43-P9.46

P9.43	P9.44	P9.45	P9.46
GND	GND	GND	GND

Cape Board Support

BeagleBone AI-64 has the ability to accept up to four EEPROM addressable expansion boards or capes stacked onto the expansion headers. The word cape comes from the shape of the expansion board for BeagleBone boards as it is fitted around the Ethernet connector on the main board. For BeagleBone this notch acts as a key to ensure proper orientation of the cape. On AI-64 you can see a clear silkscreen marking for the cape orientation. Most of BeagleBone capes can be used with your BeagleBone AI-64 also like shown in *BeagleBone AI-64 cape placement* below.

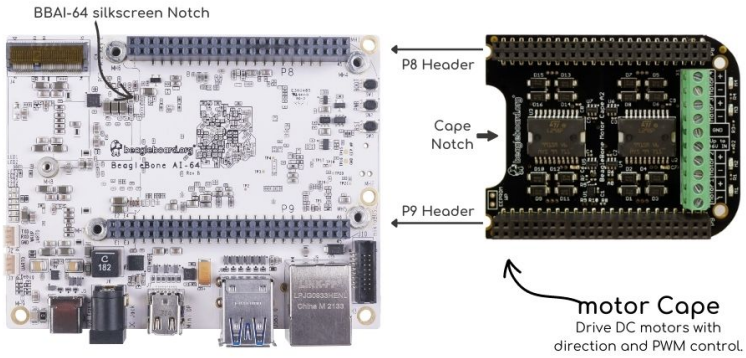


Fig. 3.20: BeagleBone AI-64 cape placement

This section describes the rules & guidelines for creating capes to ensure proper operation with BeagleBone AI-64 and proper interoperability with other capes that are intended to coexist with each other. Co-existence is not a requirement and is in itself, something that is impossible to control or administer. But, people will be able to create capes that operate with other capes that are already available based on public information as it pertains to what pins and features each cape uses. This information will be able to be read from the EEPROM on each cape.

For those wanting to create their own capes this should not put limits on the creation of capes and what they can

do, but may set a few basic rules that will allow the software to administer their operation with BeagleBone AI-64. For this reason there is a lot of flexibility in the specification that we hope most people will find it liberating in the spirit of Open Source Hardware. On the other hand we are sure that there are others who would like to see tighter control, more details, more rules and much more order to the way capes are handled.

Over time, this specification will change and be updated, so please refer to the latest version of this manual prior to designing your own capes to get the latest information.

Warning: Do not apply voltage to any I/O pin when power is not supplied to the board. It will damage the processor and void the warranty.

BeagleBone AI-64 Cape Compatibility The expansion headers on BeagleBone Black and BeagleBone AI-64 provides similar pin configuration options on P8 and P9 expansion header pins thus provide cape compatibility to a certain extent. Which means most BeagleBone Black capes will also be compatible with BeeagleBone AI-64.

See *BeagleBone cape interface spec* for compatibility information.

EEPROM Each cape must have its own EEPROM containing information that will allow the software to identify the board and to configure the expansion headers pins during boot as needed. The one exception is proto boards intended for prototyping. They may or may not have an EEPROM on them. An EEPROM is required for all capes sold in order for them operate correctly when plugged into BeagleBone AI-64.

The address of the EEPROM will be set via either jumpers or a dipswitch on each expansion board. *Expansion board EEPROM without write protect* below is the design of the EEPROM circuit.

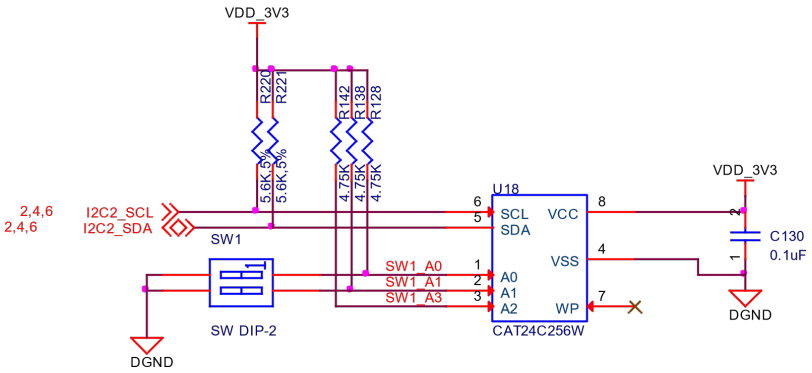


Fig. 3.21: Expansion board EEPROM without write protect

The addressing of this device requires two bytes for the address which is not used on smaller size EEPROMs, which only require only one byte. Other compatible devices may be used as well. Make sure the device you select supports 16 bit addressing. The part package used is at the discretion of the cape designer.

EEPROM Address In order for each cape to have a unique address, a board ID scheme is used that sets the address to be different depending on the setting of the dipswitch or jumpers on the capes. A two position dipswitch or jumpers is used to set the address pins of the EEPROM.

It is the responsibility of the user to set the proper address for each board and the position in the stack that the board occupies has nothing to do with which board gets first choice on the usage of the expansion bus signals. The process for making that determination and resolving conflicts is left up to the SW and, as of this moment in time, this method is a something of a mystery due to the new Device Tree methodology introduced in the 3.8 kernel.

Address line A2 is always tied high. This sets the allowable address range for the expansion cards to *0x54* to**0x57**. All other I2C addresses can be used by the user in the design of their capes. But, these addresses must not be used other than for the board EEPROM information. This also allows for the inclusion of EEPROM

devices on the cape if needed without interfering with this EEPROM. It requires that A2 be grounded on the EEPROM not used for cape identification.

I2C Bus The EEPROMs on each expansion board are connected to I2C2 on connector P9 pins 19 and 20. For this reason I2C2 must always be left connected and should not be changed by SW to remove it from the expansion header pin mux settings. If this is done, the system will be unable to detect the capes.

The I2C signals require pullup resistors. Each board must have a 5.6K resistor on these signals. With four capes installed this will result in an effective resistance of 1.4K if all capes were installed and all the resistors used were exactly 5.6K. As more capes are added the resistance is reduced to overcome capacitance added to the signals. When no capes are installed the internal pullup resistors must be activated inside the processor to prevent I2C timeouts on the I2C bus.

The I2C2 bus may also be used by capes for other functions such as I/O expansion or other I2C compatible devices that do not share the same address as the cape EEPROM.

EEPROM Write Protect The design in *Expansion board EEPROM with write protect* has the write protect disabled. If the write protect is not enabled, this does expose the EEPROM to being corrupted if the I2C2 bus is used on the cape and the wrong address written to. It is recommended that a write protection function be implemented and a Test Point be added that when grounded, will allow the EEPROM to be written to. To enable write operation, Pin 7 of the EEPROM must be tied to ground.

When not grounded, the pin is HI via pullup resistor R210 and therefore write protected. Whether or not Write Protect is provided is at the discretion of the cape designer.

Todo:

- Variable & MAC Memory
- VSYS_IO_3V3

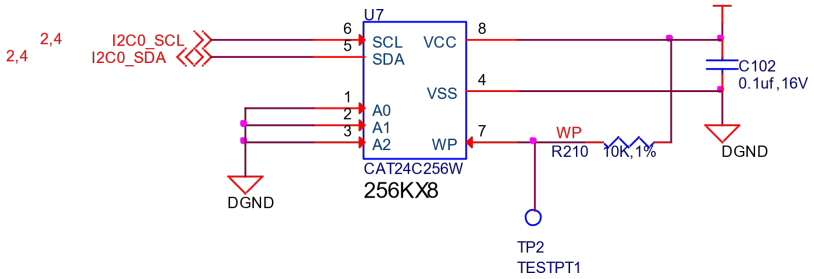


Fig. 3.22: Expansion board EEPROM with write protect

EEPROM Data Format *Expansion Board EEPROM* shows the format of the contents of the expansion board EEPROM. Data is stored in Big Endian with the least significant value on the right. All addresses read as a single byte data from the EEPROM, but two byte addressing is used. ASCII values are intended to be easily read by the user when the EEPROM contents are dumped.

Todo: *Clean/Update table*

Table 3.3: Expansion Board EEPROM

Name	Offset	Size (bytes)	Contents
Header	0	4	0xAA, 0x55, 0x33, 0xEE
EEPROM Revision	4	2	Revision number of the overall format of this EEPROM in ASCII =A1
Board Name	6	32	Name of board in ASCII so user can read it when the EEPROM is dumped. Up to developer of the board as to what they call the board..
Version	38	4	Hardware version code for board in ASCII.Version format is up to the developer.i.e. 02.1...00A1....10A0
Manufacturer	42	16	ASCII name of the manufacturer. Company or individual's name.
Part Number	58	16	ASCII Characters for the part number. Up to maker of the board.
Number of Pins	74	2	Number of pins used by the daughter board including the power pins used. Decimal value of total pins 92 max, stored in HEX.
Serial Number	76	12	Serial number of the board. This is a 12 character string which is: **WWYY&&&&nnnn** where, WW = 2 digit week of the year of production, YY = 2 digit year of production , &&&&=Assembly code to let the manufacturer document the assembly number or product. A way to quickly tell from reading the serial number what the board is. Up to the developer to determine. nnnn = incrementing board number for that week of production
Pin Usage	88	148	Two bytes for each configurable pins of the 74 pins on the expansion connectors, MSB LSB Bit order: 15..14 ..... 1..0 Bit 15....Pin is used or not...0=Unused by cape 1=Used by cape Bit 14-13...Pin Direction.....1 0=Output 01=Input 11=BDIR Bits 12-7...Reserved.....should be all zeros Bit 6....Slew Rate .....0=Fast 1=Slow Bit 5....Rx Enable.....0=Disabled 1=Enabled Bit 4....Pull Up/Dn Select....0=Pulldown 1=PullUp Bit 3....Pull Up/DN enabled...0=Enabled 1=Disabled Bits 2-0 ...Mux Mode Selection...Mode 0-7
VDD_3V3B Current	236	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500mA=0x05 0xDC 325mA=0x01 0x45
VDD_5V Current	238	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500mA=0x05 0xDC 325mA=0x01 0x45
SYS_5V Current	240	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500mA=0x05 0xDC 325mA=0x01 0x45
DC Supplied	242	2	Indicates whether or not the board is supplying voltage on the VDD_5V rail and the current rating 000=No 1-0xFFFF is the current supplied storing the decimal quivalent in HEX format
Available	244	32543	Available space for other non-volatile codes/data to be used as needed by the manufacturer or SW driver. Could also store presets for use by SW.

Todo: Align with other boards and migrate away from pin usage entries for BeagleBone Black expansion

Pin Usage Consideration This section covers things to watch for when hooking up to certain pins on the expansion headers.

Expansion Connectors A combination of male and female headers is used for access to the expansion headers on the main board. There are three possible mounting configurations for the expansion headers:

- **Single** -no board stacking but can be used on the top of the stack.
- **Stacking-up** to four boards can be stacked on top of each other.
- **Stacking with signal stealing-up** to three boards can be stacked on top of each other, but certain boards will not pass on the signals they are using to prevent signal loading or use by other cards in the stack.

The following sections describe how the connectors are to be implemented and used for each of the different configurations.

Non-Stacking Headers-Single Cape For non-stacking capes single configurations or where the cape can be the last board on the stack, the two 46 pin expansion headers use the same connectors. *Single expansion connector* is a picture of the connector. These are dual row 23 position 2.54mm x 2.54mm connectors.



Fig. 3.23: Single expansion connector

The connector is typically mounted on the bottom side of the board as shown in *Single cape expansion connector on BeagleBone Proto Cape with EEPROM from onlogic* . These are very common connectors and should be easily located. You can also use two single row 23 pin headers for each of the dual row headers.

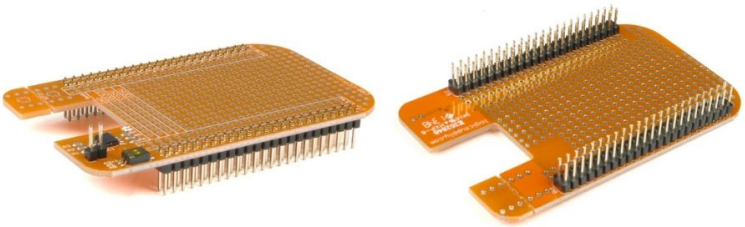


Fig. 3.24: Single cape expansion connector on BeagleBone Proto Cape with EEPROM from onlogic

It is allowed to only populate the pins you need. As this is a non-stacking configuration, there is no need for all headers to be populated. This can also reduce the overall cost of the cape. This decision is up to the cape designer.

For convenience listed in *Single Cape Connectors* are some possible choices for part numbers on this connector. They have varying pin lengths and some may be more suitable than others for your use. It should be noted, that the longer the pin and the further it is inserted into BeagleBone AI-64 connector, the harder it will be to remove due to the tension on 92 pins. This can be minimized by using shorter pins or removing those pins that are not used by your particular design. The first item in**Table 18** is on the edge and may not be the best solution. Overhang is the amount of the pin that goes past the contact point of the connector on BeagleBone AI-64

Table 3.4: Single Cape Connectors

SUPPLIER	PARTNUMBER	LENGTH(in)	OVERHANG(in)
Major League	TSHC-123-D-03-145-G-LF	.145	.004
Major League	TSHC-123-D-03-240-G-LF	.240	.099
Major League	TSHC-123-D-03-255-G-LF	.255	.114

The G in the part number is a plating option. Other options may be used as well as long as the contact area is gold. Other possible sources are Sullins and Samtec for these connectors. You will need to ensure the depth into the connector is sufficient

Main Expansion Headers-Stacking For stacking configuration, the two 46 pin expansion headers use the same connectors. *Expansion Connector* is a picture of the connector. These are dual row 23 position 2.54mm x 2.54mm connectors.

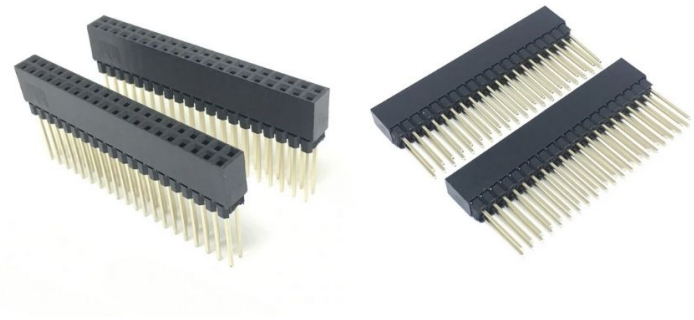


Fig. 3.25: Expansion Connector

The connector is mounted on the top side of the board with longer tails to allow insertion into BeagleBone AI-64. *Stacked cape expansion connector* is the connector configuration for the connector.

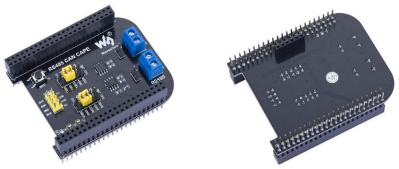


Fig. 3.26: Stacked cape expansion connector

For convenience listed in *Table 18* are some possible choices for part numbers on this connector. They have varying pin lengths and some may be more suitable than others for your use. It should be noted, that the longer the pin and the further it is inserted into BeagleBone AI-64 connector, the harder it will be to remove due to the tension on 92 pins. This can be minimized by using shorter pins. There are most likely other suppliers out there that will work for this connector as well. If anyone finds other suppliers of compatible connectors that work, let us know and they will be added to this document. The first item in **Table 19** is on the edge and may not be the best solution. Overhang is the amount of the pin that goes past the contact point of the connector on BeagleBone AI-64.

The third part listed in *Stacked Cape Connectors* will have insertion force issues.

Table 3.5: Stacked Cape Connectors

SUPPLIER	PARTNUMBER	TAIL LENGTH(in)	OVERHANG(in)
Major League	SSHQ-123-D-06-G-LF	.190	0.049
Major League	SSHQ-123-D-08-G-LF	.390	0.249
Major League	SSHQ-123-D-10-G-LF	.560	0.419

There are also different plating options on each of the connectors above. Gold plating on the contacts is the minimum requirement. If you choose to use a different part number for plating or availability purposes, make sure you do not select the "LT" option.

Other possible sources are Sullins and Samtec but make sure you select one that has the correct mating depth.

Stacked Capes w/Signal Stealing *Stacked with signal stealing expansion connector figure* is the connector configuration for stackable capes that does not provide all of the signals upwards for use by other boards. This is useful if there is an expectation that other boards could interfere with the operation of your board by exposing those signals for expansion. This configuration consists of a combination of the stacking and nonstacking style connectors.

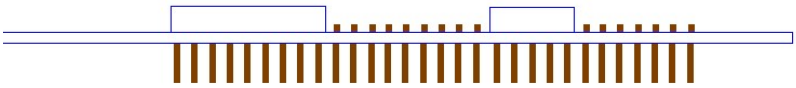


Fig. 3.27: Stacked with signal stealing expansion connector figure

Retention Force The length of the pins on the expansion header has a direct relationship to the amount of force that is used to remove a cape from BeagleBone AI-64. The longer the pins extend into the connector the harder it is to remove. There is no rule that says that if longer pins are used, that the connector pins have to extend all the way into the mating connector on BeagleBone AI-64, but this is controlled by the user and therefore is hard to control. We have also found that if you use gold pins, while more expensive, it makes for a smoother finish which reduces the friction.

This section will attempt to describe the tradeoffs and things to consider when selecting a connector and its pin length.

BeagleBone AI-64 Female Connectors *Connector Pin Insertion Depth* shows the key measurements used in calculating how much the pin extends past the contact point on the connector, what we call overhang.

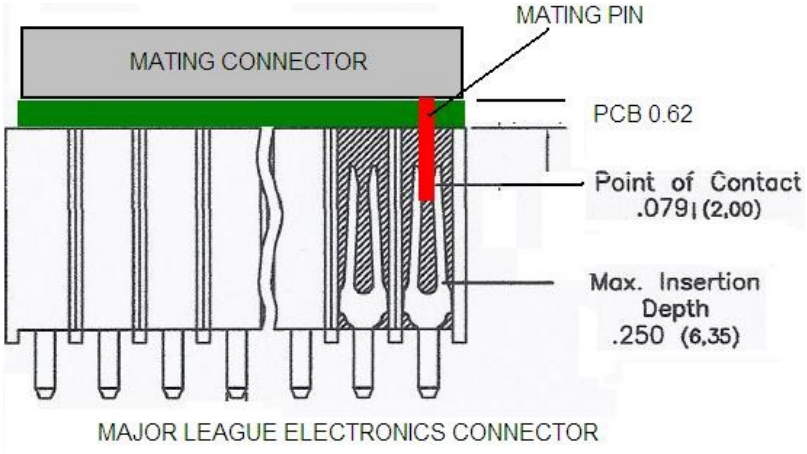


Fig. 3.28: Connector Pin Insertion Depth

To calculate the amount of the pin that extends past the Point of Contact, use the following formula:

Overhang=Total Pin Length- PCB thickness (.062) - contact point (.079)

The longer the pin extends past the contact point, the more force it will take to insert and remove the board. Removal is a greater issue than the insertion.

Signal Usage Based on the pin muxing capabilities of the processor, each expansion pin can be configured for different functions. When in the stacking mode, it will be up to the user to ensure that any conflicts are resolved between multiple stacked cards. When stacked, the first card detected will be used to set the pin muxing of each pin. This will prevent other modes from being supported on stacked cards and may result in them being inoperative.

In beaglebone-ai-64-connectors section of this document, the functions of the pins are defined as well as the pin muxing options. Refer to this section for more information on what each pin is. To simplify things, if you use the default name as the function for each pin and use those functions, it will simplify board design and reduce conflicts with other boards.

Interoperability is up to the board suppliers and the user. This specification does not specify a fixed function on any pin and any pin can be used to the full extent of the functionality of that pin as enabled by the processor.

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

Cape Power This section describes the power rails for the capes and their usage.

Main Board Power The *Expansion Voltages* describes the voltages from the main board that are available on the expansion connectors and their ratings. All voltages are supplied by connector**P9**. The current ratings listed are per pin.

Table 3.6: Expansion Voltages

Current	Name	P9	P9	Name	Current
250mA	VDD_3V3B	3	4	VDD_3V3B	250mA
1000mA	VDD_5V	5	6	VDD_5V	1000mA
250mA	SYS_5V	7	8	SYS_5V	250mA

The *VSYS_IO_3V3* rail is supplied by the LDO on BeagleBone AI-64 and is the primary power rail for expansion boards. If the power requirement for the capes exceeds the current rating, then locally generated voltage rail can be used. It is recommended that this rail be used to power any buffers or level translators that may be used.

DC_VDD_5V is the main power supply from the DC input jack. This voltage is not present when the board is powered via USB. The amount of current supplied by this rail is dependent upon the amount of current available. Based on the board design, this rail is limited to 1A per pin from the main board.

The *VSYS_5V0* rail is the main rail for the regulators on the main board. When powered from a DC supply or USB, this rail will be 5V. The available current from this rail depends on the current available from the USB and DC external supplies.

Expansion Board External Power A cape can have a jack or terminals to bring in whatever voltages may be needed by that board. Care should be taken not to let this voltage be fed back into any of the expansion header pins.

It is possible to provide 5V to the main board from an expansion board. By supplying a 5V signal into the *DC_VDD_5V* rail, the main board can be supplied. This voltage must not exceed 5V. You should not supply any voltage into any other pin of the expansion connectors. Based on the board design, this rail is limited to 1A per pin to BeagleBone AI-64.

There are several precautions that need to be taken when working with the expansion headers to prevent damage to the board.

1. *Do not apply any voltages to any I/O pins when the board is not powered on.*
2. *Do not drive any external signals into the I/O pins until after the VSYS_IO_3V3 rail is up.*
3. *Do not apply any voltages that are generated from external sources.*
4. *If voltages are generated from the DC_VDD_5V signal, those supplies must not become active until after the VSYS_IO_3V3 rail is up.*
5. *If you are applying signals from other boards into the expansion headers, make sure you power the board up after you power up the BeagleBone AI-64 or make the connections after power is applied on both boards.*

Powering the processor via its I/O pins can cause damage to the processor.

Todo: Add BeagleBone AI-64 cape mechanical characteristics**

Standard Cape Size *Cape board dimensions* shows the outline of the standard cape. The dimensions are in inches.

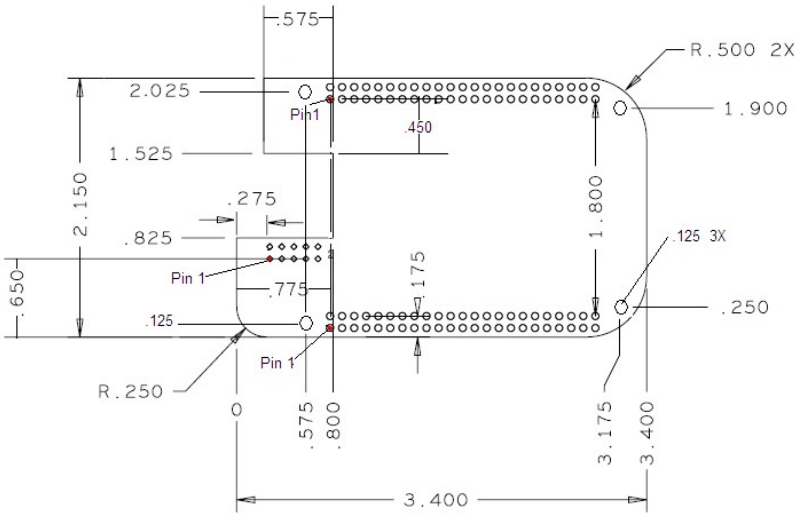


Fig. 3.29: Cape board dimensions

A notch is provided for BeagleBone Ethernet connector to stick up higher than the cape when mounted. This also acts as a key function to ensure that the cape is oriented correctly. Space is also provided to allow access to the user LEDs and reset button on BeagleBone board. On BeagleBone AI-64 board align it with the notch on the board silkscreen.

Extended Cape Size Capes larger than the standard board size are also allowed. A good example would be the new BeagleBone AI-64 robotics cape. There is no practical limit to the sizes of these types of boards. The notch is also optional, but it is up to the supplier to ensure that the cape is not plugged incorrectly on BeagleBone AI-64 such that damage would be cause to BeagleBone AI-64. Any such damage will be the responsibility of the supplier of such a cape to repair. As with all capes, the EEPROM is required and compliance with the power requirements must be adhered to.

3.4.2 RANDOM PRU STUFF THAT MIGHT NEED A HOME

Note: I don't want to blow this information away until I know no work went into it for TDA4VM. It is probably just AM3358 or AM5729 information. :-(

---

table-11 below shows which PRU-ICSS signals can be accessed on the BeagleBone AI-64 and on which connector and pins they are accessible from. Some signals are accessible on the same pins.

Table 3

	PIN	PROC	NAME
P8	11	R12	GPIO1_13
	12	T12	GPIO1_12
	15	U13	GPIO1_15
	16	V13	GPIO1_14
	20	V9	GPIO1_31
	21	U9	GPIO1_30
	27	U5	GPIO2_22
	28	V5	GPIO2_24
	29	R5	GPIO2_23
	39	T3	GPIO2_12
	40	T4	GPIO2_13
	41	T1	GPIO2_10
	42	T2	GPIO2_11
	43	R3	GPIO2_8
	44	R4	GPIO2_9
	45	R1	GPIO2_6
	46	R2	GPIO2_7
P9	17	A16	I2C1_SCL
	18	B16	I2C1_SDA
	19	D17	I2C2_SCL
	20	D18	I2C2_SDA
	21	B17	UART2_TXD
	22	A17	UART2_RXD
	24	D15	UART1_TXD
	25	A14	GPIO3_21footnote:[GPIO3_21 is also the 24.576MHZ clock input to the processor to enable HDMI audio.
	26	D16	UART1_RXD
	27	C13	GPIO3_19
	28	C12	SPI1_CS0
	29	B13	SPI1_D0
	30	D12	SPI1_D1
	31	A13	SPI1_SCLK

3.5 Demos and Tutorials

- *Edge AI*

3.5.1 Edge AI

Getting Started

Hardware setup BeagleBone® AI-64 has TI's TDA4VM SoC which houses dual core A72, high performance vision accelerators, video codec accelerators, latest C71x and C66x DSP, high bandwidth realtime IPs for capture and display, GPU, dedicated safety island security accelerators. The SoC is power optimized to provide best in class performance for perception, sensor fusion, localization and path planning tasks in robotics, industrial and automotive applications.

For more details visit https://www.ti.com/product/TDA4VM

BeagleBone® AI-64 BeagleBone® AI-64 brings a complete system for developing artificial intelligence (AI) and machine learning solutions with the convenience and expandability of the BeagleBone® platform and the peripherals on board to get started right away learning and building applications. With locally hosted, ready-to-use, open-source focused tool chains and development environment, a simple web browser, power source and network connection are all that need to be added to start building performance-optimized embedded applications. Industry-leading expansion possibilities are enabled through familiar BeagleBone® cape headers, with hundreds of open-source hardware examples and dozens of readily available embedded expansion options available off-the-shelf.

To run the demos on BeagleBone® AI-64 you will require,

- BeagleBone® AI-64
- USB camera (Any V4L2 compliant 1MP/2MP camera, Eg. Logitech C270/C920/C922)
- Full HD eDP/HDMI display
- Minimum 16GB high performance SD card
- 100Base-T Ethernet cable connected to internet
- UART cable
- External Power Supply or Power Accessory Requirements
    a. Nominal Output Voltage: 5VDC
    b. Maximum Output Current: 5000 mA

Connect the components to the SK as shown in the image.

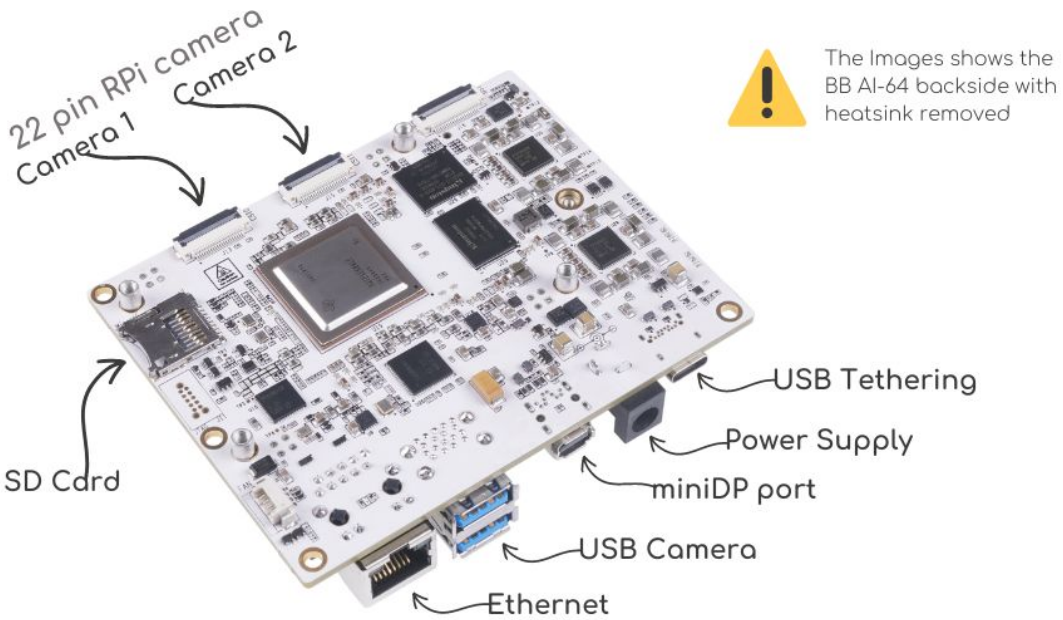


Fig. 3.30: BeagleBone® AI-64 for Edge AI connections

USB Camera UVC (USB video class) compliant USB cameras are supported on the BeagleBone® AI-64. The driver for the same is enabled in linux image. The linux image has been tested with C270/C920/C922 versions of Logitech USB cameras. Please refer to the TI Edge AI SDK FAQ to stream from multiple USB cameras simultaneously.

IMX219 Raw sensor **IMX219 camera module** from **Raspberry pi / Arducam** is supported by BeagleBone® AI-64. It is a 8MP sensor with no ISP, which can transmit raw SRGGB8 frames over CSI lanes at 1080p 60 fps. This camera module can be ordered from https://www.amazon.com/Raspberry-Pi-Camera-Module-Megapixel/dp/B01ER2SKFS The camera can be connected to any of the 2 RPi zero 22 pin camera headers on BB AI-64 as shown below

---

Todo: IMX219 CSI sensor connection with BeagleBone® AI-64 for Edge AI

Note that the headers have to be lifted up to connect the cameras

---

Note: To be updated By default IMX219 is disabled. After connecting the camera you can enable it by specifying the dtb overlay file in `/run/media/mmcblk0p1/uenv.txt` as below,

`name_overlays=k3-j721e-edgeai-apps.dtbo k3-j721e-sk-rpi-cam-imx219.dtbo`

Reboot the board after editing and saving the file.

Two RPi cameras can be connected to 2 headers for multi camera use-cases

Please refer *Camera sources (v4l2)* to know how to list all the cameras connected and select which one to use for the demo.

By default imx219 will be configured to capture at 8 bit, but it also supports 10 bit capture in 16 bit container. To use it in 10 bit mode, below steps are required:

- Modify the `/opt/edge_ai_apps/scripts/setup_cameras.sh` to set the format to 10 bit like below

```
CSI_CAM_0_FMT='[fmt:SRGGB8_1X10/1920x1080]'
CSI_CAM_1_FMT='[fmt:SRGGB8_1X10/1920x1080]'
```

- Change the imaging binaries to use 10 bit versions

```
mv /opt/imaging/imx219/dcc_2a.bin /opt/imaging/imx219/dcc_2a_8b.bin
mv /opt/imaging/imx219/dcc_viss.bin /opt/imaging/imx219/dcc_viss_8b.
↪bin
mv /opt/imaging/imx219/dcc_2a_10b.bin /opt/imaging/imx219/dcc_2a.bin
mv /opt/imaging/imx219/dcc_viss_10b.bin /opt/imaging/imx219/dcc_viss.
↪bin
```

- Set the input format in the `/opt/edge_ai_apps/configs/rpiV2_cam_example.yaml` as `rggb10`

Software setup

Preparing SD card image Download the `bullseye-xfce-edgeai-arm64` image from the links below and flash it to SD card using Balena etcher tool.

- To use via SD card: bbai64-debian-11.7-xfce-edgeai-arm64-2023-08-05-10gb.img.xz
- To flash on eMMC: bbai64-emmc-flasher-debian-11.7-xfce-edgeai-arm64-2023-08-05-10gb.img.xz

The Balena etcher tool can be installed either on Windows/Linux. Just download the etcher image and follow the instructions to prepare the SD card.

The etcher image is created for 16 GB SD cards, if you are using larger SD card, it is possible to expand the root filesystem to use the full SD card capacity using below steps

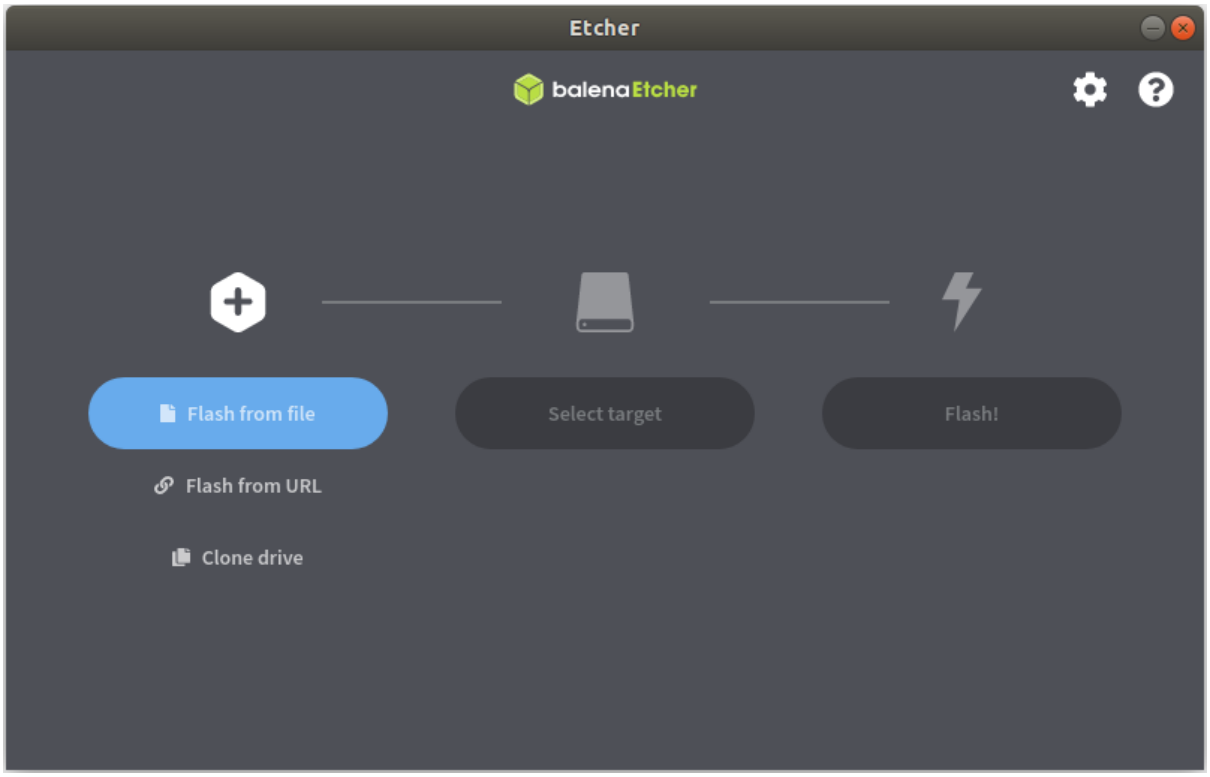


Fig. 3.31: Balena Etcher tool to flash SD card with Processor linux image Linux for Edge AI

```
#find the SD card device entry using lsblk (Eg: /dev/sdc)
#use the following commands to expand the filesystem
#Make sure you have write permission to SD card or run the commands as root

#Unmount the BOOT and rootfs partition before using parted tool
umount /dev/sdX1
umount /dev/sdX2

#Use parted tool to resize the rootfs partition to use
#the entire remaining space on the SD card
#You might require sudo permissions to execute these steps
parted -s /dev/sdX resizepart 2 '100%'
e2fsck -f /dev/sdX2
resize2fs /dev/sdX2

#replace /dev/sdX in above commands with SD card device entry
```

Power ON and Boot Ensure that the power supply is disconnected before inserting the SD card. Once the SD card is firmly inserted in its slot and the board is powered ON, the board will take less than 20sec to boot and display a wallpaper as shown in the image below.

Todo: BeagleBone® AI-64 wallpaper upon boot

You can also view the boot log by connecting the UART cable to your PC and use a serial port communications program.

For **Linux OS minicom** works well. Please refer to the below documentation on 'minicom' for more details.

https://help.ubuntu.com/community/Minicom

When starting minicom, turn on the colors options like below:

```
sudo minicom -D /dev/ttyUSB2 -c on
```

For **Windows OS Tera Term** works well. Please refer to the below documentation on 'TeraTerm' for more details

https://learn.sparkfun.com/tutorials/terminal-basics/tera-term-windows

---

Note: Baud rate should be configured to 115200 bps in serial port communication program. You may not see any log in the UART console if you connect to it after the booting is complete or login prompt may get lost in between boot logs, press ENTER to get login prompt

---

As part of the linux systemd `/opt/edge_ai_apps/init_script.sh` is executed which does the below,

- This kills weston compositor which holds the display pipe. This step will make the wallpaper showing on the display disappear and come back
- The display pipe can now be used by 'kmssink' GStreamer element while running the demo applications.
- The script can also be used to setup proxies if connected behind a firewall.

Once Linux boots login as `root` user with no password.

Connect remotely If you don't prefer the UART console, you can also access the device with the IP address that is shown on the display.

With the IP address one can ssh directly to the board, view the contents and run the demos.

For best experience we recommend using VSCode which can be downloaded from here.

https://code.visualstudio.com/download

You also require the "Remote development extension pack" installed in VSCode as mentioned here:

https://code.visualstudio.com/docs/remote/ssh

Todo: Microsoft Visual Studio Code for connecting to BeagleBone® AI-64 for Edge AI via SSH

Running Simple demos

This chapter describes how to run Python and C++ demo applications in edge_ai_apps with live camera and display.

Note: Please note that the Python demos are useful for quick prototyping while C++ demos are similar by design but tuned for performance.

Running Python based demo applications Python based demos are simple executable scripts written for image classification, object detection and semantic segmentation. Demos are configured using a YAML file. Details on configuration file parameters can be found in *Demo Configuration file*

Sample configuration files for out of the box demos can be found in `edge_ai_apps/configs` this folder also contains a template config file which has brief info on each configurable parameter `edge_ai_apps/configs/app_config_template.yaml`

Here is how a Python based image classification demo can be run,

```
# go to edge-ai-apps folder
debian@beaglebone:~$ cd /opt/edge_ai_apps/apps_python

# enable root (password: temppwd)
debian@beaglebone:~$ sudo su
[sudo] password for beaglebone:

# use edge-ai-apps
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# sudo ./app_edgeai.py ../
↪configs/image_classification.yaml
```

The demo captures the input frames from connected USB camera and passes through pre-processing, inference and post-processing before sent to display. Sample output for image classification and object detection demos are as below,



To exit the demo press Ctrl+C.

Building and running C++ based demo applications C++ apps needs to be built directly on target and requires header files of different deep-learning runtime framework and its dependencies which are installed in the setup script. The setup script builds the C++ apps when executed. However one can also follow below steps to clean build C++ apps

```
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# rm -rf build bin lib
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# mkdir build
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# cd build
debian@beaglebone:/opt/edge_ai_apps/apps_cpp/build# cmake ..
debian@beaglebone:/opt/edge_ai_apps/apps_cpp/build# make -j2
```

Run the demo once the application is successfully built

```
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# ./bin/Release/app_edgeai ../
↪configs/image_classification.yaml
```

To exit the demo press Ctrl+C.

Note: Both Python and C++ applications are similar by construction and can accept the same config file and command line arguments

---

Note: The C++ apps built on Yocto Linux may not run in Docker as there could be a mismatch in Glib and other related tools. So its **highly recommended** to rebuild the C++ apps within the Docker environment.

DL models for Edge Inference

Model Downloader Tool TI Model Zoo is a large collection of deep learning models validated to work on TI processors for edge AI. It hosts several pre-compiled model artifacts for TI hardware.

Use the **Model Downloader Tool** to download more models on target as shown,

```
debian@beaglebone:/opt/edge_ai_apps# ./download_models.sh
```

The script will launch an interactive menu showing the list of available, pre-imported models for download. The downloaded models will be placed under `/opt/model_zoo/` directory

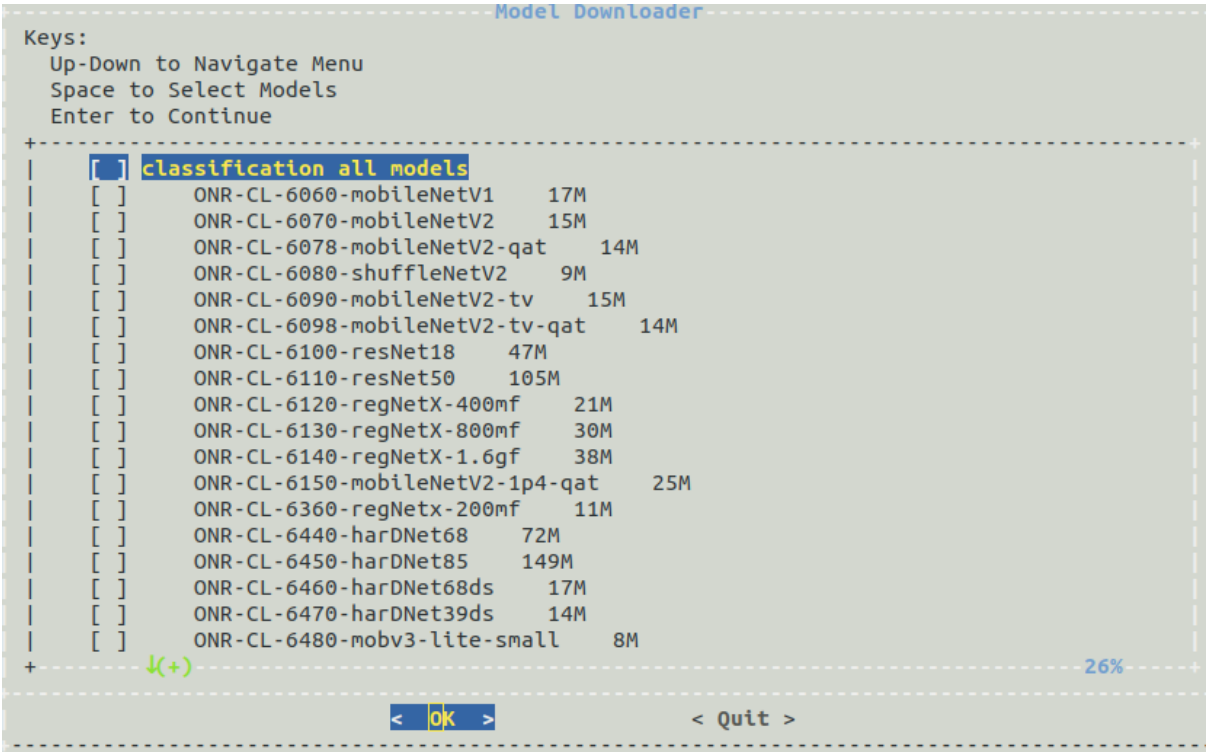


Fig. 3.32: Model downloader tool menu option to download models

The script can also be used in a non-interactive way as shown below:

```
debian@beaglebone:/opt/edge_ai_apps# ./download_models.sh --help
```

Import Custom Models The BeagleBone® AI-64 Linux for Edge AI also supports importing pre-trained custom models to run inference on target.

The SDK makes use of pre-compiled DNN (Deep Neural Network) models and performs inference using various OSRT (open source runtime) such as TFLite runtime, ONNX runtime and Neo AI-DLR. In order to infer a DNN, SDK expects the DNN and associated artifacts in the below directory structure.

```
TFL-OD-2010-ssd-mobV2-coco-mlperf-300x300
|
├── param.yaml
|
├── artifacts
|   ├── 264_tidl_io_1.bin
|   ├── 264_tidl_net.bin
|   ├── 264_tidl_net.bin.layer_info.txt
|   ├── 264_tidl_net.bin_netLog.txt
|   ├── 264_tidl_net.bin.svg
```

(continues on next page)

(continued from previous page)

```
|   ├── allowedNode.txt
|   └── runtimes_visualization.svg
|
└── model
    └── ssd_mobilenet_v2_300_float.tflite
```

DNN directory structure Each DNN must have the following 3 components:

1. **model**: This directory contains the DNN being targeted to infer
2. **artifacts**: This directory contains the artifacts generated after the compilation of DNN for SDK, and described in *DNN compilation for SDK – Basic Instructions*
3. **param.yaml**: A configuration file in yaml format to provide basic information about DNN, and associated pre and post processing parameters. More details can be find *Param file format*

Param file format Each DNN has its own pre-process, inference and post-process parameters to get the correct output. This information is typically available in the training software that was used to train the model. In order to convey this information to the SDK in a standardized fashion, we have defined a set of parameters that describe these operations. These parameters are in the param.yaml file.

Please see sample yaml files for various tasks such as image classification, semantic segmentation and object detection in edgeai-benchmark examples. Descriptions of various parameters are also in the yaml files. If users want to bring their own model to the SDK, then they need to prepare this information offline and get to the SDK. In next section we explain how to prepare this information

DNN compilation for SDK – Basic Instructions The BeagleBone® AI-64 Linux for Edge AI supports three different runtimes to infer a DNN, and user can choose a run time depending on the format of DNN. We recommend users to use different run times and compare the performance and select the one which provides best performance. User can find the steps to generate the artifacts directory at Edge AI TIDL Tools

DNN compilation for SDK – Advanced Instructions For beginners who are trying to compile models for the SDK, we recommend the basic instructions given in the previous section. However, DNNs have lot of variety and some models may need a different kind of preprocessing or postprocessing operations. In order to help customers deal with different kinds of models, we have prepared a model zoo in the repository edgeai-modelzoo

For the DNNs which are part of TI's model zoo, one can find the compilation settings and pre-compiled model artifacts in edgeai-benchmark repository. Instructions are also given to compile custom models. When using edgeai-benchmark for model compilation, the yaml file is automatically generated and artifacts are packaged in the way SDK understands. Please follow the instructions in the repository to get started.

Demo Configuration file

The demo config file uses YAML format to define input sources, models, outputs and finally the flows which defines how everything is connected. Config files for out-of-box demos are kept in `edge_ai_apps/configs` folder. The folder contains config files for all the use cases and also multi-input and multi-inference case. The folder also has a template YAML file `app_config_template.yaml` which has detailed explanation of all the parameters supported in the config file.

Config file is divided in 4 sections:

1. Inputs
2. Models
3. Outputs
4. Flows

Inputs The input section defines a list of supported inputs like camera, filesrc etc. Their properties like shown below.

```
inputs:
    input0:                                             #Camera Input
        source: /dev/video2                             #Device file entry of the␣
↪camera
        format: jpeg                                    #Input data format␣
↪supported by camera
        width: 1280                                     #Width and Height of the␣
↪input
        height: 720
        framerate: 30                                   #Framerate of the source

    input1:                                             #Video Input
        source: ../data/videos/video_0000_h264.mp4      #Video file
        format: h264                                    #File encoding format
        width: 1280
        height: 720
        framerate: 25

    input2:                                             #Image Input
        source: ../data/images/%04d.jpg                 #Sequence of Image files,␣
↪printf style formatting is used
        width: 1280
        height: 720
        index: 0                                        #Starting Index␣
↪(optional)
        framerate: 1
```

All supported inputs are listed in template config file. Below are the details of most commonly used inputs.

Camera sources (v4l2) **v4l2src** GStreamer element is used to capture frames from camera sources which are exposed as v4l2 devices. In Linux, there are many devices which are implemented as v4l2 devices. Not all of them will be camera devices. You need to make sure the correct device is configured for running the demo successfully.

`init_script.sh` is ran as part of systemd, which detects all cameras connected and prints the detail like below in the UART console:

```
debian@beaglebone:/opt/edge_ai_apps# ./init_script.sh
USB Camera detected
    device = /dev/video18
    format = jpeg
CSI Camera 0 detected
    device = /dev/video2
    name = imx219 8-0010
    format = [fmt:SRGGB8_1X8/1920x1080]
    subdev_id = 2
    isp_required = yes
IMX390 Camera 0 detected
    device = /dev/video18
    name = imx390 10-001a
    format = [fmt:SRGGB12_1X12/1936x1100 field: none]
    subdev_id = /dev/v4l-subdev7
    isp_required = yes
    ldc_required = yes
```

script can also be run manually later to get the camera details.

From the above log we can determine that 1 USB camera is connected (/dev/video18), and 1 CSI camera is connected (/dev/video2) which is imx219 raw sensor and needs ISP. IMX390 camera needs both ISP and LDC.

Using this method, you can configure correct device for camera capture in the input section of config file.

```
input0:
    source: /dev/video18  #USB Camera
    format: jpeg          #if connected USB camera supports jpeg
    width: 1280
    height: 720
    framerate: 30

input1:
    source: /dev/video2  #CSI Camera
    format: auto         #let the gstreamer negotiate the format
    width: 1280
    height: 720
    framerate: 30

input2:
    source: /dev/video2  #IMX219 raw sensor that needs ISP
    format: rggb         #ISP will be added in the pipeline
    width: 1920
    height: 1080
    framerate: 30
    subdev-id: 2         #needed by ISP to control sensor params via ioctls

input3:
    source: /dev/video2  #IMX390 raw sensor that needs ISP
    width: 1936
    height: 1100
    format: rggb12       #ISP will be added in the pipeline
    subdev-id: 2         #needed by ISP to control sensor params via ioctls
    framerate: 30
    sen-id: imx390
    ldc: True            #LDC will be added in the pipeline
```

Make sure to configure correct `format` for camera input. `jpeg` for USB camera that supports MJPEG (Ex. C270 logitech USB camera). `auto` for CSI camera to allow gstreamer to negotiate the format. `rggb` for sensor that needs ISP.

Video sources H.264 and H.265 encoded videos can be provided as input sources to the demos. Sample video files are provided under `/opt/edge_ai_apps/data/videos/video_0000_h264.mp4` and `/opt/edge_ai_apps/data/videos/video_000_h265.mp4`

```
input1:
    source: ../data/videos/video_0000_h264.mp4
    format: h264
    width: 1280
    height: 720
    framerate: 25

input2:
    source: ../data/videos/video_0000_h265.mp4
    format: h265
    width: 1280
    height: 720
    framerate: 25
```

Make sure to configure correct `format` for video input as shown above. By default the format is set to `auto` which will then use the GStreamer bin `decodebin` instead.

Image sources JPEG compressed images can be provided as inputs to the demos. A sample set of images are provided under `/opt/edge_ai_apps/data/images`. The names of the files are numbered se-

quentially and incrementally and the demo plays the files at the fps specified by the user.

```
input2:
    source: ../data/images/%04d.jpg
    width: 1280
    height: 720
    index: 0
    framerate: 1
```

RTSP sources H.264 encoded video streams either coming from a RTSP compliant IP camera or via RTSP server running on a remote PC can be provided as inputs to the demo.

```
input0:
    source: rtsp://172.24.145.220:8554/test # rtsp stream url, replace this␣
↪with correct url
    width: 1280
    height: 720
    framerate: 30
```

Note: Usually video streams from any IP camera will be encrypted and cannot be played back directly without a decryption key. We tested RTSP source by setting up an RTSP server on a Ubuntu 18.04 PC by referring to this writeup, Setting up RTSP server on PC

Models The model section defines a list of models that are used in the demo. Path to the model directory is a required argument for each model and rest are optional properties specific to given use cases like shown below.

```
models:
    model0:
        model_path: ../models/segmentation/ONR-SS-871-deeplabv3lite-mobv2-
↪cocoseg21-512x512   #Model Directory
        alpha: 0.4                                                          ␣
↪                  #alpha for blending segmentation mask (optional)
    model1:
        model_path: ../models/detection/TFL-OD-202-ssdLite-mobDet-DSP-coco-
↪320x320
        viz_threshold: 0.3                                                  ␣
↪                 #Visualization threshold for adding bounding boxes␣
↪(optional)
    model2:
        model_path: ../models/classification/TVM-CL-338-mobileNetV2-qat
        topN: 5                                                             ␣
↪                  #Number of top N classes (optional)
```

Below are some of the use case specific properties:

1. **alpha**: This determines the weight of the mask for blending the semantic segmentation output with the input image `alpha * mask + (1 - alpha) * image`
2. **viz_threshold**: Score threshold to draw the bounding boxes for detected objects in object detection. This can be used to control the number of boxes in the output, increase if there are too many and decrease if there are very few
3. **topN**: Number of most probable classes to overlay on image classification output

The content of the model directory and its structure is discussed in detail in *Import Custom Models*

Outputs The output section defines a list of supported outputs.

```
outputs:
    output0:                                                    #Display␣
↪Output
        sink: kmssink
        width: 1920                                             #Width and␣
↪Height of the output
        height: 1080
        connector: 39                                           #Connector␣
↪ID for kmssink (optional)

    output1:                                                    #Video Output
        sink: ../data/output/videos/output_video.mkv            #Output␣
↪video file
        width: 1920
        height: 1080

    output2:                                                    #Image Output
        sink: ../data/output/images/output_image_%04d.jpg       #Image file␣
↪name, printf style formatting is used
        width: 1920
        height: 1080
```

All supported outputs are listed in template config file. Below are the details of most commonly used outputs

Display Sink (kmssink) When you have only one display connected to the SK, kmssink will try to use it for displaying the output buffers. In case you have connected multiple display monitors (e.g. Display Port and HDMI), you can select a specific display for kmssink by passing a specific connector ID number. Following command finds out the connected displays available to use.

Note: Run this command outside docker container. The first number in each line is the connector-id which we will use in next step.

```
debian@beaglebone:/opt/edge_ai_apps# modetest -M tidss -c | grep connected
39      38      connected       DP-1            530x300         12      38
48      0       disconnected    HDMI-A-1        0x0             0       47
```

From above output, we can see that connector ID 39 is connected. Configure the connector ID in the output section of the config file.

Video sinks The post-processed outputs can be encoded in H.264 format and stored on disk. Please specify the location of the video file in the configuration file.

```
output1:
    sink: ../data/output/videos/output_video.mkv
    width: 1920
    height: 1080
```

Image sinks The post-processed outputs can be stored as JPEG compressed images. Please specify the location of the image files in the configuration file. The images will be named sequentially and incrementally as shown.

```
output2:
    sink: ../data/output/images/output_image_%04d.jpg
    width: 1920
    height: 1080
```

Flows The flows section defines how inputs, models and outputs are connected. Multiple flows can be defined to achieve multi input, multi inference like below.

```
flows:
    flow0:                                  #First Flow
        input: input0                       #Input for the Flow
        models: [model1, model2]            #List of models to be used
        outputs: [output0, output0]         #Outputs to be used for each model␣
↪inference output
        mosaic:                             #Positions to place the inference␣
↪outputs in the output frame
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  90
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  90
    flow1:                                  #Second Flow
        input: input1
        models: [model0, model3]
        outputs: [output0, output0]
        mosaic:
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  540
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  540
```

Each flow should have exactly **1 input**, **n models** to infer the given input and **n outputs** to render the output of each inference. Along with input, models and outputs it is required to define **n mosaics** which are the position of the inference output in the final output plane. This is needed because multiple inference outputs can be rendered to same output (Ex: Display).

Command line arguments Limited set of command line arguments can be provided, run with '-h' or '–help' option to list the supported parameters.

```
usage: Run : ./app_edgeai.py -h for help

positional arguments:
  config           Path to demo config file
                       ex: ./app_edgeai.py ../configs/app_config.yaml

optional arguments:
  -h, --help       show this help message and exit
  -n, --no-curses  Disable curses report
                   default: Disabled
  -v, --verbose    Verbose option to print profile info on stdout
                   default: Disabled
```

Running Advance demos

The same Python and C++ demo applications can be used to run multiple inference models and also work with multiple inputs with just simple changes in the config file.

From a repo of input sources, output sources and models one can define advance dataflows which connect them in various configurations. Details on configuration file parameters can be found in *Demo Configuration file*

Single input multi inference demo Here is an example of a single-input, multi-inference demo which takes a camera input and run multiple networks on each of them.

```
debian@beaglebone:/opt/edge_ai_apps/apps_python# ./app_edgeai.py ../configs/
↪single_input_multi_infer.yaml
```

Sample output for single input, multi inference demo is as shown below,



Fig. 3.33: Sample output showing single input, mutli-inference output

We can specify the output window location and sizes as shown in the configuration file,

```
flows:
    flow0:
        input: input0
        models: [model0, model1, model2, model3]
        outputs: [output0, output0, output0, output0]
        mosaic:
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  90
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  90
            mosaic2:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  540
```

(continues on next page)

(continued from previous page)

```
            mosaic3:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  540
```

Multi input multi inference demo Here is an example of a multi-input, multi-inference demo which takes a camera input and video input and runs multiple networks on each of them.

```
debian@beaglebone:/opt/edge_ai_apps/apps_python# ./app_edgeai.py ../configs/
↪multi_input_multi_infer.yaml
```

Sample output for multi input, multi inference demo is as shown below,



Fig. 3.34: Sample output showing multi-input, mutli-inference output

We can specify the output window location and sizes as shown in the configuration file,

```
flows:
    flow0:
        input: input0
        models: [model1, model2]
        outputs: [output0, output0]
        mosaic:
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  90
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  90
    flow1:
        input: input1
```

(continues on next page)

(continued from previous page)

```
        models: [model0, model3]
        outputs: [output0, output0]
        mosaic:
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  540
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  540
```

Docker Environment

Docker is a set of "platform as a service" products that uses the OS-level virtualization to deliver software in packages called containers. Docker container provides a quick start environment to the developer to run the out of box demos and build applications.

The Docker image is based on Ubuntu 20.04.LTS and contains different open source components like OpenCV, GStreamer, Python and pip packages which are required to run the demos. The user can choose to install any additional 3rd party applications and packages as required.

Building Docker image The *docker/Dockerfile* in the edge_ai_apps repo describes the recipe for creating the Docker container image. Feel free to review and update it to include additional packages before building the image.

Note: Building Docker image on target using the provided Dockerfile will take about 15-20 minutes to complete with good internet connection. Building Docker containers on target can be slow and resource constrained. The Dockerfile provided will build on target without any issues but if you add more packages or build components from source, running out of memory can be a common problem. As an alternative we highly recommend trying QEMU builds for cross-compiling the images for arm64 architecture on a PC and then load the compiled image on target.

Initiate the Docker image build as shown,

```
debian@beaglebone:/opt/edge_ai_apps/docker#./docker_build.sh
```

Running the Docker container Enter the Docker session as shown,

```
debian@beaglebone:/opt/edge_ai_apps/docker#./docker_run.sh
```

This will start a Ubuntu 20.04.LTS image based Docker container and the prompt will change as below,

```
[docker] debian@beaglebone:/opt/edge_ai_apps#
```

The Docker container has been created in privilege mode, so that it has root capabilities to all devices on the target system like Network etc. The container file system also mounts the target file system of /dev, /opt to access camera, display and other hardware accelerators the SoC has to offer.

Note: It is highly recommended to use the docker_run.sh script to launch the Docker container because this script will take care of saving any changes made to the filesystem. This will make sure that any modifications to the Docker filesystem including new package installation, updates to some files and also command history is saved automatically and is available the next time you launch the container. The container will be committed

only if you exit from the container explicitly. If you restart the board without exiting container, any changes done from last saved state will be lost.

Note: After building and running the docker container, one needs to run `setup_script.sh` before running any of the demo applications. Please refer to *Software setup* for more details.

Handling proxy settings If the board running the Docker container is behind a proxy server, the default settings for downloading files and installing packages via apt-get will not work. If you are running the board from TI network, docker build and run scripts will automatically detect and configure necessary proxy settings

For other cases, you need to modify the script `/usr/bin/setup_proxy.sh` to add the custom proxy settings required for your network.

Additional Docker commands

Note: This section is provided only for additional reference and not required to run out-of-box demos

Commit Docker container

Generally, containers have a short life cycle. If the container has any local changes it is good to save the changes on top of the existing Docker image. When re-running the Docker image, the local changes can be restored.

Following commands show how to save the changes made to the last container. Note that this is already done automatically by `docker_run.sh` when you exit the container.

```
cont_id=`docker ps -q -l`
docker commit $cont_id edge_ai_kit
docker container rm $cont_id
```

For more information refer: Commit Docker image

Save Docker Image

Docker image can be saved as tar file by using the command below:

```
docker save --output <pre_built_docker_image.tar>
```

For more information refer here. Save Docker image

Load Docker image

Load a previously saved Docker image using the command below:

```
docker load --input <pre_built_docker_image.tar>
```

For more information refer here. Load Docker image

Remove Docker image

Docker image can be removed by using the command below:

```
Remove selected image:
docker rmi <image_name/ID>

Remove all image:
docker image prune -a
```

For more information refer rmi reference and Image prune reference

Remove Docker container

Docker container can be removed by using the command below:

```
Remove selected container:
docker rm <container_ID>

Remove all container:
docker container prune
```

For more information refer here. rm reference and Container Prune reference

Relocating Docker Root Location The default location for Docker files is **/var/lib/docker**. Any Docker images created will be stored here. This will be a problem anytime the SD card is updated with a new targetfs. If a secondary storage (SSD or USB based storage) is available, then it is recommended to relocate the default Docker root location so as to preserve any existing Docker images. Once the relocation has been done, the Docker content will not be affected by any future targetfs updates or accidental corruptions of the SD card.

The following steps outline the process for Docker root directory relocation assuming that the current Docker root is not at the desired location. If the current location is the desired location then exit this procedure.

1. Run 'Docker info' command inspect the output. Locate the line with content **Docker Root Dir**. It will list the current location.
2. To preserve any existing images, export them to .tar files for importing later into the new location.
3. Inspect the content under /etc/docker to see if there is a file by name **daemon.json**. If the file is not present then create **/etc/docker/docker.json** and add the following content. Update the 'key:value' pair for the key "graph" to reflect the desired root location. If the file already exists, then make sure that the line with "graph" exists in the file and points to the desired target location.

```
{
  "graph": "/run/media/nvme0n1/docker_root",
  "storage-driver": "overlay",
  "live-restore": true
}
```

In the configuration above, the key/value pair **'"graph": "/run/media/nvme0n1/docker_root"'** defines the root location **'/run/media/nvme0n1/docker_root'.**

4. Once the daemon.json file has been copied and updated, run the following commands

```
$ systemctl restart docker
$ docker info
```

Make sure that the new Docker root appears under **Docker Root Dir** value.

5. If you exported the existing images in step (2) then import them and they will appear under the new Docker root.
6. Anytime the SD card is updated with a new targetfs, steps (1), (3), and (4) need to be followed.

Additional references

https://docs.docker.com/engine/reference/commandline/images/
https://docs.docker.com/engine/reference/commandline/ps/

Data Flows

The **app_edgeai** application at a high level can be split into 3 parts,

- Input pipeline - Grabs a frame from camera, video, image or RTSP source
- Output pipeline - Sends the output to display or a file

- Compute pipeline - Performs pre-processing, inference and post-processing

Here are the data flows for each reference demo and the corresponding GStreamer launch strings that **app_edgeai** application generates. User can interact with the application via the *Demo Configuration file*

Image classification In this demo, a frame is grabbed from an input source and split into two paths. The "analytics" path resizes the input maintaining the aspect ratio and crops the input to match the resolution required to run the deep learning network. The "visualization" path is provided to the post-processing module which overlays the detected classes. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=0 mean-0=123.
↪675000 mean-1=116.280000 mean-2=103.530000 scale-0=0.017125 scale-1=0.
↪017507 scale-2=0.017429 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

Object Detection In this demo, a frame is grabbed from an input source and split into two paths. The "analytics" path resizes the input to match the resolution required to run the deep learning network. The "visualization" path is provided to the post-processing module which overlays rectangles around detected objects. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

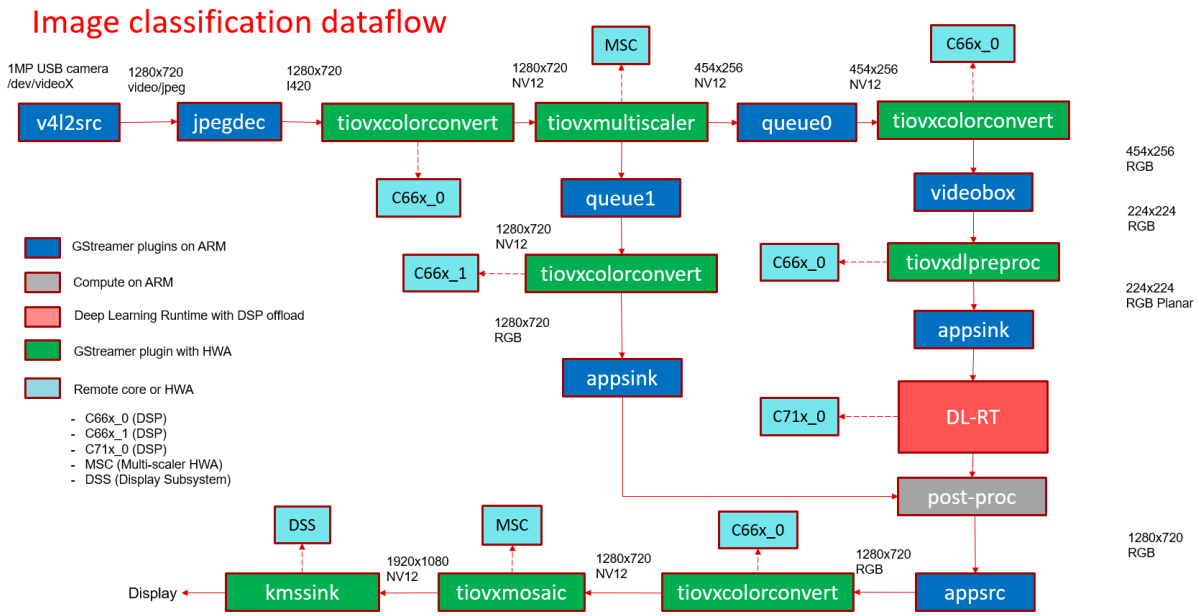


Fig. 3.35: GStreamer based data-flow pipeline for image classification demo with USB camera and display

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

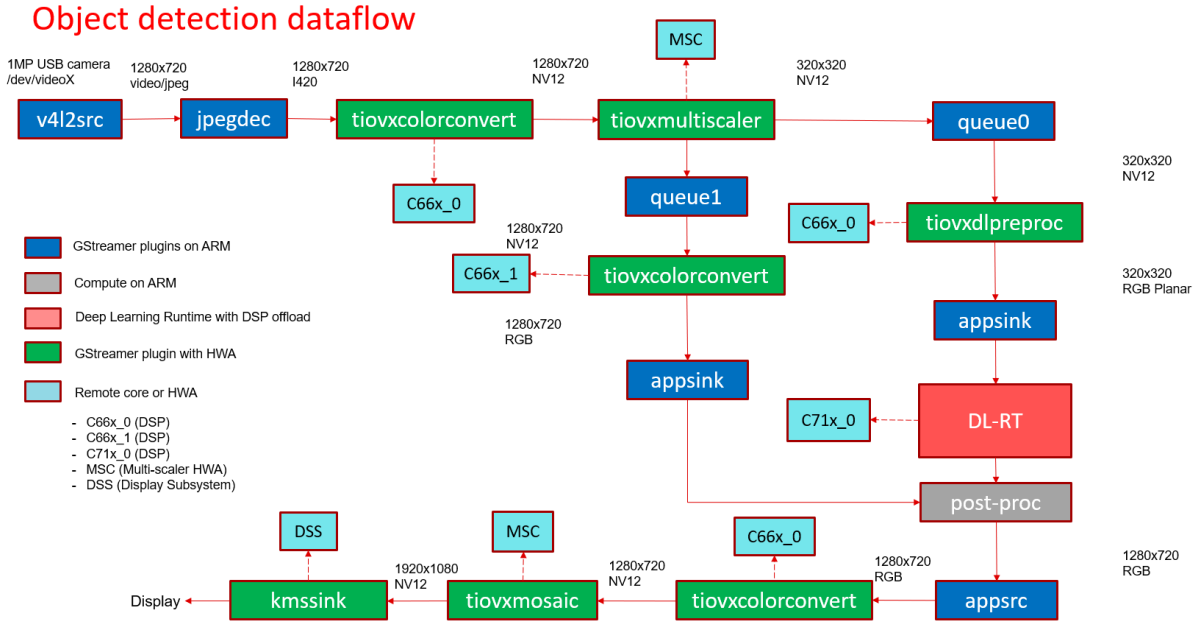


Fig. 3.36: GStreamer based data-flow pipeline for object detection demo with USB camera and display

Semantic Segmentation In this demo, a frame is grabbed from an input source and split into two paths. The "analytics" path resize the input to match the resolution required to run the deep learning network. The "visualization" path is provided to the post-processing module which blends each segmented pixel to a color map. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=512, height=512 ! tiovxdlpreproc data-
↪type=10 channel-order=0 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.015625 scale-1=0.015625 scale-2=0.015625 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

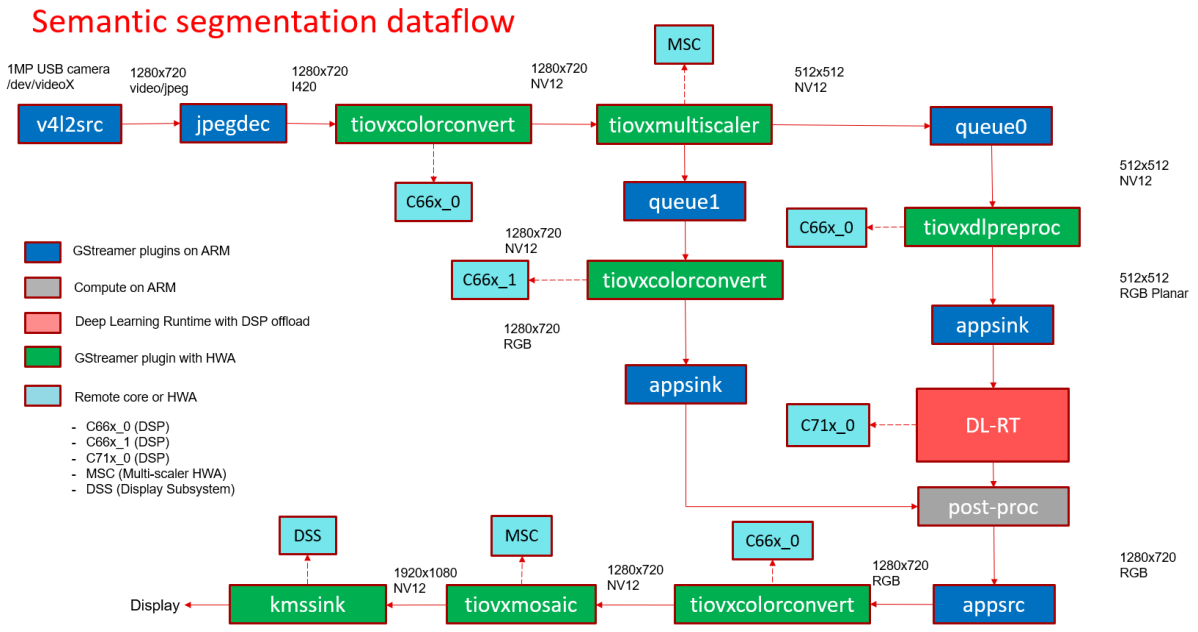


Fig. 3.37: GStreamer based data-flow pipeline for semantic segmentation demo with USB camera and display

Human Pose Estimation In this demo, a frame is grabbed from an input source and split into two paths. The "analytics" path resize the input to match the resolution required to run the deep learning network. The "visualization" path is provided to the post-processing module which overlays the keypoints and lines to draw

the pose. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video2 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=640, height=640 ! tiovxdlpreproc data-
↪type=10 target=0 channel-order=0 mean-0=0.000000 mean-1=0.000000 mean-2=0.
↪000000 scale-0=1.000000 scale-1=1.000000 scale-2=1.000000 tensor-
↪format=bgr out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
GStreamer output pipeline:
```

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
tiovxmosaic name=mosaic_0 background=/tmp/background_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

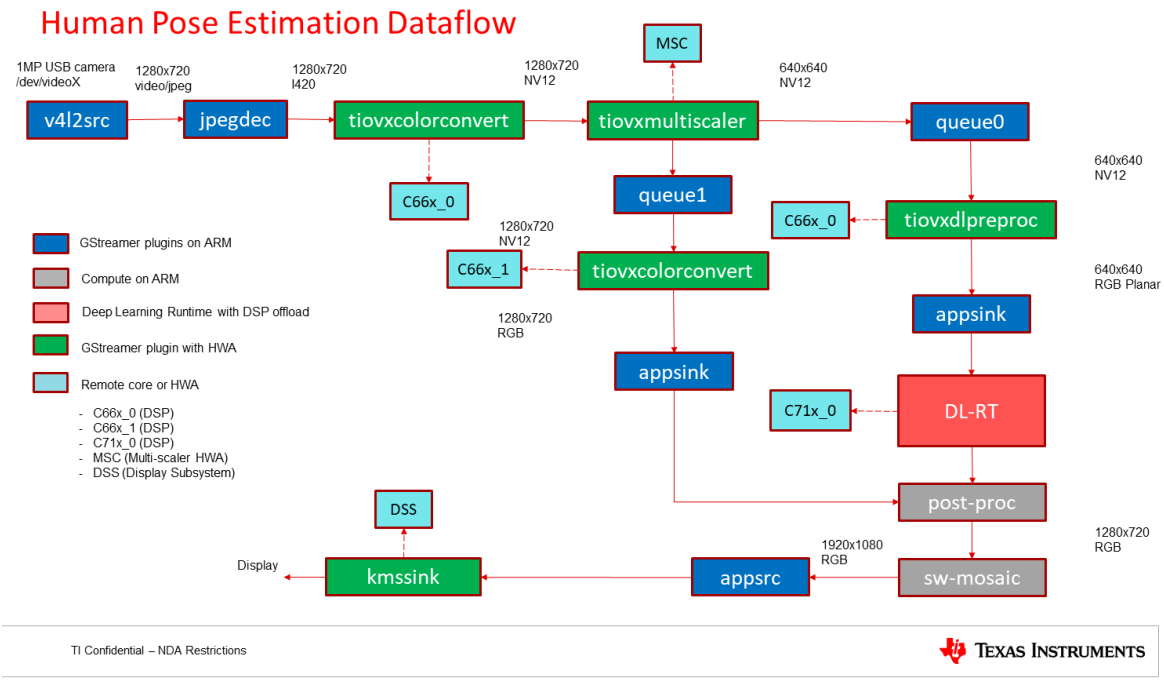


Fig. 3.38: GStreamer based data-flow pipeline for Human Pose Estimation demo with USB camera and display

Video source In this demo, a video file is read from a known location and passed to a de-muxer to extract audio and video streams, the video stream is parsed and raw encoded information is passed to a HW video decoder. Note that H.264 and H.265 encoded videos are supported, making use of the respective HW decoders. The resulting output is split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
filesrc location=/opt/edge_ai_apps/data/videos/video_0000_h264.mp4 ! qtdemux␣
↪! h264parse ! v4l2h264dec ! video/x-raw, format=NV12  ! tiovxmultiscaler␣
↪name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

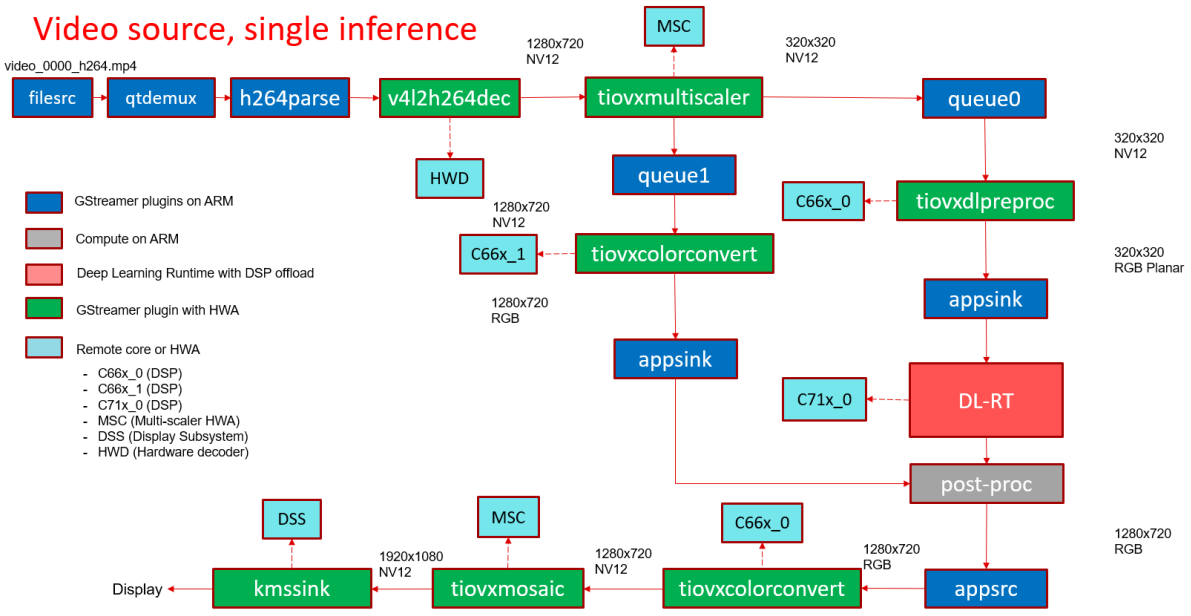


Fig. 3.39: GStreamer based data-flow pipeline with video file input source and display

RTSP source In this demo, a video file is read from a RTSP source and passed to a de-muxer to extract audio and video streams, the video stream is parsed and raw encoded information is passed to a video decoder and the resulting output is split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
rtspsrc location=rtsp://172.24.145.220:8554/test latency=0 buffer-mode=auto !
↪ rtph264depay ! h264parse ! v4l2h264dec ! video/x-raw, format=NV12 !
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

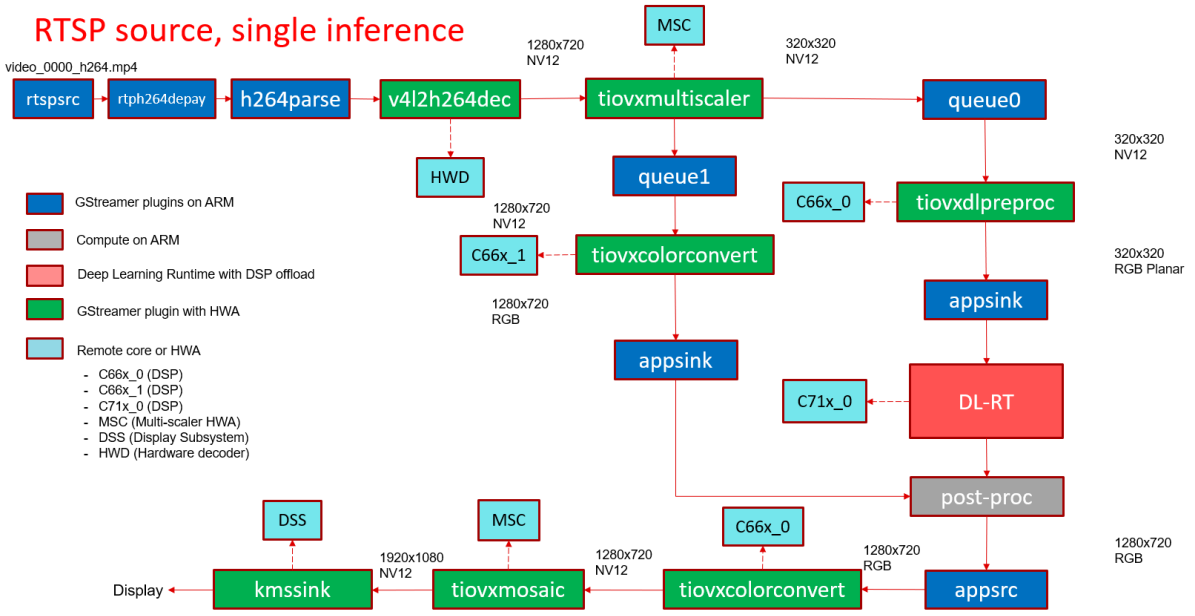


Fig. 3.40: GStreamer based data-flow pipeline with RTSP based video file source and display

RPiV2 Camera Sensor (IMX219) In this demo, raw frames in SRGGB8 format are captured form RPiV2 (imx219) camera sensor. VISS (Vision Imaging Subsystem) is used to process the raw frames and get the output in NV12, VISS also cotrols the sensor parameters like exposure, gain etc.. via v4l2 ioctls. The NV12 output is split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video2 io-mode=5 ! video/x-bayer, width=1920,␣
↪height=1080, format=rggb ! tiovxisp device=/dev/v4l-subdev2 dcc-isp-file=/
↪opt/imaging/imx219/dcc_viss.bin dcc-2a-file=/opt/imaging/imx219/dcc_2a.bin␣
↪format-msb=7 ! video/x-raw, format=NV12 ! tiovxmultiscaler ! video/x-raw,␣
↪width=1280, height=720 ! tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

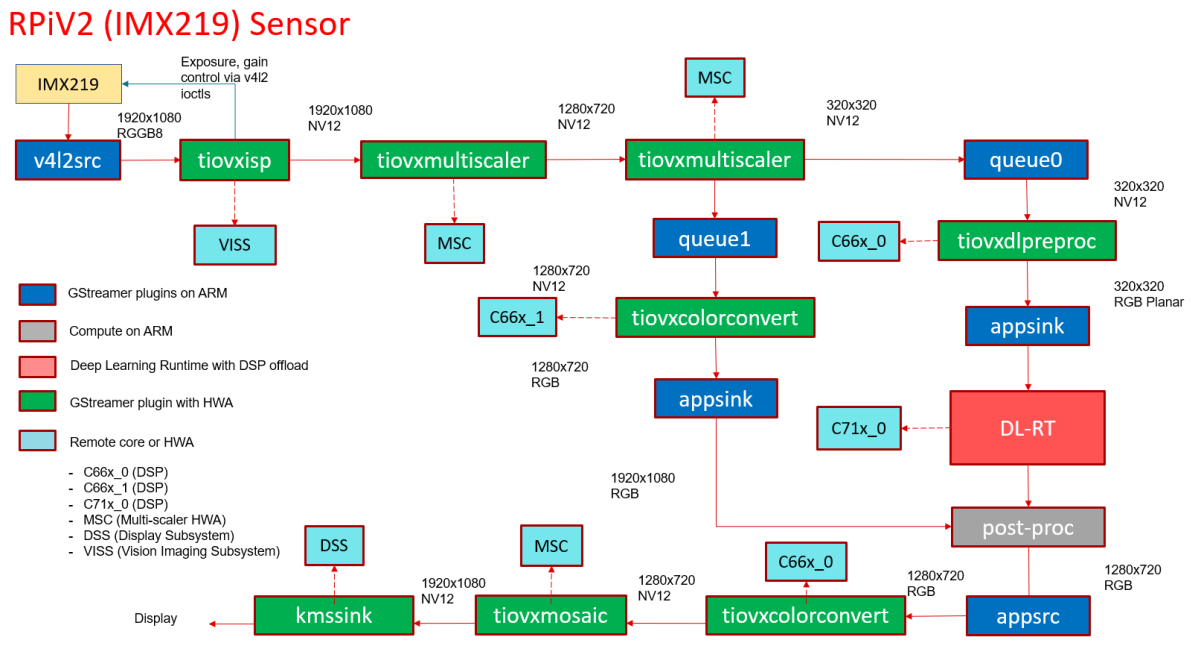


Fig. 3.41: GStreamer based data-flow pipeline with IMX219 sensor, ISP and display

IMX390 Camera Sensor In this demo, raw frames in SRGGB12 format are captured from IMX390 camera sensor. VISS (Vision Imaging Subsystem) is used to process the raw frames and get the output in NV12, VISS also controls the sensor parameters like exposure, gain etc.. via v4l2 ioctls. This is followed by LDC (Lens Distortion Correction) required due to the fisheye lens. The NV12 output is split into two paths. The "analytics" path resizes the input to match the resolution required to run the deep learning network. The "visualization" path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty

background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 ! queue leaky=2 ! video/x-bayer, width=1936,␣
↪height=1100, format=rggb12 ! tiovxisp sink_0::device=/dev/v4l-subdev7␣
↪sensor-name=IMX390-UB953_D3 dcc-isp-file=/opt/imaging/imx390/dcc_viss.bin␣
↪sink_0::dcc-2a-file=/opt/imaging/imx390/dcc_2a.bin format-msb=11 ! video/x-
↪raw, format=NV12 ! tiovxldc dcc-file=/opt/imaging/imx390/dcc_ldc.bin␣
↪sensor-name=IMX390-UB953_D3 ! video/x-raw, format=NV12, width=1920,␣
↪height=1080 !tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=512, height=512 ! tiovxdlpreproc data-
↪type=3 target=0 channel-order=0 tensor-format=bgr out-pool-size=4 !␣
↪application/x-tensor-tiovx ! appsink name=pre_0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
tiovxmosaic name=mosaic_0 background=/tmp/background_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

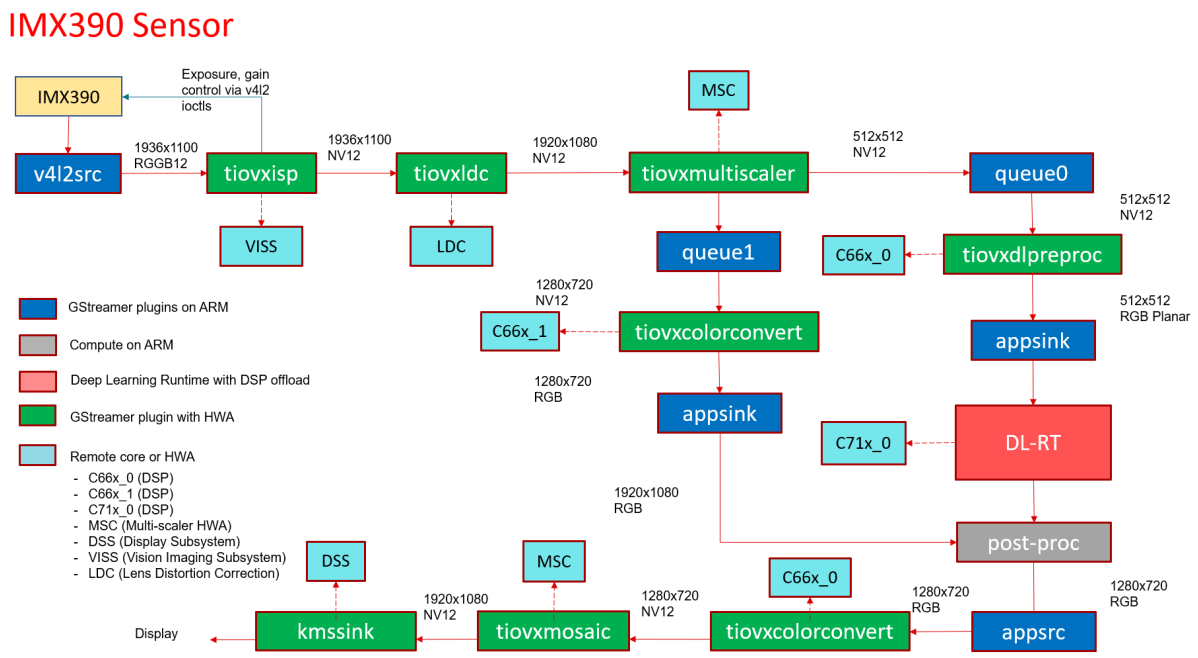


Fig. 3.42: GStreamer based data-flow pipeline with IMX390 sensor, ISP, LDC and display

Video output In this demo, a frame is grabbed from an input source and split into two paths. The "analytics" path resizes the input to match the resolution required to run the deep learning network. The "visualization" path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background. Finally the video is encoded using the H.264 HW encoder and written to a video file.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! v4l2h264enc␣
↪bitrate=10000000 ! h264parse ! matroskamux !  filesink location=/opt/edge_
↪ai_apps/data/output/videos/output_video.mkv
```

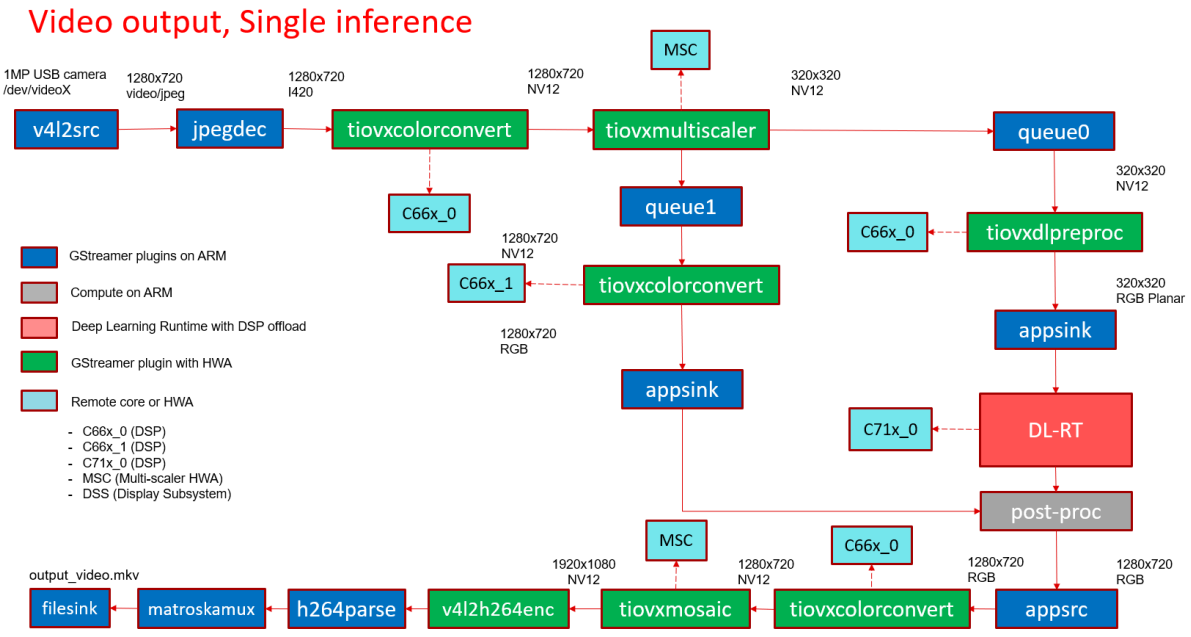


Fig. 3.43: GStreamer based data-flow pipeline with video file input source and display

Single Input Multi inference In this demo, a frame is grabbed from an input source and split into multiple paths. Each path is further split into two sub-paths one for analytics and another for visualization. Each path can run any type of network, image classification, object detection, semantic segmentation and using any supported run-time.

For example the below GStreamer pipeline splits the input into 4 paths for running 4 deep learning networks. First is a semantic segmentation network, followed by object detection network, followed by two image classification networks. If we look at the image classification path, the analytics sub-path resizes the input to maintain the aspect ratio and crops the input to match the resolution required to run the deep learning network. The

visualization sub-path is provided to the post-processing module which overlays the detected classes. Post-processed output from all the 4 paths is given to HW mosaic plugin which positions and resizes the output windows on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 ! tee name=tee_
↪split0
tee_split0. ! queue ! tiovxmultiscaler name=split_01
tee_split0. ! queue ! tiovxmultiscaler name=split_02
tee_split0. ! queue ! tiovxmultiscaler name=split_03
tee_split0. ! queue ! tiovxmultiscaler name=split_04
split_01. ! queue ! video/x-raw, width=512, height=512 ! tiovxdlpreproc data-
↪type=10 channel-order=0 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.015625 scale-1=0.015625 scale-2=0.015625 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_0␣
↪max-buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪1 max-buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_1␣
↪max-buffers=2 drop=true
split_03. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=1 mean-0=128.
↪000000 mean-1=128.000000 mean-2=128.000000 scale-0=0.007812 scale-1=0.
↪007812 scale-2=0.007812 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_2 max-buffers=2 drop=true
split_03. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_2␣
↪max-buffers=2 drop=true
split_04. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=0 mean-0=123.
↪675000 mean-1=116.280000 mean-2=103.530000 scale-0=0.017125 scale-1=0.
↪017507 scale-2=0.017429 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_3 max-buffers=2 drop=true
split_04. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_3␣
↪max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_1 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_1
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_2 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_2
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_3 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_3
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
```

(continues on next page)

(continued from previous page)

```
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=640   sink_
↪0::height=360
sink_1::startx=960  sink_1::starty=180  sink_1::width=640   sink_
↪1::height=360
sink_2::startx=320  sink_2::starty=560  sink_2::width=640   sink_
↪2::height=360
sink_3::startx=960  sink_3::starty=560  sink_3::width=640   sink_
↪3::height=360
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

Multi Input Multi inference In this demo, a frame is grabbed from multiple input sources and split into multiple paths. The multiple input sources could be either multiple cameras or a combination of camera, video, image, RTSP source. Each path is further split into two sub-paths one for analytics and another for visualization. Each path can run any type of network, image classification, object detection, semantic segmentation and using any supported run-time.

For example the below GStreamer pipeline splits two inputs into 4 paths for running 2 deep learning networks. First is a object detection network, followed by image classification networks. If we look at the image classification path, the analytics sub-path resizes the input to maintain the aspect ratio and crops the input to match the resolution required to run the deep learning network. The visualization sub-path is provided to the post-processing module which overlays the detected classes. Post-processed output from all the 4 paths is given to HW mosaic plugin which positions and resizes the output windows on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 ! tee name=tee_
↪split0
tee_split0. ! queue ! tiovxmultiscaler name=split_01
tee_split0. ! queue ! tiovxmultiscaler name=split_02
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_0␣
↪max-buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=1 mean-0=128.
↪000000 mean-1=128.000000 mean-2=128.000000 scale-0=0.007812 scale-1=0.
↪007812 scale-2=0.007812 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_1 max-buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_1␣
↪max-buffers=2 drop=true

filesrc location=/opt/edge_ai_apps/data/videos/video_0000_h264.mp4 ! qtdemux␣
↪! h264parse ! v4l2h264dec ! video/x-raw, format=NV12  ! tee name=tee_split1
tee_split1. ! queue ! tiovxmultiscaler name=split_11
tee_split1. ! queue ! tiovxmultiscaler name=split_12
split_11. ! queue ! video/x-raw, width=512, height=512 ! tiovxdlpreproc data-
↪type=10 channel-order=0 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.015625 scale-1=0.015625 scale-2=0.015625 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪2 max-buffers=2 drop=true
```

(continues on next page)

(continued from previous page)

```
split_11. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_2␣
↪max-buffers=2 drop=true
split_12. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=0 mean-0=123.
↪675000 mean-1=116.280000 mean-2=103.530000 scale-0=0.017125 scale-1=0.
↪017507 scale-2=0.017429 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_3 max-buffers=2 drop=true
split_12. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_3␣
↪max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_1 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_1
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_2 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_2
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_3 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_3
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=640   sink_
↪0::height=360
sink_1::startx=960  sink_1::starty=180  sink_1::width=640   sink_
↪1::height=360
sink_2::startx=320  sink_2::starty=560  sink_2::width=640   sink_
↪2::height=360
sink_3::startx=960  sink_3::starty=560  sink_3::width=640   sink_
↪3::height=360
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

Performance Visualization Tool

The performance visualization tool can be used to view all the performance statistics recorded when running the edge AI C++ demo application. This includes the CPU and HWA loading, DDR bandwidth, Junction Temperatures and FPS obtained. Refer to *Available options* for details on the performance metrics available to be plotted.

This tool works as follows:

- Logging: When running the application, the performance statistics can be recorded and stored in log files. This is done automatically when running the C++ application, but the Python application does not generate logs. However a standalone binary executable is provided that can be run in parallel with the Python application, which will generate these performance logs.
- Visualization: There is a Python script which parses these logs and plots graphs, which can be easily viewed by a visiting a URL in any browser. This script uses Streamlit package to update the graphs in real-time, as the Edge AI application runs in parallel. However, since Streamlit is not supported in the SDK out of box, this script needs to run on docker. Please refer to *Docker Environment* for building and running a docker container.

Generating Performance Logs

Each log file contains real-time values for some performance metrics, averaged over a 2s window. The temperature sensor values are sampled in real time, every 2s. The performance visualization tool then parses these log files one by one based on the modification timestamps.

The edge AI C++ demo will automatically generate log files and store them in the directory `../perf_logs`, that is, one level up from where the C++ app is run. For example, if the app is run from `edge_ai_apps/apps_cpp`, the logs will be stored in `edge_ai_apps/perf_logs`.

Similarly, there is a binary executable that can be compiled that does the same logging standalone. The source for this is available under `edge_ai_apps/scripts/perf_stats/`. The README.md file has simple instructions to build and run this standalone logger binary. After building it, use following command to print the statistics on the terminal as well as save them in log files that can be parsed.

```
debian@beaglebone:/opt/edge_ai_apps/scripts/perf_stats/build# ../bin/Release/
↪ti_perfstats -l
```

Running the Visualization tool

To use this tool, simply start a docker session and then run the command given below. This script expects some log files to be present in the directory `edge_ai_apps/perf_logs` after running any C++ demo. One can also bring up this tool while running the demo but it might affect the performance of the demo itself as it consumes a bit of ARM cycles during launch but stabilizes over a certain duration.

```
[docker] debian@beaglebone:/opt/edge_ai_apps# streamlit run scripts/perf_vis.
↪py --theme.base="light"
```

This script also accepts the log directory as a command line argument as follows:

```
[docker] debian@beaglebone:/opt/edge_ai_apps# streamlit run scripts/perf_vis.
↪py --theme.base="light" -- -D <path/to/logs/directory/>
```

A network URL can be seen in the terminal output. The graphs can be viewed by visiting this URL in any browser. The plotted graphs will keep updating based on the available log files.



Fig. 3.44: Performance visualizer dashboard showing CPU and HWA loading, DDR bandwidth, Junction Temperatures and the FPS obtained

To exit press Ctrl+C in the terminal.

Available options Average frames per second (FPS) recorded by the application is displayed by default. Using the checkboxes in the sidebar, one can select which performance metrics to view. There are 14 metrics available to be plotted, as seen from the above image:

- CPU Load: Total loading for the A72(mpu1_0), R5F(mcu2_0/1), C66x(c6x_1/2) and C71x(c7x_1) DSPs.
- HWA Load: Loading (percentage) for the various available hardware accelerators.
- DDR Bandwidth: Average read, write and total bandwidth recorded in the previous 2s interval.
- Junction Temperatures: The live temperatures recorded at various junctions
- Task Table: A separate graph for each cpu showing the loading due to various tasks running on it.
- Heap Table: A separate graph for each cpu showing the heap memory usage statistics.

For the first three metrics, there is a choice to view line graphs with a 30s history or bar graphs with only the real-time values. The remaining eleven have real-time bar graphs as the only option.

SDK Components

The BeagleBone® AI-64 Linux for Edge AI can be divided into 3 parts, Applications, BeagleBone® AI-64 Linux and Processor SDK RTOS. Users can get the latest application updates and bug fixes from the public repositories (GitHub and git.ti.com) which aligns with the SDK releases done quarterly. One can also build every component from source by following the steps in the TI Edge AI SDK development flow.

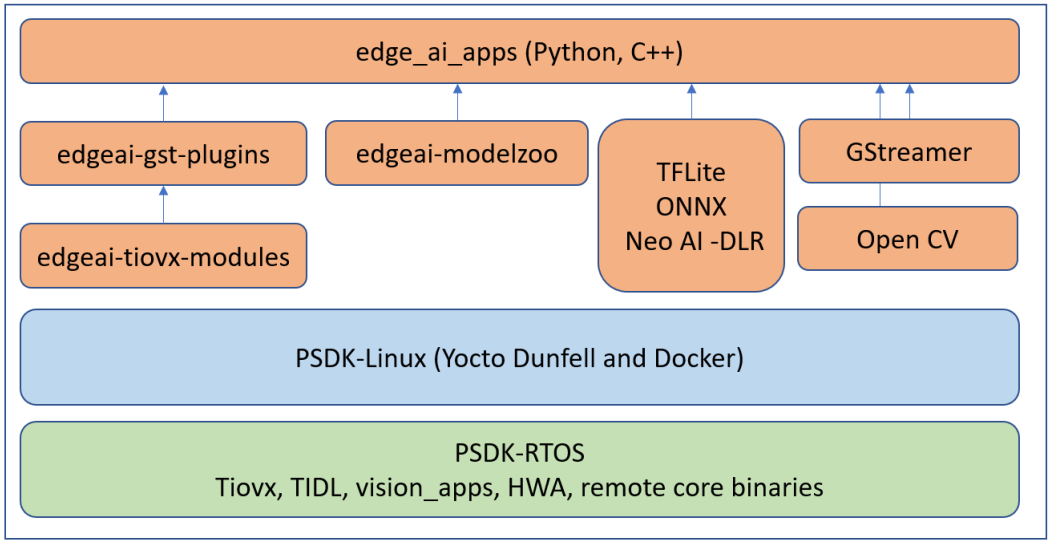


Fig. 3.45: BeagleBone® AI-64 Linux for Edge AI components

Edge AI Applications The edge AI applications are designed for users to quickly evaluate various Deep Learning networks on TDA4 SoC. The user can run standalone examples and Jupyter notebook applications to evaluate inference models either from TI Edge AI Model Zoo or a custom network. Once a network is finalized for performance and accuracy it can also be easily integrated in a typical capture-inference-display usecase using example GStreamer based applications for rapid prototyping and deployment.

edgeai-tidl-tools This application repository provides standalone Python and C/C++ examples to quickly evaluate inference models using TFLite, ONNX and NeoAI-DLR runtime using file based inputs. It also houses the Jupyter notebooks similar to TI Edge AI Cloud which can be executed right on the TDA4VM Starter Kit.

For more details on using this application repo please refer to the documentation and source code found here: https://github.com/TexasInstruments/edgeai-tidl-tools

edgeai-modelzoo This repo provides collection of example Deep Neural Network (DNN) Models for various computer vision tasks. A few example models are packaged as part of the SDK to run out-of-box demos. More can be downloaded using a download script made available in the edge_ai_apps repo.

For more details on the pre-imported models and related documentation please visit: https://github.com/TexasInstruments/edgeai-modelzoo

edge_ai_apps These are plug-and-play Deep Learning applications which support running open source runtime frameworks such as TFLite, ONNX and NeoAI-DLR with a live camera and display. They help connect realtime camera, video or RTSP sources to DL inference to live display, bitstream or RTSP sinks.

The latest source code with fixes can be pulled from: https://git.ti.com/cgit/edgeai/edge_ai_apps

edgeai-gst-plugins This repo provides the source of custom GStreamer plugins which helps offload tasks to TDA4 hardware accelerators and advanced DSPs with the help of edgeai-tiovx-modules. The repo gets downloaded, built and installed as part of the *Software setup* step.

Source code and documentation: https://github.com/TexasInstruments/edgeai-gst-plugins

edgeai-tiovx-modules This repo provides OpenVx modules which help access underlying hardware accelerators in the TDA4 SoC and serves as a bridge between GStreamer custom elements and underlying OpenVx custom kernels. The repo gets downloaded, built and installed as part of the *Software setup* step.

Source code and documentation: https://github.com/TexasInstruments/edgeai-tiovx-modules

Processor SDK RTOS The BeagleBone® AI-64 Linux for Edge AI gets all the HWA drivers, optimized libraries, OpenVx framework and more from Processor SDK RTOS

For more information visit Processor SDK RTOS Getting Started Guide.

BeagleBone® AI-64 Linux The BeagleBone® AI-64 Linux for Edge AI gets all the Linux kernel, filesystem, device-drivers and more from BeagleBone® AI-64 Linux

For more information visit BeagleBone® AI-64 Linux Software Developer's Guide.

Datasheet

This chapter describes the performance measurements of the Edge AI Inference demos.

Performance data of the demos can be auto generated by running following command on target:

```
debian@beaglebone:/opt/edge_ai_apps/tests# ./gen_data_sheet.sh
```

The performance measurements includes the following

1. **FPS** : Effective framerate at which the application runs
2. **Total time** : Average time taken to process each frame, which includes pre-processing, inference and post-processing time
3. **Inference time** : Average time taken to infer each frame
4. **CPU loading** : Loading on different CPU cores present
5. **DDR BW** : DDR read and write BW used
6. **HWA Loading** : Loading on different Hardware accelerators present

Following are the latest performance numbers of the C++ demos:

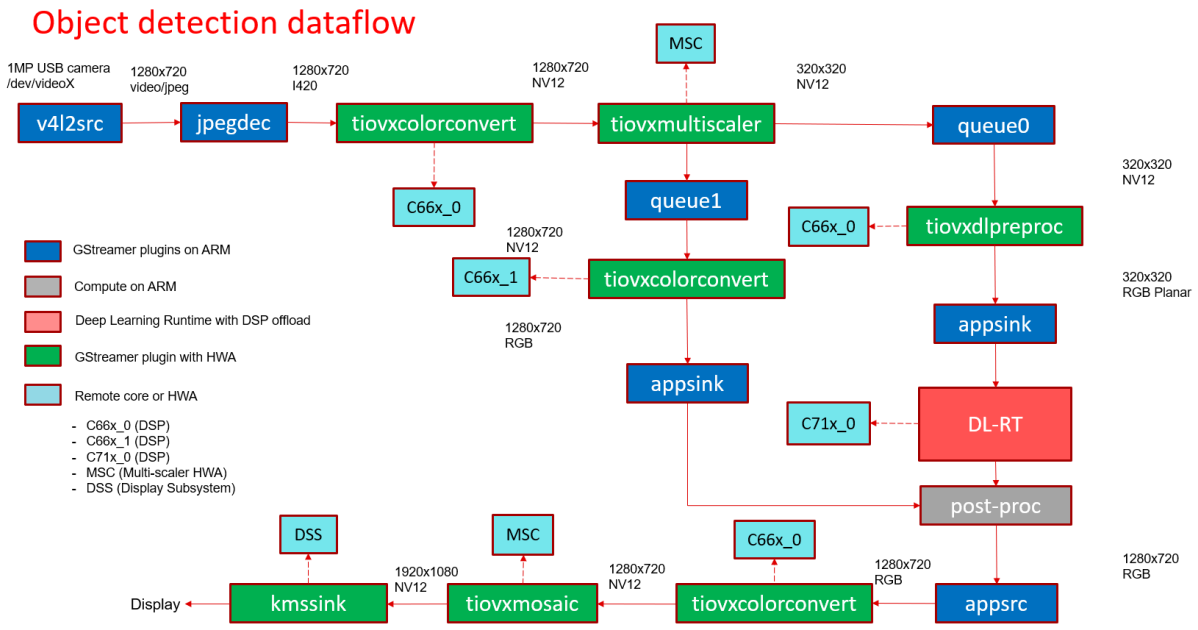


Fig. 3.46: GStreamer based data-flow pipeline with USB camera input and display output

Source : USB Camera Capture Framerate : **30 fps** Resolution : **720p** format : **JPEG**

Model	FPS	Total time (ms)	Inference time (ms)	A72 Load (%)	DDR Read BW (MB/s)	DDR Write BW (MB/s)	DDR Total BW (MB/s)	C71 Load (%)	C66_0 Load (%)	C66_1 Load (%)	C66_2 Load (%)	MCU2_0 Load (%)	MCU2_1 (%)	MSC_0 (%)	MSC_1 (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150-mobileNetV2-1p4-qat	30.80	33.22	3.02	21.60	1596	619	2215	9.0	20.0	9.0	6.0	1.0	22.17	0	0	0	0	0	0	
TFL-CL-0000-mobileNetV1-mlperf	30.69	33.19	1.04	15.93	1425	563	1988	5.0	22.0	9.0	6.0	1.0	21.90	0	0	0	0	0	0	
TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	30.69	33.25	5.00	10.24	1534	570	2104	15.0	29.0	9.0	6.0	1.0	22.67	0	0	0	0	0	0	
TVM-CL-3410-gluoncv-mxnet-mobv2	30.58	33.21	2.02	22.80	1522	617	2139	6.0	20.0	9.0	6.0	1.0	21.84	0	0	0	0	0	0	

Source : Video Video Framerate : **30 fps** Resolution : **720p** Encoding : **h264**

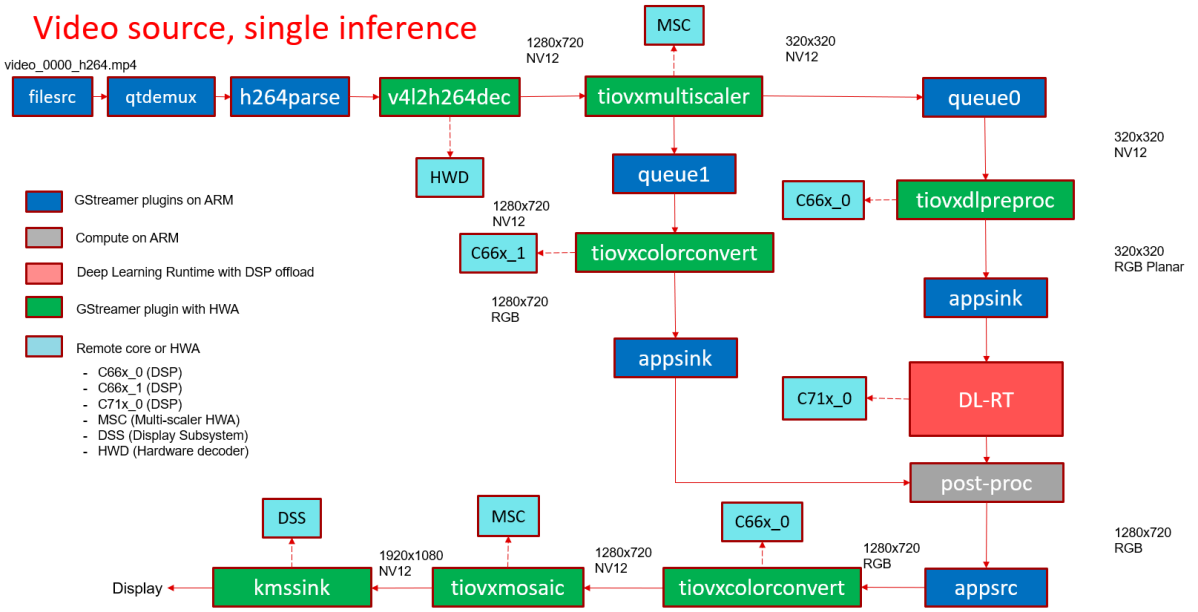


Fig. 3.47: GStreamer based data-flow pipeline with video file input source and display output

Model	FPS	Total time (ms)	Inference time (ms)	A72 Load (%)	DDR Read BW (MB/s)	DDR Write BW (MB/s)	DDR Total BW (MB/s)	C71 Load (%)	C66_0 Load (%)	C66_1 Load (%)	C66_2 Load (%)	MCU2_0 Load (%)	MCU2_1 Load (%)	MSC_0 (%)	MSC_1 (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150-mobileNetV2-1p4-qat	30.52	33.46	3.03	14.28	990	403	1393	2.0	7.0	4.0	1.0	1.0	10.27	0	0	0	0	0	0	
TFL-CL-0000-mobileNetV1-mlperf	30.77	33.47	1.07	30.76	746	97	843	2.0	2.0	1.0	1.0	1.0	15.76	0	0	0	0	0	0	
TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	30.56	33.54	5.06	22.58	736	92	828	2.0	2.0	1.0	1.0	1.0	16.9	0	0	0	0	0	0	
TVM-CL-3410-gluoncv-mxnet-mobv2	30.64	33.47	2.01	33.33	712	110	822	1.0	1.0	0.0	1.0	1.0	15.3	0	0	0	0	0	0	

Source : CSI Camera (ov5640) Capture Framerate : **30 fps** Resolution : **720p** format : **YUYV**

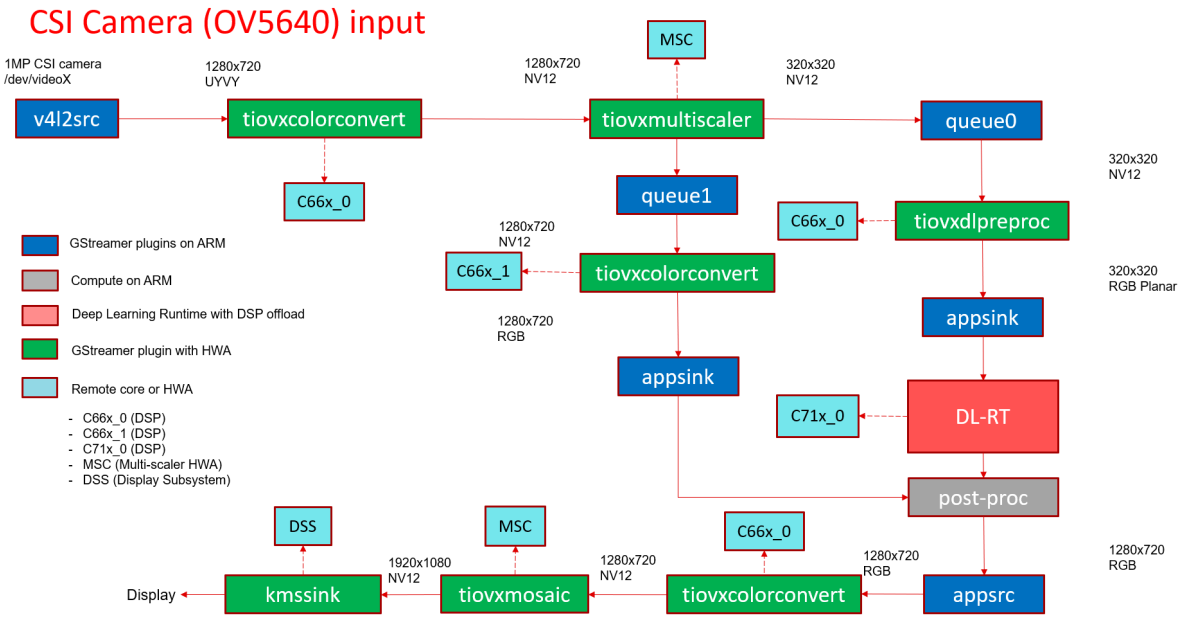


Fig. 3.48: GStreamer based data-flow pipeline for with CSI camera (OV5640) input and display output

Model	FPS	Total time (ms)	Inference time (ms)	A72 Load (%)	DDR Read BW (MB/s)	DDR Write BW (MB/s)	DDR Total BW (MB/s)	C71 Load (%)	C66_1 Load (%)	C66_2 Load (%)	MCU2_0 Load (%)	MCU2_1 Load (%)	MSC_0 (%)	MSC_1 (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150-mobileNetV2-1p4-qat	29.57	34.09	3.02	12.21	1671	699	2370	8.0	45.0	9.0	6.0	1.0	21.35	0	0	0	0	0	0
TFL-CL-0000-mobileNetV1-mlperf	29.41	34.15	1.01	10.27	1502	645	2147	5.0	47.0	9.0	6.0	1.0	20.96	0	0	0	0	0	0
TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	29.36	34.65	5.00	10.5	1610	655	2265	14.0	53.0	9.0	6.0	1.0	21.47	0	0	0	0	0	0
TVM-CL-3410-gluoncv-mxnet-mobv2	29.38	34.17	2.01	11.66	1596	698	2294	6.0	45.0	9.0	5.0	1.0	21.10	0	0	0	0	0	0

Source : CSI Camera with VISS (imx219) Capture Framerate : **30 fps** Resolution : **1080p** format : **SRGGB8**

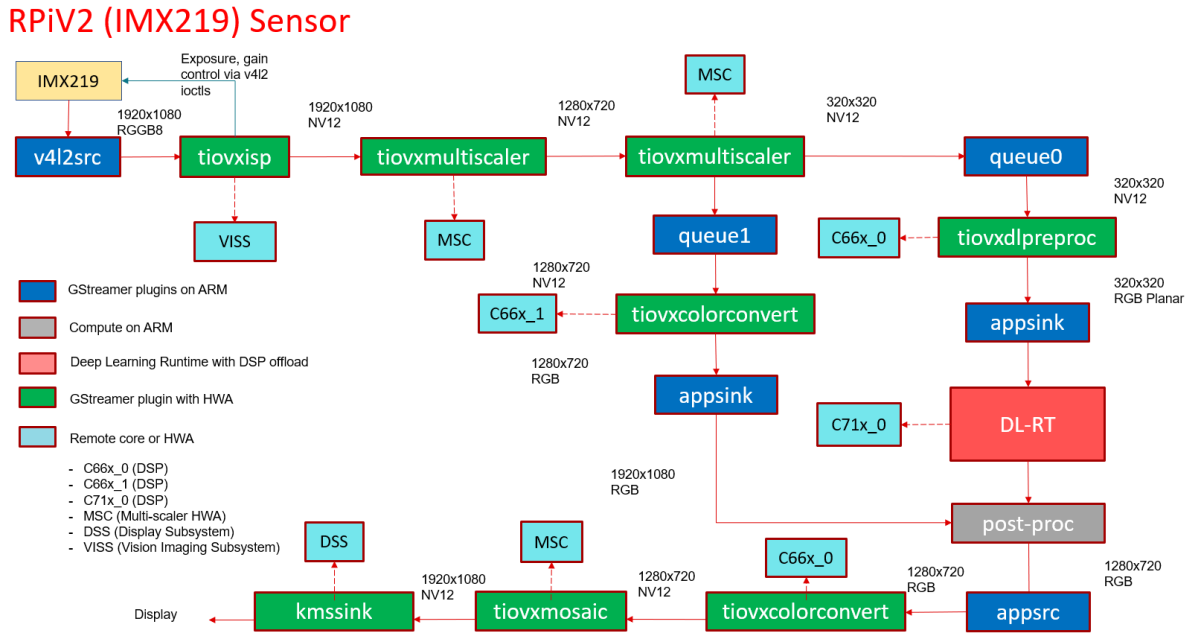


Fig. 3.49: GStreamer based data-flow pipeline with IMX219 sensor, ISP and display

Model	FPS	Total time (ms)	Inference time (ms)	A72 Load (%)	DDR Read BW (MB/s)	DDR Write BW (MB/s)	DDR Total BW (MB/s)	C71 Load (%)	C66_1 Load (%)	C66_2 Load (%)	MCU2_0 Load (%)	MCU2_1 Load (%)	MSC_0 (%)	MSC_1 (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150-mobileNetV2-1p4-qat	30.64	33.19	3.01	15.72	1781	853	2634	9.0	16.0	9.0	13.0	1.0	31.78	0	22.37	0	0	0	0
TFL-CL-0000-mobileNetV1-mlperf	30.59	33.14	1.04	12.78	1612	798	2410	5.0	18.0	9.0	13.0	1.0	31.65	0	22.31	0	0	0	0
TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	30.56	33.07	5.00	13.30	1730	809	2539	15.0	25.0	9.0	13.0	1.0	32.6	0	22.19	0	0	0	0
TVM-CL-3410-gluoncv-mxnet-mobv2	30.48	33.14	2.01	12.91	1708	852	2560	7.0	16.0	9.0	13.0	1.0	31.83	0	22.26	0	0	0	0

Source : IMX390 over FPD-Link Capture Framerate : **30 fps** Resolution : **1080p** format : **SRGGB12**

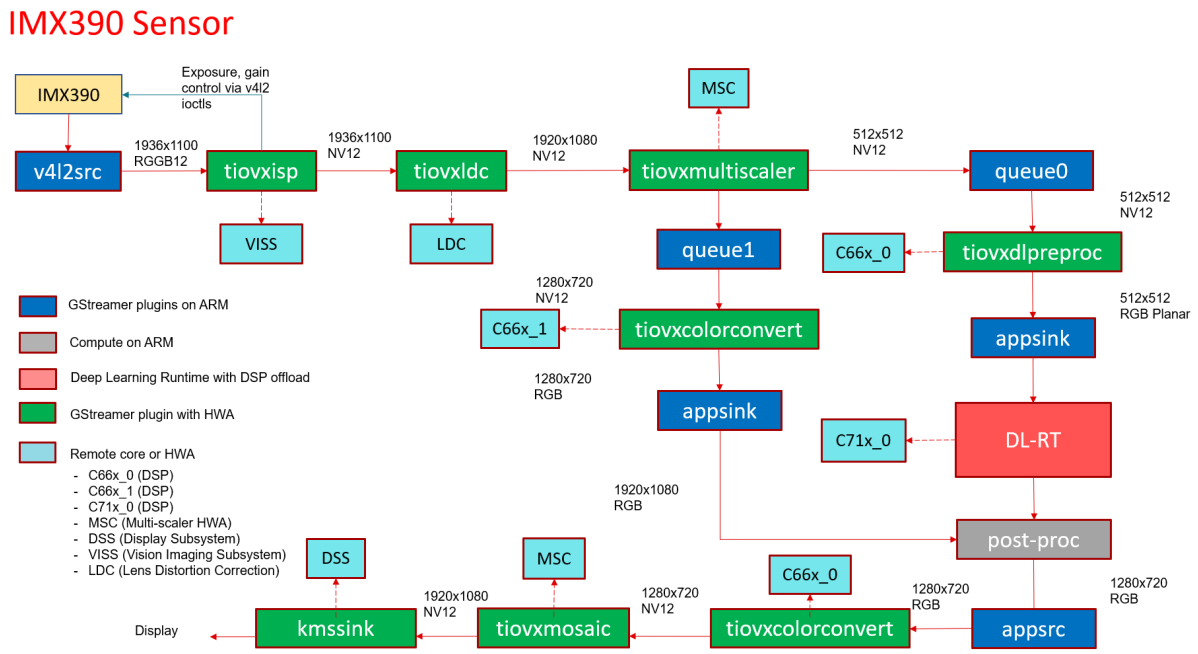


Fig. 3.50: GStreamer based data-flow pipeline with IMX390 sensor, ISP, LDC and display

Model	FPS	Total time (ms)	Inference time (ms)	A72 Load (%)	DDR Read BW (MB/s)	DDR Write BW (MB/s)	DDR Total BW (MB/s)	C71 Load (%)	C66_1 Load (%)	C66_2 Load (%)	MCU2_0 Load (%)	MCU2_1 Load (%)	MSC_0 (%)	MSC_1 (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150-mobileNetV2-1p4-qat	30.59	33.15	3.09	25.18	2207	1102	3309	10.0	16.0	9.0	14.0	1.0	31.73	0	22.94	0	10.8	0	0
TFL-CL-0000-mobileNetV1-mlperf	30.53	33.15	1.21	16.20	2019	1040	3059	5.0	18.0	9.0	15.0	1.0	32.80	0	23.34	0	10.10	0	0
TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	30.43	33.13	5.02	23.7	2201	1067	3268	15.0	25.0	9.0	14.0	1.0	32.80	0	22.88	0	9.95	0	
TVM-CL-3410-gluoncv-mxnet-mobv2	30.44	33.16	2.12	21.50	2111	1100	3211	7.0	16.0	9.0	15.0	1.0	32.28	0	22.88	0	10.6	0	0

Test Report

Here is the summary of the sanity tests we ran with both Python and C++ demos. Test cases vary with different inputs, outputs, runtime, models, python/c++ apps.

1. Inputs:
    - Camera (Logitech C270, 1280x720, JPEG)

- Camera (Omnivision OV5640, 1280x720, YUV)
- Camera (Rpi v2 Sony IMX219, 1920x1080, RAW)
- Image files (30 images under edge_ai_apps/data/images)
- Video file (10s video 1 file under edge_ai_apps/data/videos)
- RSTP Video Server

2. Outputs:
   - Display (eDP or HDMI)
   - File write to SD card
3. Inference Type:
   - Image classification
   - Object detection
   - Semantic segmentation
4. Runtime/models:
   - DLR
   - TFLite
   - ONNX
5. Applications:
   - Python
   - C++
6. Platform:
   - Host OS
   - Docker

Demo Apps test report

Single Input Single Output

Category	# test case	Pass	Fail
Host OS - Python	99	99	0
Host OS - C++	99	99	0

S.No	Models	Input	Output	Host OS-C-
1	TVM-CL-3410-gluoncv-mxnet-mobv2	Image	Display	Pass
2	TVM-CL-3410-gluoncv-mxnet-mobv2	Image	Video-Filewrite	Fail
3	TVM-CL-3410-gluoncv-mxnet-mobv2	Image	Image-Filewrite	Pass
4	TVM-CL-3410-gluoncv-mxnet-mobv2	Video	Display	Pass
5	TVM-CL-3410-gluoncv-mxnet-mobv2	Video	Video-Filewrite	Pass
6	TVM-CL-3410-gluoncv-mxnet-mobv2	USB Camera	Display	Pass
7	TVM-CL-3410-gluoncv-mxnet-mobv2	USB Camera	Video-Filewrite	Pass
8	TVM-CL-3410-gluoncv-mxnet-mobv2	CSI Camera	Display	Pass
9	TVM-CL-3410-gluoncv-mxnet-mobv2	CSI Camera	Video-Filewrite	Pass
10	TVM-CL-3410-gluoncv-mxnet-mobv2	RPI Camera	Display	Pass
11	TVM-CL-3410-gluoncv-mxnet-mobv2	RPI Camera	Video-Filewrite	Pass
12	TVM-CL-3410-gluoncv-mxnet-mobv2	RTSP - Video	Display	Pass
13	TVM-CL-3410-gluoncv-mxnet-mobv2	RTSP - Video	Video-Filewrite	Pass

Table 3.8 – continued from previous page

S.No	Models	Input	Output	Host OS-C-
14	TFL-CL-0000-mobileNetV1-mlperf	Image	Display	Pass
15	TFL-CL-0000-mobileNetV1-mlperf	Image	Video-Filewrite	Fail
16	TFL-CL-0000-mobileNetV1-mlperf	Image	Image-Filewrite	Pass
17	TFL-CL-0000-mobileNetV1-mlperf	Video	Display	Pass
18	TFL-CL-0000-mobileNetV1-mlperf	Video	Video-Filewrite	Pass
19	TFL-CL-0000-mobileNetV1-mlperf	USB Camera	Display	Pass
20	TFL-CL-0000-mobileNetV1-mlperf	USB Camera	Video-Filewrite	Pass
21	TFL-CL-0000-mobileNetV1-mlperf	CSI Camera	Display	Pass
22	TFL-CL-0000-mobileNetV1-mlperf	CSI Camera	Video-Filewrite	Pass
23	TFL-CL-0000-mobileNetV1-mlperf	RPI Camera	Display	Pass
24	TFL-CL-0000-mobileNetV1-mlperf	RPI Camera	Video-Filewrite	Pass
25	TFL-CL-0000-mobileNetV1-mlperf	RTSP - Video	Display	Pass
26	TFL-CL-0000-mobileNetV1-mlperf	RTSP - Video	Video-Filewrite	Pass
27	ONR-CL-6360-regNetx-200mf	Image	Display	Pass
28	ONR-CL-6360-regNetx-200mf	Image	Video-Filewrite	Fail
29	ONR-CL-6360-regNetx-200mf	Image	Image-Filewrite	Pass
30	ONR-CL-6360-regNetx-200mf	Video	Display	Pass
31	ONR-CL-6360-regNetx-200mf	Video	Video-Filewrite	Pass
32	ONR-CL-6360-regNetx-200mf	USB Camera	Display	Pass
33	ONR-CL-6360-regNetx-200mf	USB Camera	Video-Filewrite	Pass
34	ONR-CL-6360-regNetx-200mf	CSI Camera	Display	Pass
35	ONR-CL-6360-regNetx-200mf	CSI Camera	Video-Filewrite	Pass
36	ONR-CL-6360-regNetx-200mf	RPI Camera	Display	Pass
37	ONR-CL-6360-regNetx-200mf	RPI Camera	Video-Filewrite	Pass
38	ONR-CL-6360-regNetx-200mf	RTSP - Video	Display	Pass
39	ONR-CL-6360-regNetx-200mf	RTSP - Video	Video-Filewrite	Pass
40	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Image	Display	Pass
41	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Image	Video-Filewrite	Fail
42	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Image	Image-Filewrite	Pass
43	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Video	Display	Pass
44	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Video	Video-Filewrite	Pass
45	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	USB Camera	Display	Pass
46	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	USB Camera	Video-Filewrite	Pass
47	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	CSI Camera	Display	Pass
48	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	CSI Camera	Video-Filewrite	Pass
49	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	RPI Camera	Display	Pass
50	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	RPI Camera	Video-Filewrite	Pass
51	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	RTSP - Video	Display	Pass
52	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	RTSP - Video	Video-Filewrite	Pass
53	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Image	Display	Pass
54	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Image	Video-Filewrite	Fail
55	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Image	Image-Filewrite	Pass
56	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Video	Display	Pass
57	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Video	Video-Filewrite	Pass
58	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	USB Camera	Display	Pass
59	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	USB Camera	Video-Filewrite	Pass
60	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	CSI Camera	Display	Pass
61	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	CSI Camera	Video-Filewrite	Pass
62	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	RPI Camera	Display	Pass
63	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	RPI Camera	Video-Filewrite	Pass
64	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	RTSP - Video	Display	Pass
65	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	RTSP - Video	Video-Filewrite	Pass
66	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Image	Display	Pass

Table 3.8 – continued from previous page

S.No	Models	Input	Output	Host OS-C-
67	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Image	Video-Filewrite	Fail
68	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Image	Image-Filewrite	Pass
69	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Video	Display	Pass
70	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Video	Video-Filewrite	Pass
71	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	USB Camera	Display	Pass
72	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	USB Camera	Video-Filewrite	Pass
73	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	CSI Camera	Display	Pass
74	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	CSI Camera	Video-Filewrite	Pass
75	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	RPI Camera	Display	Pass
76	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	RPI Camera	Video-Filewrite	Pass
77	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	RTSP - Video	Display	Pass
78	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	RTSP - Video	Video-Filewrite	Pass
79	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Image	Display	Pass
80	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Image	Video-Filewrite	Fail
81	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Image	Image-Filewrite	Pass
82	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Video	Display	Pass
83	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Video	Video-Filewrite	Pass
84	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	USB Camera	Display	Pass
85	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	USB Camera	Video-Filewrite	Pass
86	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	CSI Camera	Display	Pass
87	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	CSI Camera	Video-Filewrite	Pass
88	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	RPI Camera	Display	Pass
89	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	RPI Camera	Video-Filewrite	Pass
90	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	RTSP - Video	Display	Pass
91	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	RTSP - Video	Video-Filewrite	Pass
92	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Image	Display	Pass
93	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Image	Video-Filewrite	Fail
94	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Image	Image-Filewrite	Pass
95	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Video	Display	Pass
96	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Video	Video-Filewrite	Pass
97	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	USB Camera	Display	Pass
98	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	USB Camera	Video-Filewrite	Pass
99	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	CSI Camera	Display	Pass
100	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	CSI Camera	Video-Filewrite	Pass
101	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	RPI Camera	Display	Pass
102	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	RPI Camera	Video-Filewrite	Pass
103	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	RTSP - Video	Display	Pass
104	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	RTSP - Video	Video-Filewrite	Pass
105	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Image	Display	Pass
106	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Image	Video-Filewrite	Fail
107	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Image	Image-Filewrite	Pass
108	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Video	Display	Pass
109	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Video	Video-Filewrite	Pass
110	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	USB Camera	Display	Pass
111	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	USB Camera	Video-Filewrite	Pass
112	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	CSI Camera	Display	Pass
113	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	CSI Camera	Video-Filewrite	Pass
114	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	RPI Camera	Display	Pass
115	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	RPI Camera	Video-Filewrite	Pass
116	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	RTSP - Video	Display	Pass
117	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	RTSP - Video	Video-Filewrite	Pass

Single Input Multi Output

Category	# test case	Pass	Fail
Host OS - Python	15	15	0
docker - Python	15	15	0
Host OS - C++	15	15	0
Docker - C++	15	15	0

S.No	Models	Input	Output	Host OS-C++	Host OS-Python	Docker-C++	Docker-Python	Comments
1	2 Models (TFL-CL, ONR-SS)	%04d.jpg	Display	Pass	Pass	Pass	Pass	
2	3-Models (TVM-CL, TFL-OD, ONR-SS)	%04d.jpg	Display	Pass	Pass	Pass	Pass	
3	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	%04d.jpg	Display	Pass	Pass	Pass	Pass	
4	2 Models (TFL-CL, ONR-SS)	video_0000.mp4	Display	Pass	Pass	Pass	Pass	
5	3-Models (TVM-CL, TFL-OD, ONR-SS)	video_0000.mp4	Display	Pass	Pass	Pass	Pass	
6	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	video_0000.mp4	Display	Pass	Pass	Pass	Pass	
7	2 Models (TFL-CL, ONR-SS)	USB_camera	Display	Pass	Pass	Pass	Pass	
8	3-Models (TVM-CL, TFL-OD, ONR-SS)	USB_camera	Display	Pass	Pass	Pass	Pass	
9	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	USB_camera	Display	Pass	Pass	Pass	Pass	
10	2 Models (TFL-CL, ONR-SS)	CSI_camera	Display	Pass	Pass	Pass	Pass	
11	3-Models (TVM-CL, TFL-OD, ONR-SS)	CSI_camera	Display	Pass	Pass	Pass	Pass	
12	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	CSI_camera	Display	Pass	Pass	Pass	Pass	
13	2 Models (TFL-CL, ONR-SS)	rtsp	Display	Pass	Pass	Pass	Pass	
14	3-Models (TVM-CL, TFL-OD, ONR-SS)	rtsp	Display	Pass	Pass	Pass	Pass	
15	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	rtsp	Display	Pass	Pass	Pass	Pass	

Multi Input Multi Output

Category	# test case	Pass	Fail
Host OS - Python	8	8	0
docker - Python	8	8	0
Host OS - C++	8	8	0
Docker - C++	8	8	0

S.No	Models	Input	Output	Host OS-C++	Host OS-Python	Docker-C++	Docker-Python	Comments
1	2 Models (TVM-CL, TFL-OD)	%04d.jpg,video_0000.mp4	Display	Pass	Pass	Pass	Pass	
2	2 Models (TVM-OD, ONR-SS)	%04d.jpg,rtsp	Video-Filewrite	Pass	Pass	Pass	Pass	
3	2 Models (ONR-CL, TVM-SS)	%04d.jpg,USB_camera	Display	Pass	Pass	Pass	Pass	
4	3-Models (TVM-CL, TFL-OD, ONR-SS)	%04d.jpg,CSI_camera,rtsp	Video-Filewrite	Pass	Pass	Pass	Pass	
5	3-Models (TVM-CL, TFL-OD, ONR-SS)	video_0000.mp4,rtsp,%04d.jpg	Display	Pass	Pass	Pass	Pass	
6	3-Models (TFL-CL, ONR-CL, TVM-SS)	video_0000.mp4,USB_camera,CSI_camera	Video-Filewrite	Pass	Pass	Pass	Pass	
7	4-Models (TVM-CL, TFL-SS, ONR-OD, TFL-CL)	USB_camera,CSI_camera	Display	Pass	Pass	Pass	Pass	
8	4-Models (TVM-SS, TFL-SS, ONR-SS, ONR-OD)	USB_camera,video_0000.mp4	Video-Filewrite	Pass	Pass	Pass	Pass	

Note:

- Video file from RTSP server used for RTSP input test
- Please refer to the TI Edge AI SDK release notes and known issues for more details

3.6 Additional Support Information

All support for this design is through BeagleBoard.org community at: link: BeagleBoard.org forum .

3.6.1 Hardware Design

You can find all BeagleBone AI-64 hardware files here under the *hw* folder.

3.6.2 Software Updates

Follow instructions below to download the latest image for your BeagleBone AI-64:

1. Go to BeagleBoard.org distro page.
2. *Filter Software Distributions for BeagleBone AI-64* from dropdown and download the image.

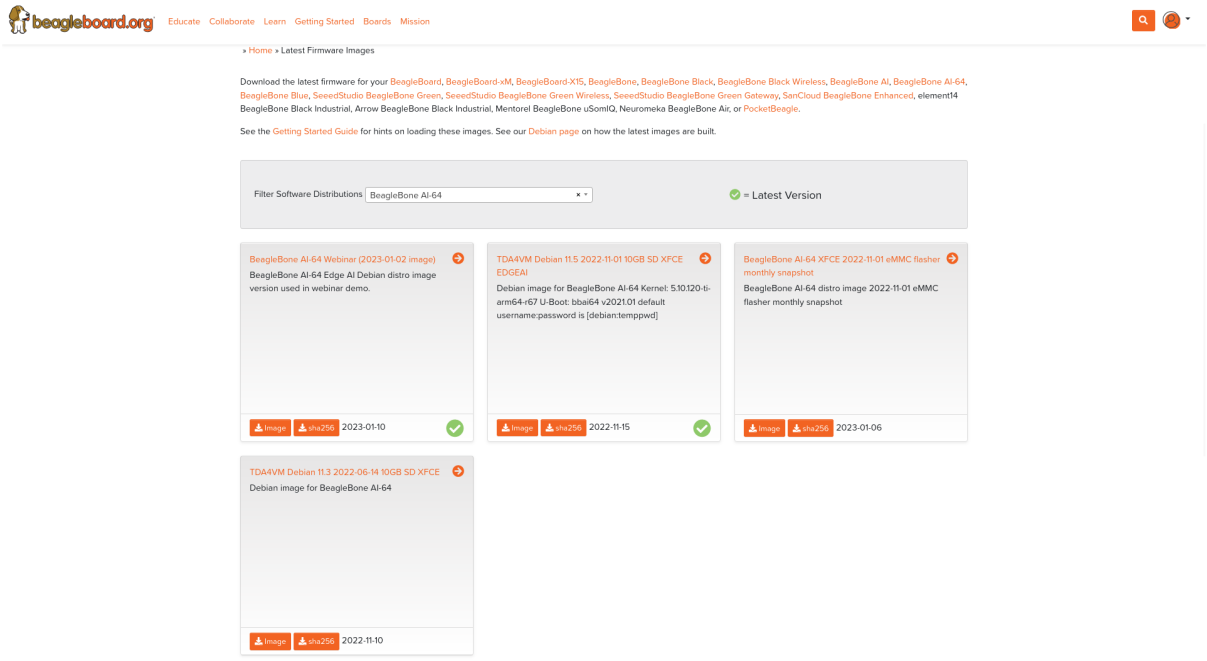


Fig. 3.51: Filter Software Distributions for BeagleBone AI-64

Tip: You can follow the *Update board with latest software* guide for more information on flashing the downloaded image to your board.

To see what SW revision is loaded into the eMMC check */etc/dogtag*. It should look something like as shown below,

```
` root@BeagleBone:~# cat /etc/dogtag BeagleBoard.org Debian Bullseye Xfce Image 2022-01-14 `
```

3.6.3 RMA Support

If you feel your board is defective or has issues, request an Return Merchandise Application (RMA) by filling out the form at http://beagleboard.org/support/rma . You will need the serial number and revision of the board. The serial numbers and revisions keep moving. Different boards can have different locations depending on when they were made. The following figures show the three locations of the serial and revision number.

3.6.4 Troubleshooting video output issues

Warning: When connecting to an HDMI monitor, make sure your miniDP adapter is *active*. A *passive* adapter will not work. See *Display adapters*.

3.6.5 Getting Help

If you need some up to date troubleshooting techniques, you can post your queries on link: BeagleBoard.org forum

3.6.6 Change History

This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

Document Change History

This table seeks to keep track of major revision cycles in the documentation. Moving forward, we'll seek to align these version numbers across all of the various documentation.

Table 3.9: Table 1: Change History

Rev	Changes	Date	By
0.0.1	AI-64 initial prototype	September 2021	James Anderson
0.0.2	AI-64 final prototype	December 2021	James Anderson
0.0.3	AI-64 initial production release	June 9, 2022	Deepak Khatri and Jason Kridner

Board Changes

Be sure to check the board revision history in the schematic file in the BeagleBone AI-64 git repository . Also check the issues list .

Rev B We are starting with revision B based on this being an update to the BeagleBone Black AI. However, because this board ended up being so different, we've decided to name it BeagleBone AI-64, rather than simply a new revision. This refers to the Seeed release on 21 Dec 2021 of "BeagleBone AI-64_SCH_Rev B_211221". This is the initial production release.

3.6.7 Mechanical Details

Dimensions and Weight

Size: 102.5 x 80 (4" x 3.15")

Max height: #TODO#

PCB Layers: #TODO#

PCB thickness: 2mm (0.08”)

RoHS Compliant: Yes

Weight: 192gm

Silkscreen and Component Locations

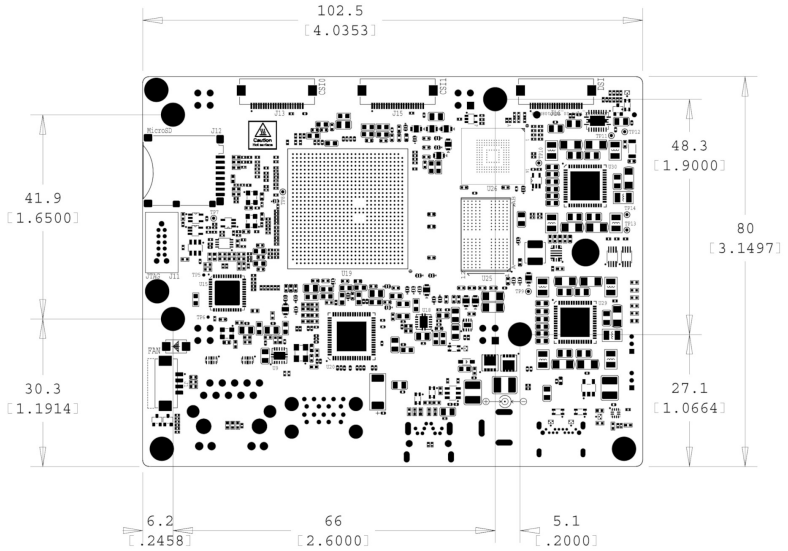


Fig. 3.52: Board Dimensions

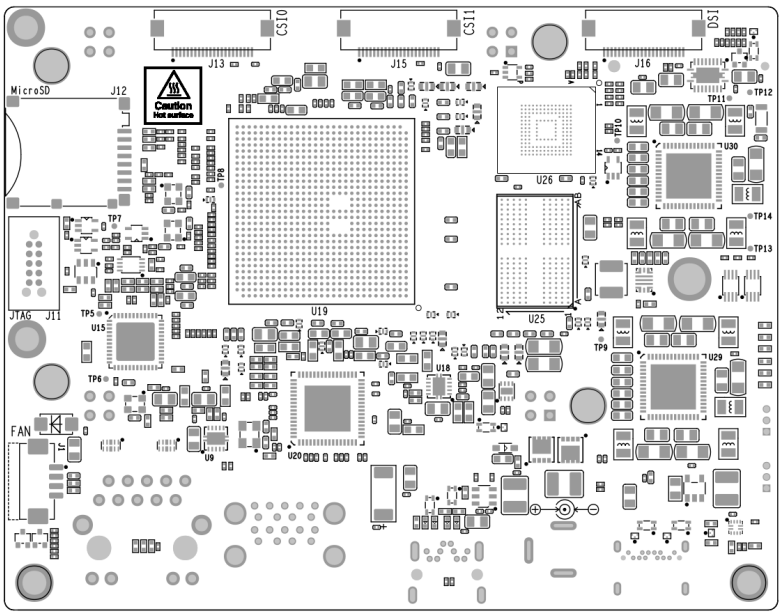


Fig. 3.53: Top silkscreen

3.6.8 Pictures

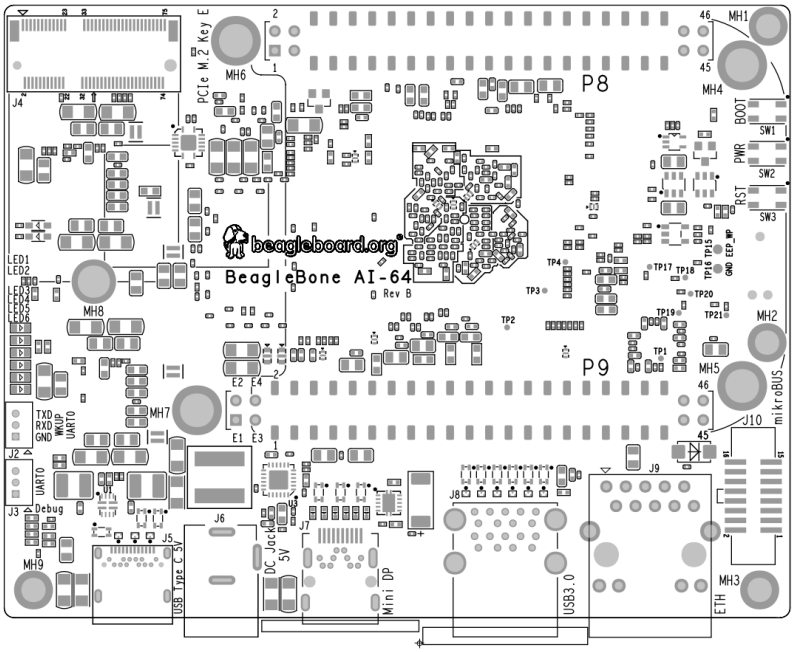


Fig. 3.54: Bottom silkscreen

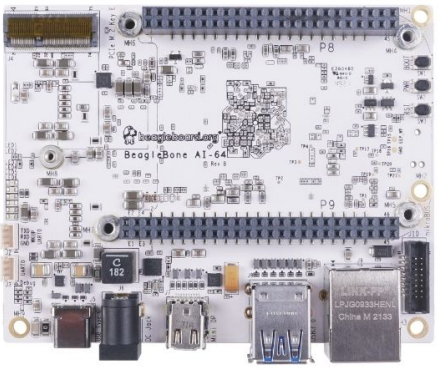


Fig. 3.55: BeagleBone AI-64 front

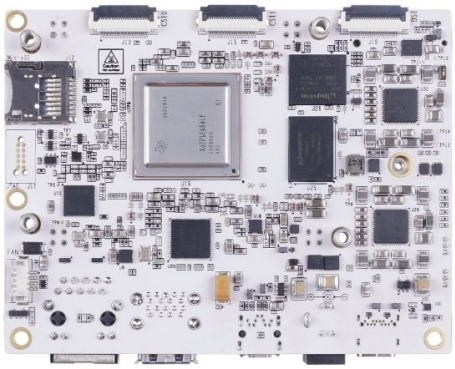


Fig. 3.56: BeagleBone AI-64 back

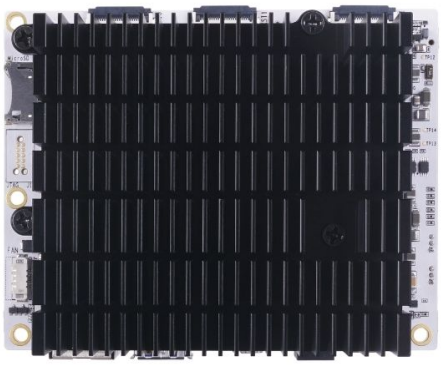


Fig. 3.57: BeagleBone AI-64 back with heatsink



Fig. 3.58: BeagleBone AI-64 front at 45° angle

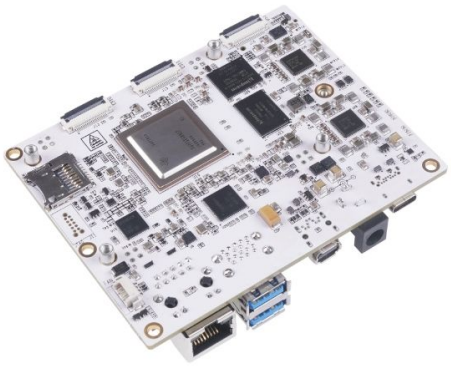


Fig. 3.59: BeagleBone AI-64 back at 45° angle



Fig. 3.60: BeagleBone AI-64 back with heatsink at 45° angle

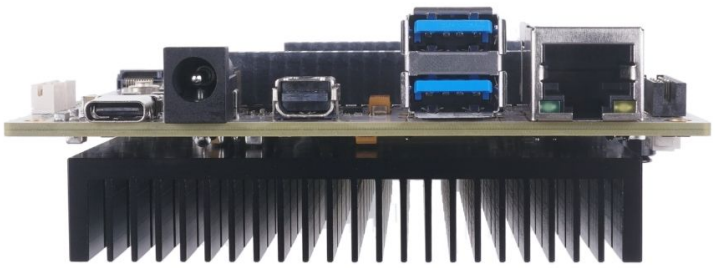


Fig. 3.61: BeagleBone AI-64 ports

Chapter 4

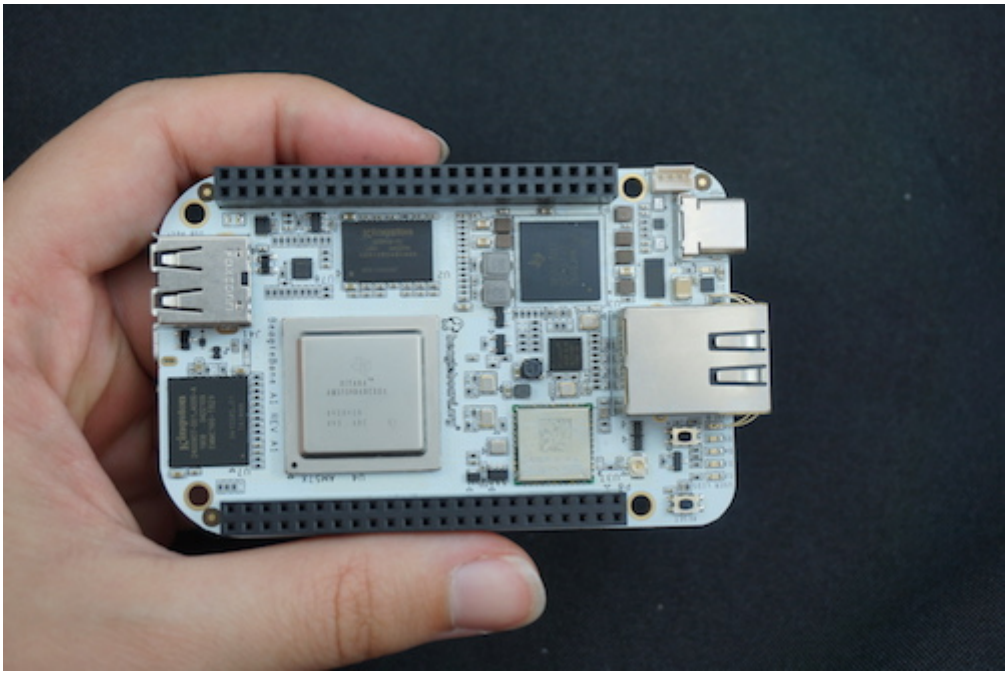
BeagleBone AI

BeagleBone AI is based on the Texas Instruments AM5729 dual-core Cortex-A15 SoC with flexible BeagleBone Black header and mechanical compatibility. BeagleBone AI makes it easy to explore how artificial intelligence (AI) can be used in everyday life via the TI C66x digital-signal-processor (DSP) cores and embedded-vision-engine (EVE) cores supported through an optimized TIDL machine learning OpenCL API with pre-installed tools. Focused on everyday automation in industrial, commercial and home applications.



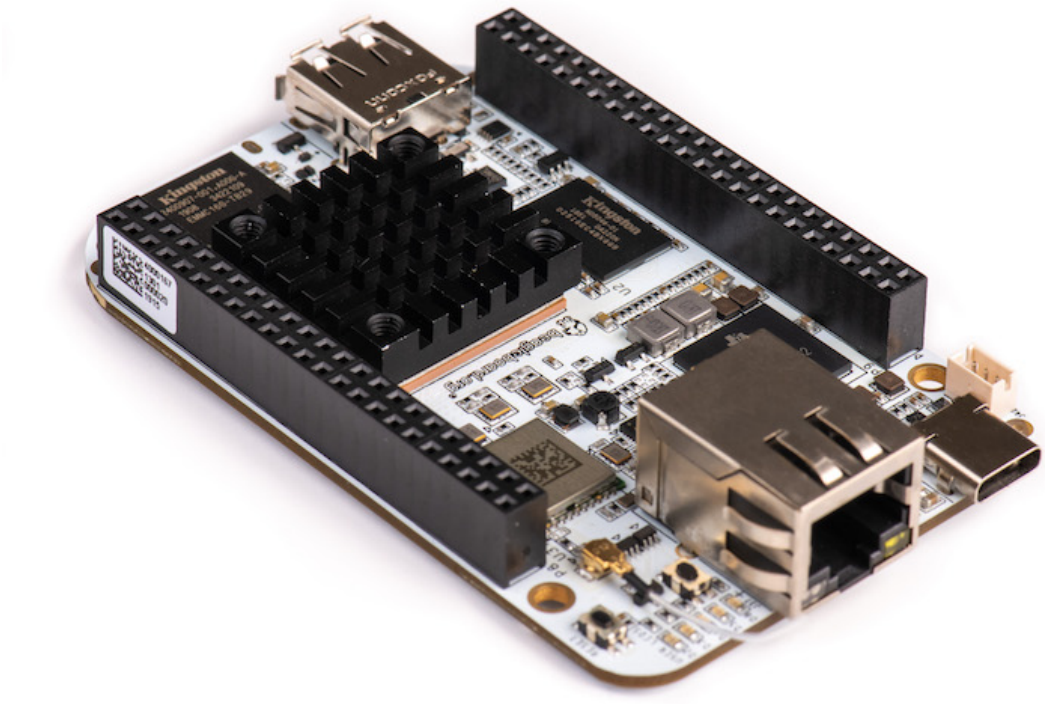
License Terms

- This documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page
- For export, emissions and other compliance, see *Additional Support Information*



4.1 Introduction

Built on the proven BeagleBoard.org® open source Linux approach, BeagleBone® AI fills the gap between small SBCs and more powerful industrial computers. Based on the Texas Instruments AM5729, developers have access to the powerful SoC with the ease of BeagleBone® Black header and mechanical compatibility. BeagleBone® AI makes it easy to explore how artificial intelligence (AI) can be used in everyday life via TI C66x digital-signal-processor (DSP) cores and embedded-vision-engine (EVE) cores supported through an optimized TIDL machine learning OpenCL API with pre-installed tools. Focused on everyday automation in industrial, commercial and home applications.



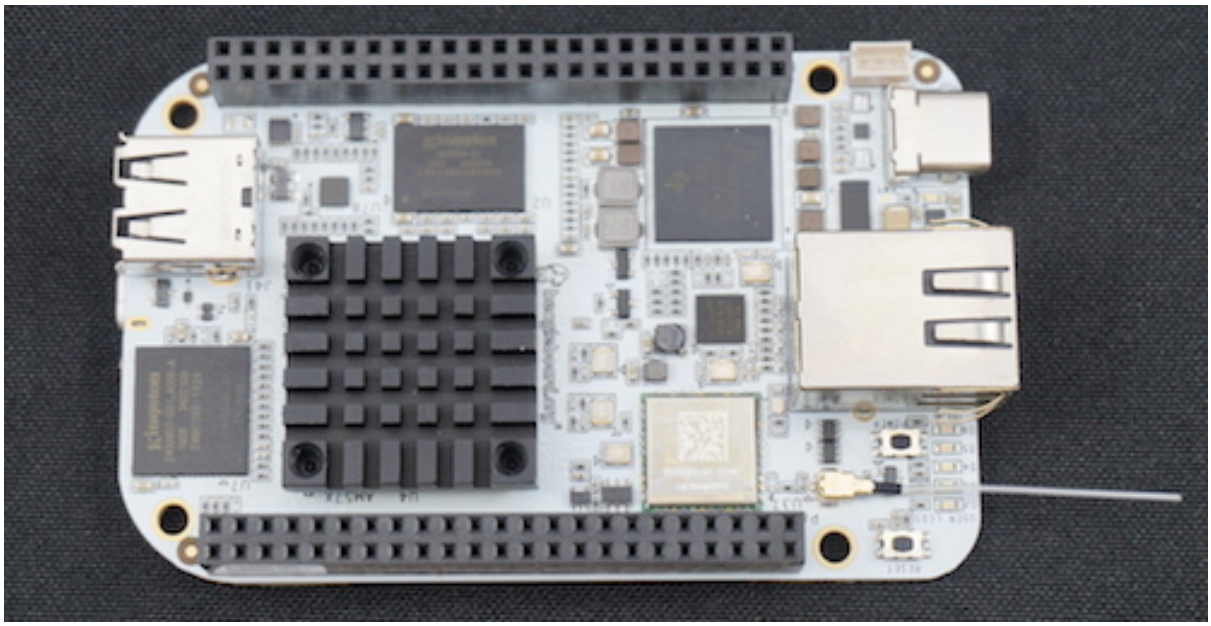
4.2 Connecting Up Your BeagleBone AI

4.2.1 What's In the Box

BeagleBone® AI comes in the box with the heat sink and antenna already attached. Developers can get up and running in five minutes with no microSD card needed. BeagleBone® AI comes preloaded with a Linux distribution. In the box you will find:

- BeagleBone® AI
- Quick Start Guide

TODO: Add links to the design materials for both



4.2.2 What's Not in the Box

You will need to purchase:

- USB C cable or USB C to USB A cable
- MicroSD Card (optional)
- Serial cable (optional)

More information or to purchase a replacement heat sink or antenna, please go to these websites:

- Antenna
- Heat Sink

4.2.3 Fans

The pre-attached heat sink has M3 holes spaced 20x20 mm. The height of the heat sink clears the USB type A socket, and all other components on the board except the 46-way header sockets and the Ethernet socket.

If you run all of the accelerators or have an older software image, you'll likely need fan. To find a fan, visit the link to fans in the FAQ.

Caution: BeagleBone AI can run **HOT**! Even without running the accelerators, getting up to 70C is not uncommon.

Official BeagleBone Fan Cape: https://www.newark.com/element14/6100310/beaglebone-ai-fan-cape/dp/50AH3704

TODO: create short-links for any long URLs so that text works.

4.2.4 Main Connection Scenarios

This section will describe how to connect the board for use. The board can be configured in several different ways. Below we will walk through the most common scenarios. NOTE: These connection scenarios are dependent on the software image presently on your BeagleBone® AI. When all else fails, follow the instructions at https://beagleboard.org/upgrade

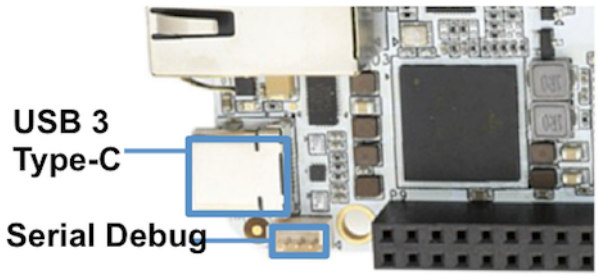
- *Tethered to a PC via USB C cable*
- *Standalone Desktop with powered USB hub, display, keyboard and mouse*
- *Wireless Connection to BeagleBone® AI*

4.2.5 Tethered to a PC

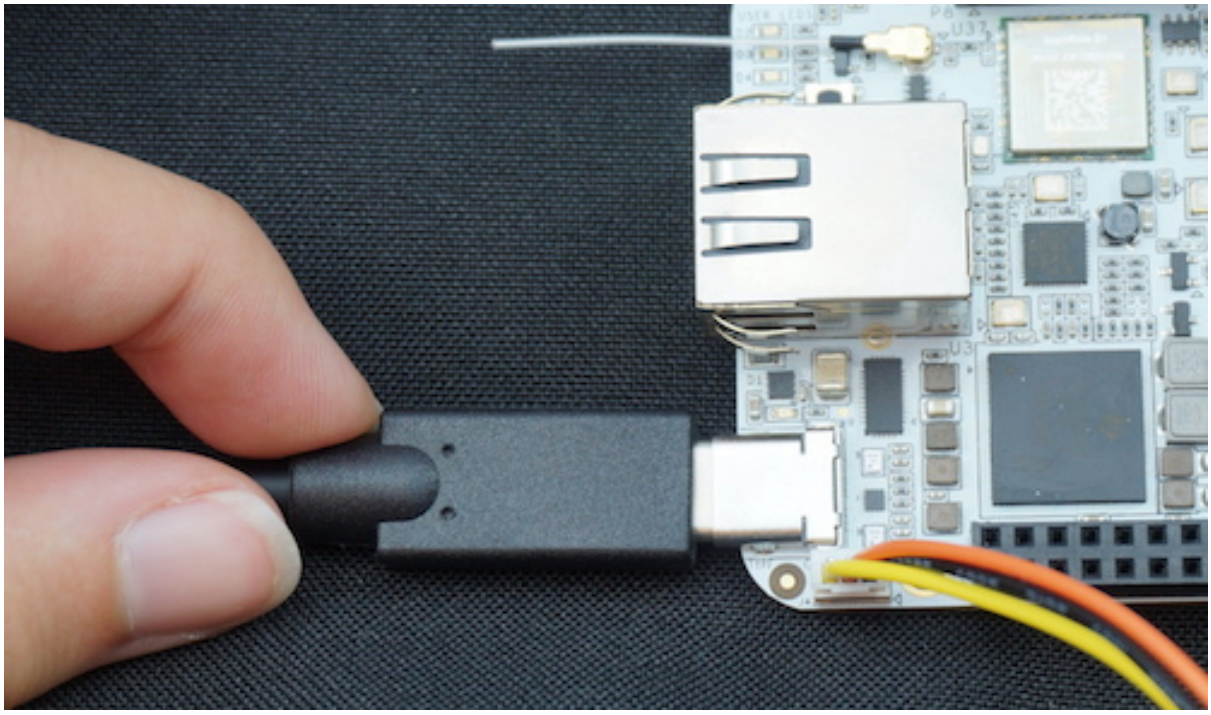
The most common way to program BeagleBone® AI is via a USB connection to a PC. If your computer has a USB C type port, BeagleBone® AI will both communicate and receive power directly from the PC. If your computer does not support USB C type, you can utilize a powered USB C hub to power and connect to BeagleBone® AI which in turn will connect to your PC. You can also use a powered USB C hub to power and connect peripheral devices such as a USB camera. After booting, the board is accessed either as a USB storage device or via the browser on the PC. You will need Chrome or Firefox on the PC.

NOTE:Start with this image “am57xx-eMMC-flasher-debian-10.3-iot-tidl-armhf-2020-04-06-6gb.img.xz” loaded on your BeagleBone® AI.

1. Locate the USB Type-C connector on BeagleBone® AI



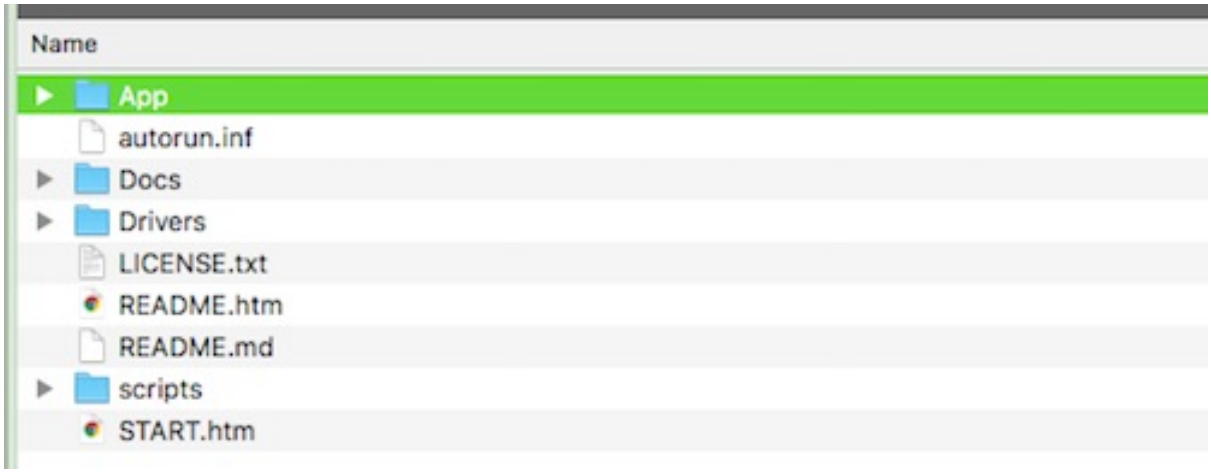
2. Connect a USB type-C cable to BeagleBone® AI USB type-C port.



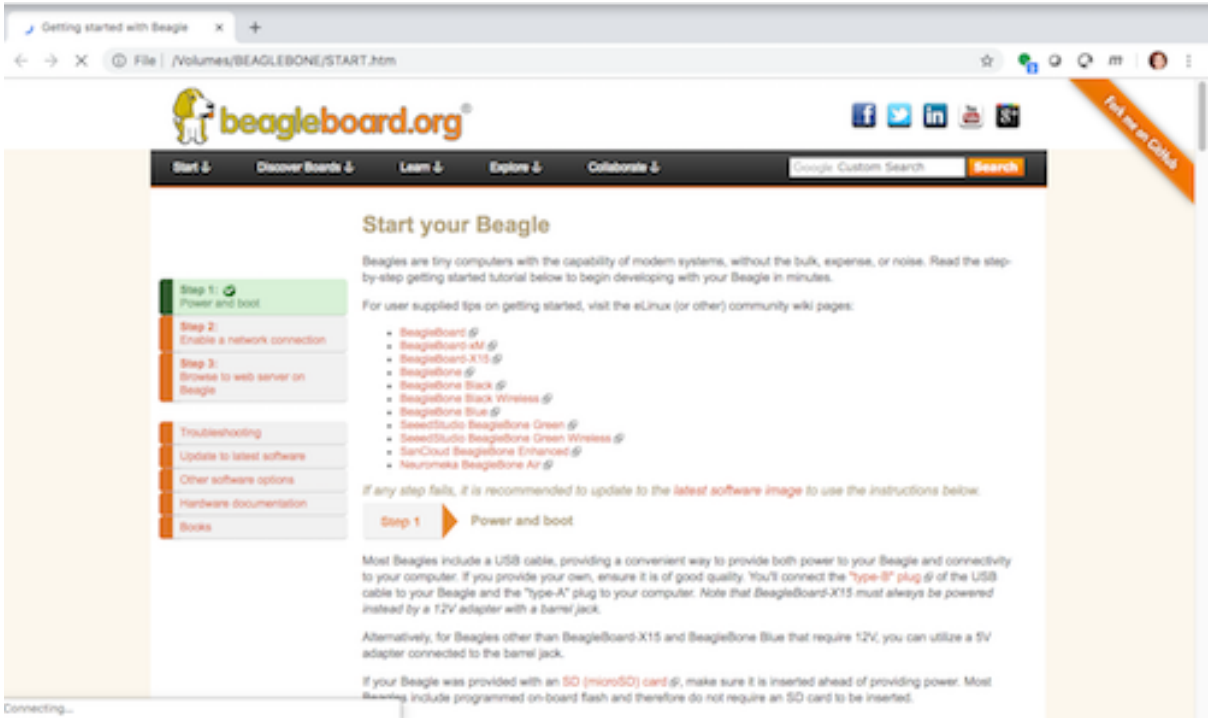
3. Connect the other end of the USB cable to the PC USB 3 port.

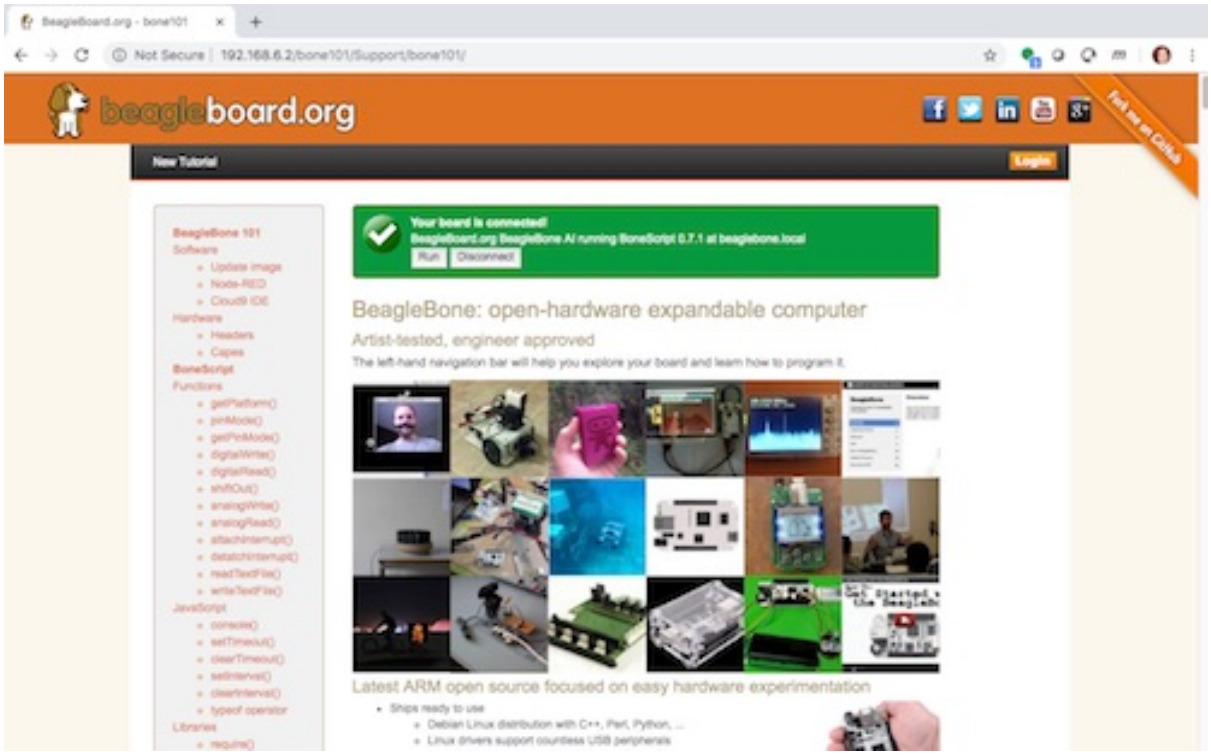


4. BeagleBone® AI will boot.
5. You will notice some of the 5 user LEDs flashing
6. Look for a new mass storage drive to appear on the PC.

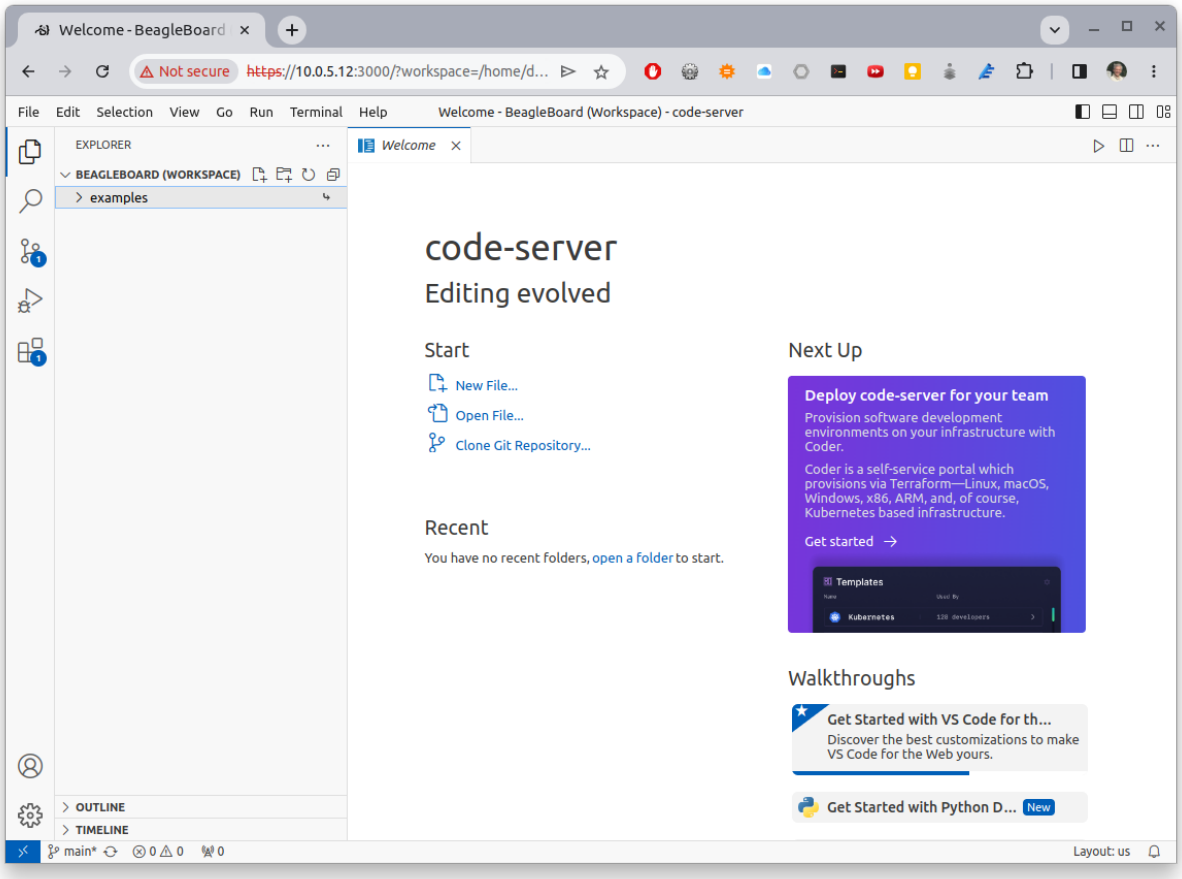


7. Open the drive and open START.HTM with your web browser.



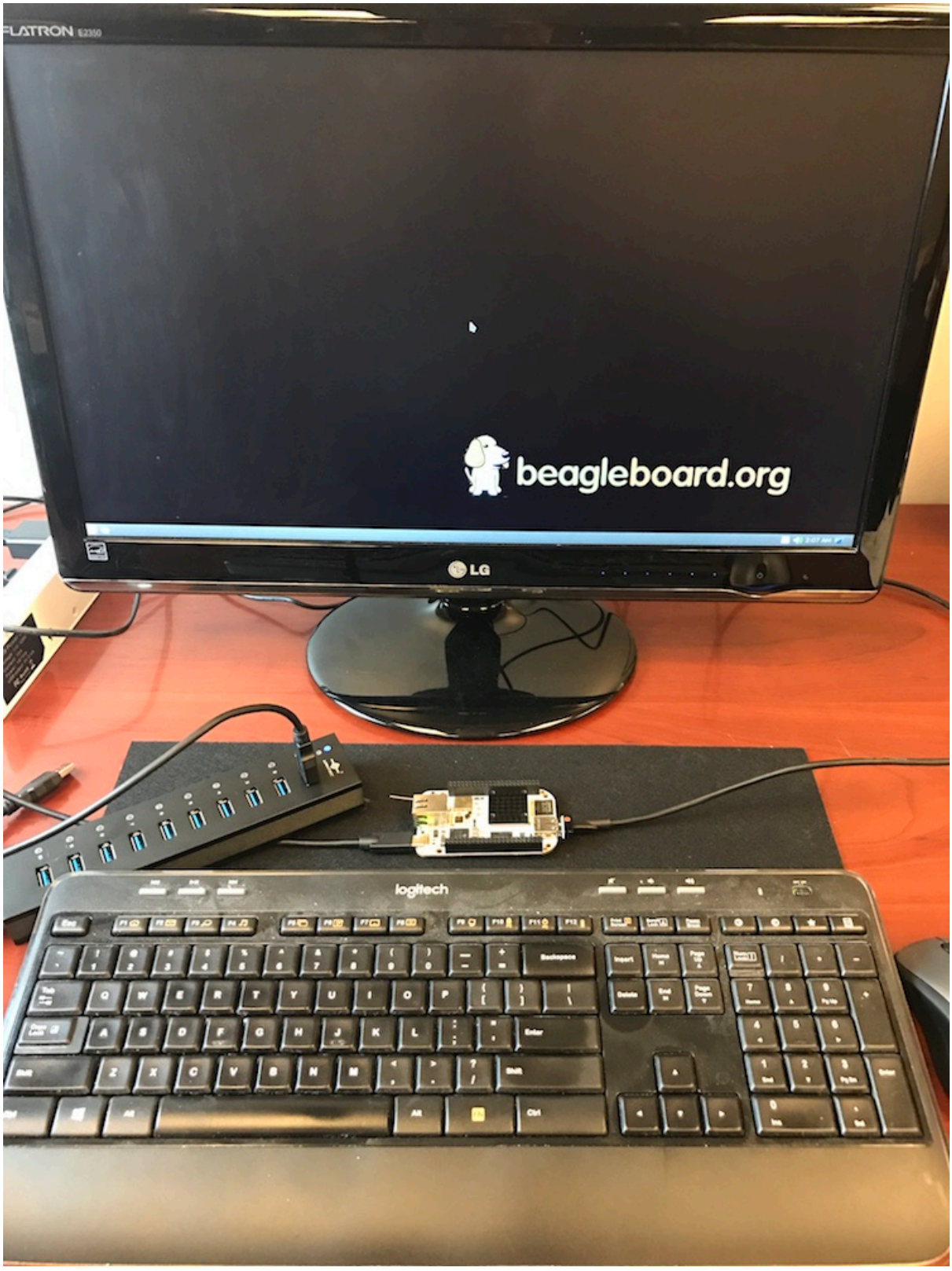


8. Follow the instructions in the browser window.



9. Go to Visual Studio Code IDE.

4.2.6 Standalone w/Display and Keyboard/Mouse



Note: This configuration requires loading the latest debian 9 image from https://elinux.org/Beagleboard:Latest-images-testing

Load “am57xx-eMMC-flasher-debian-9.13-lxqt-tidl-armhf-2020-08-25-6gb.img.xz” image on the BeagleBone® AI

1. Connect a combo keyboard and mouse to BeagleBone® AI’s USB host port.
2. Connect a microHDMI-to-HDMI cable to BeagleBone® AI’s microHDMI port.
3. Connect the microHDMI-to-HDMI cable to an HDMI monitor.
4. Plug a 5V 3A USB type-C power supply into BeagleBone® AI’s USB type-C port.
5. BeagleBone® AI will boot. No need to enter any passwords.
6. Depending on which software image is loaded, either a Desktop or a login shell will appear on the monitor.
7. Follow the instructions at https://beagleboard.org/upgrade

4.2.7 Wireless Connection

NOTE:Start with this image “am57xx-eMMC-flasher-debian-10.3-iot-tidl-armhf-2020-04-06-6gb.img.xz” loaded on your BeagleBone® AI.

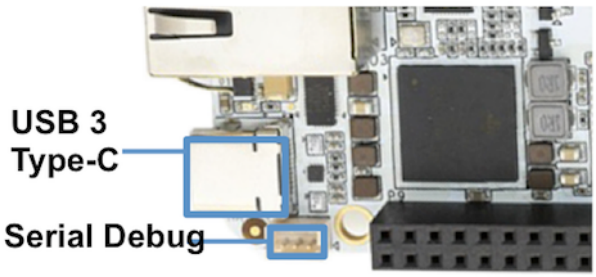
1. Plug a 5V 3A USB type-C power supply into BeagleBone® AI’s USB type-C port.
2. BeagleBone® AI will boot.
3. Connect your PC’s WiFi to SSID “BeagleBone-XXXX” where XXXX varies for your BeagleBone® AI.
4. Use password “BeagleBone” to complete the WiFi connection.
5. Open http://192.168.8.1 in your web browser.
6. Follow the instructions in the browser window.

4.2.8 Connecting a 3 PIN Serial Debug Cable

A 3 PIN serial debug cable can be helpful to debug when you need to view the boot messages through a terminal program such as putty on your host PC. This cable is not needed for most BeagleBone® AI boot up scenarios.

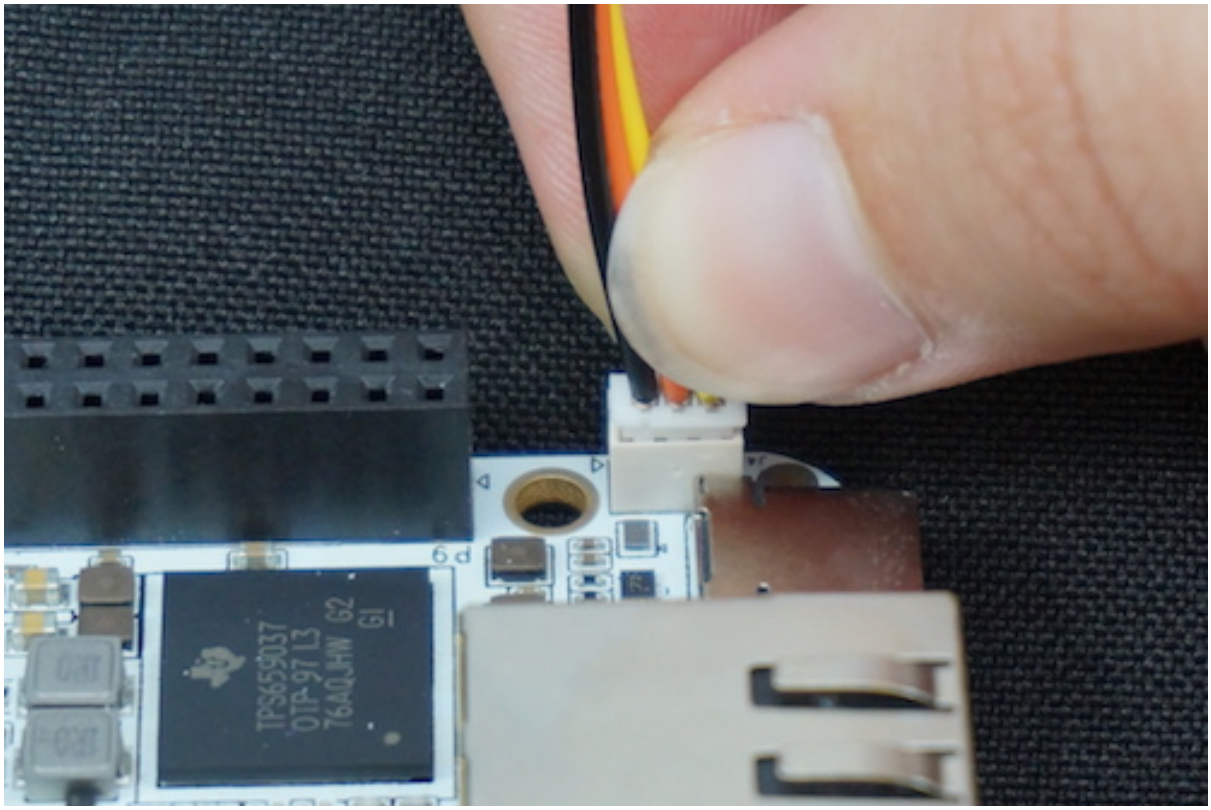
Cables: https://git.beagleboard.org/beagleboard/beaglebone-ai/-/wikis/Frequently-Asked-Questions#serial-cable

Locate the 3 PIN debug header on BeagleBone® AI, near the USB C connection.

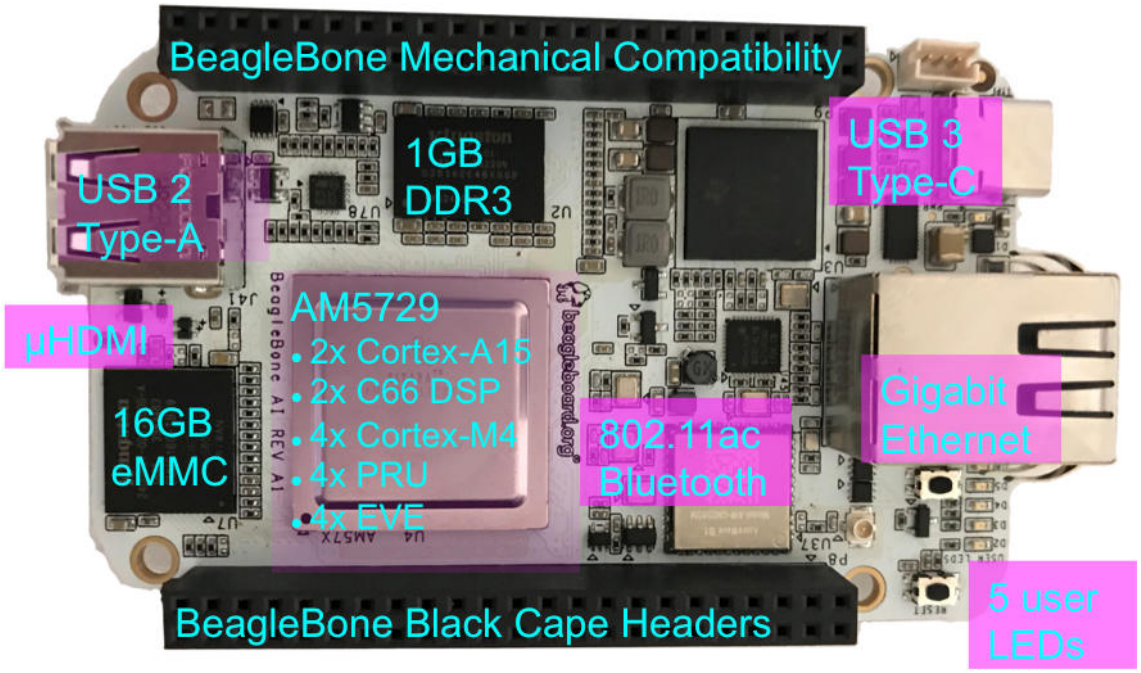


Press the small white connector into the 3 PIN debug header. The pinout is:

- Pin 1 (the pin closest to the screw-hole in the board. It is also marked with a shape on the silkscreen): GND
- Pin 2: UART1_RX (i.e. this is a BB-AI input pin)
- Pin 3: UART1_TX (i.e. BB-AI transmits out on this pin)



4.3 BeagleBone AI Overview



4.3.1 BeagleBone® AI Features

Main Processor Features of the AM5729 Within BeagleBone® AI

- Dual 1.5GHz ARM® Cortex®-A15 with out-of-order speculative issue 3-way superscalar execution pipeline for the fastest execution of existing 32-bit code
- 2 C66x Floating-Point VLIW DSP supported by OpenCL
- 4 Embedded Vision Engines (EVEs) supported by TIDL machine learning library
- 2x Dual-Core Programmable Real-Time Unit (PRU) subsystems (4 PRUs total) for ultra low-latency control and software generated peripherals
- 2x Dual ARM® Cortex®-M4 co-processors for real-time control
- IVA-HD subsystem with support for 4K @ 15fps H.264 encode/decode and other codecs @ 1080p60
- Vivante® GC320 2D graphics accelerator
- Dual-Core PowerVR® SGX544™ 3D GPU

Communications

- BeagleBone Black header and mechanical compatibility
- 16-bit LCD interfaces
- 4+ UARTs
- 2 I2C ports
- 2 SPI ports
- Lots of PRU I/O pins

Memory

- 1GB DDR3L
- 16GB on-board eMMC flash

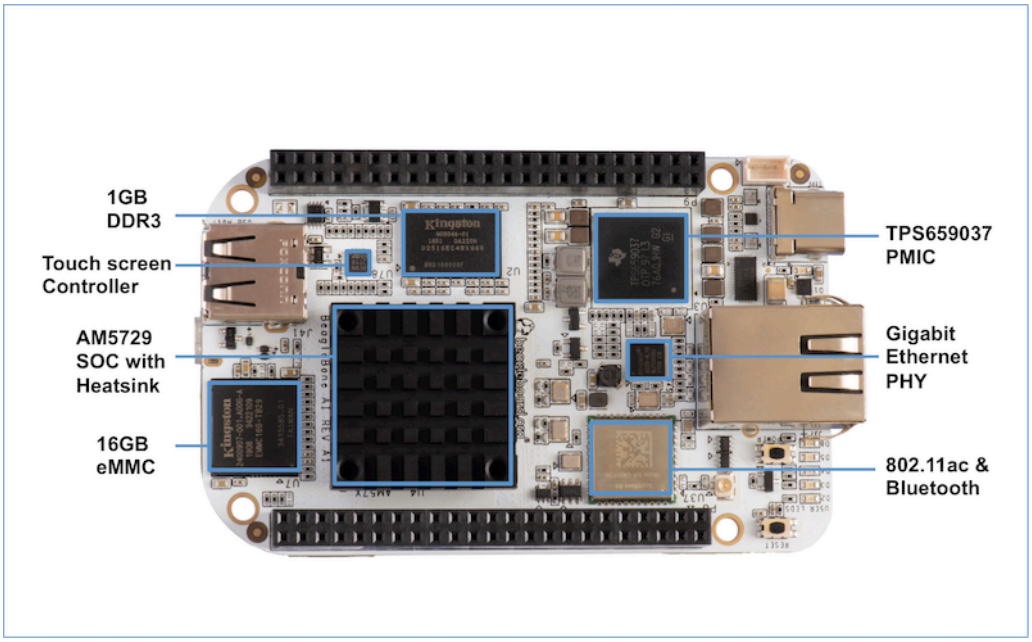
Connectors

- USB Type-C connector for power and SuperSpeed dual-role controller
- Gigabit Ethernet
- 802.11ac 2.4/5GHz WiFi via the AzureWave AW-CM256SM

Out of Box Software

- Zero-download out of box software environment

4.3.2 Board Component Locations

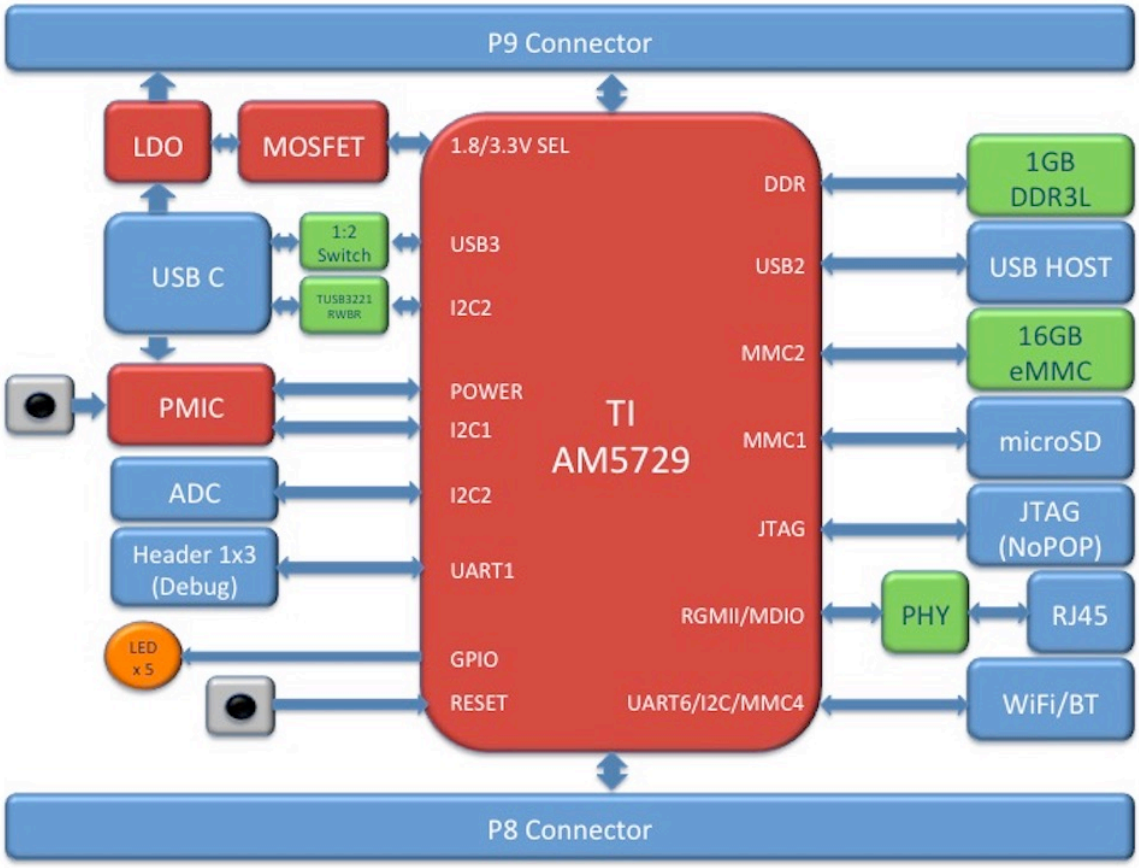


4.4 BeagleBone AI High Level Specification

This section provides the high level specification of BeagleBone® AI

4.4.1 Block Diagram

The figure below is the high level block diagram of BeagleBone® AI. For detailed layout information please check the schematics.

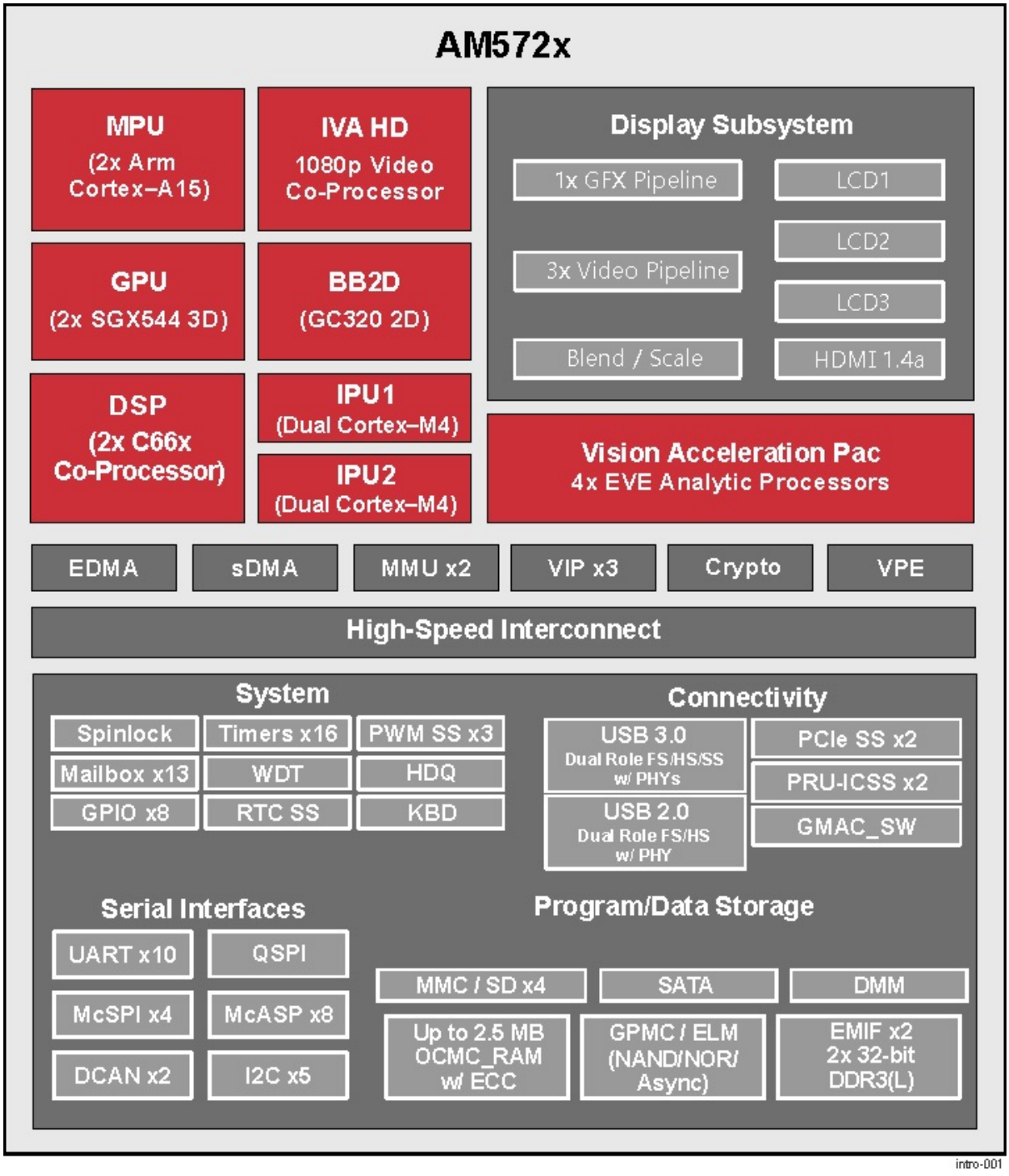


4.4.2 AM572x Sitara™ Processor

The Texas Instruments AM572x Sitara™ processor family of SOC devices brings high processing performance through the maximum flexibility of a fully integrated mixed processor solution. The devices also combine programmable video processing with a highly integrated peripheral set ideal for AI applications. The AM5729 used on BeagleBone® AI is the super-set device of the family.

Programmability is provided by dual-core ARM® Cortex®-A15 RISC CPUs with Arm® Neon™ extension, and two TI C66x VLIW floating-point DSP core, and Vision AccelerationPac (with 4x EVEs). The Arm allows developers to keep control functions separate from other algorithms programmed on the DSPs and coprocessors, thus reducing the complexity of the system software.

Texas Instruments AM572x Sitara™ Processor Family Block Diagram*



intro-001

MPU Subsystem The Dual Cortex-A15 MPU subsystem integrates the following submodules:

- ARM Cortex-A15 MPCore
  - Two central processing units (CPUs)
  - ARM Version 7 ISA: Standard ARM instruction set plus Thumb®-2, Jazelle® RCT Java™ accelerator, hardware virtualization support, and large physical address extensions (LPAE)
  - Neon™ SIMD coprocessor and VFPv4 per CPU
  - Interrupt controller with up to 160 interrupt requests
  - One general-purpose timer and one watchdog timer per CPU – Debug and trace features
  - 32-KiB instruction and 32-KiB data level 1 (L1) cache per CPU
- Shared 2-MiB level 2 (L2) cache

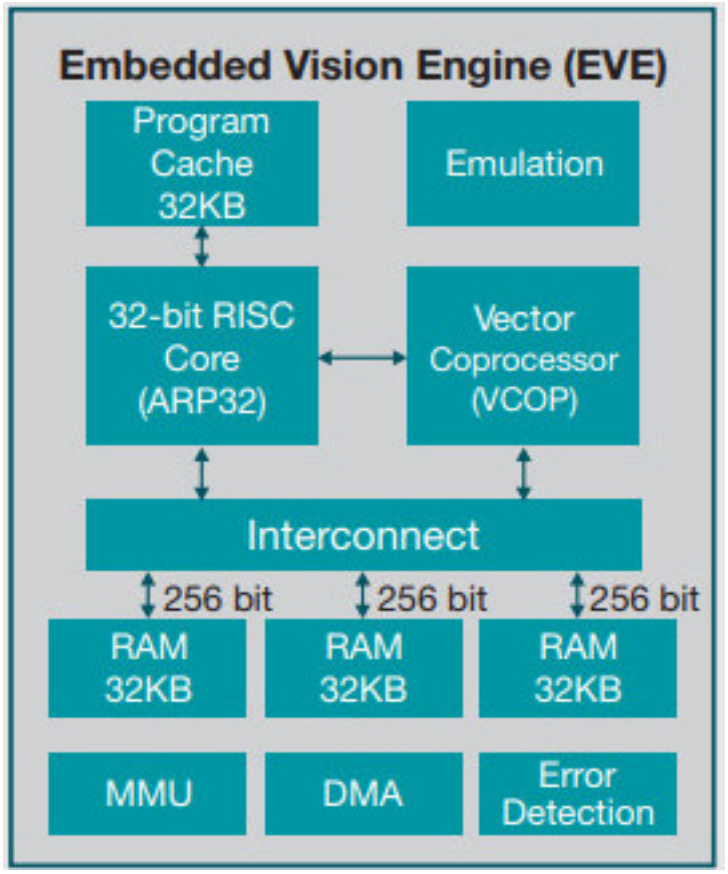
- 48-KiB bootable ROM
- Local power, reset, and clock management (PRCM) module
- Emulation features
- Digital phase-locked loop (DPLL)

DSP Subsystems There are two DSP subsystems in the device. Each DSP subsystem contains the following submodules:

- TMS320C66x™ Floating-Point VLIW DSP core for audio processing, and general-purpose imaging and video processing. It extends the performance of existing C64x+™ and C647x™ DSPs through enhancements and new features.
  - 32-KiB L1D and 32-KiB L1P cache or addressable SRAM
  - 288-KiB L2 cache
- 256-KiB configurable as cache or SRAM
- 32-KiB SRAM
- Enhanced direct memory access (EDMA) engine for video and audio data transfer
- Memory management units (MMU) for address management.
- Interrupt controller (INTC)
- Emulation capabilities
- Supported by OpenCL

EVE Subsystems

- 4 Embedded Vision Engines (EVEs) supported by TIDL machine learning library



The Embedded Vision Engine (EVE) module is a programmable imaging and vision processing engine. Software support for the EVE module is available through OpenCL Custom Device model with fixed set of functions. More information is available http://www.ti.com/lit/wp/spry251/spry251.pdf

PRU-ICSS Subsystems

- 2x Dual-Core Programmable Real-Time Unit (PRU) subsystems (4 PRUs total) for ultra low-latency control and software generated peripherals. Access to these powerful subsystems is available through through the P8 and P9 headers. These are detailed in Section 7.

IPU Subsystems There are two Dual Cortex-M4 IPU subsystems in the device available for general purpose usage, particularly real-time control. Each IPU subsystem includes the following components:

- Two Cortex-M4 CPUs
- ARMv7E-M and Thumb-2 instruction set architectures
- Hardware division and single-cycle multiplication acceleration
- Dedicated INTC with up to 63 physical interrupt events with 16-level priority
- Two-level memory subsystem hierarchy
  - L1 (32-KiB shared cache memory)
  - L2 ROM + RAM
- 64-KiB RAM
- 16-KiB bootable ROM
- MMU for address translation
- Integrated power management
- Emulation feature embedded in the Cortex-M4

IVA-HD Subsystem

- IVA-HD subsystem with support for 4K @ 15fps H.264 encode/decode and other codecs @ 1080p60 The IVA-HD subsystem is a set of video encoder and decoder hardware accelerators. The list of supported codecs can be found in the software development kit (SDK) documentation.

BB2D Graphics Accelerator Subsystem The Vivante® GC320 2D graphics accelerator is the 2D BitBlt (BB2D) graphics accelerator subsystem on the device with the following features:

- API support:
  - OpenWF™, DirectFB
  - GDI/DirectDraw
- BB2D architecture:
  - BitBlt and StretchBlt
  - DirectFB hardware acceleration
  - ROP2, ROP3, ROP4 full alpha blending and transparency
  - Clipping rectangle support
  - Alpha blending includes Java 2 Porter-Duff compositing rules
  - 90-, 180-, 270-degree rotation on every primitive
  - YUV-to-RGB color space conversion
  - Programmable display format conversion with 14 source and 7 destination formats
  - High-quality, 9-tap, 32-phase filter for image and video scaling at 1080p
  - Monochrome expansion for text rendering
  - 32K × 32K coordinate system

Dual-Core PowerVR® SGX544™ 3D GPU The 3D graphics processing unit (GPU) subsystem is based on POWERVR® SGX544 subsystem from Imagination Technologies. It supports general embedded applications. The GPU can process different data types simultaneously, such as: pixel data, vertex data, video data, and general-purpose data. The GPU subsystem has the following features:

- Multicore GPU architecture: two SGX544 cores.
- Shared system level cache of 128 KiB
- Tile-based deferred rendering architecture
- Second-generation universal scalable shader engines (USSE2), multithreaded engines incorporating pixel and vertex shader functionality
- Present and texture load accelerators
  - Enables to move, rotate, twiddle, and scale texture surfaces.
  - Supports RGB, ARGB, YUV422, and YUV420 surface formats.
  - Supports bilinear upscale.
  - Supports source colorkey.
- Fine-grained task switching, load balancing, and power management
- Programmable high-quality image antialiasing
- Bilinear, trilinear, anisotropic texture filtering
- Advanced geometry DMA driven operation for minimum CPU interaction
- Fully virtualized memory addressing for OS operation in a unified memory architecture (MMU)

4.4.3 Memory

1GB DDR3L

Dual 256M x 16 DDR3L memory devices are used, one on each side of the board, for a total of 1 GB. They will each operate at a clock frequency of up to 533 MHz yielding an effective rate of 1066Mb/s on the DDR3L bus allowing for 4GB/s of DDR3L memory bandwidth.

16GB Embedded MMC

A single 16GB embedded MMC (eMMC) device is on the board.

microSD Connector

The board is equipped with a single microSD connector to act as a secondary boot source for the board and, if selected as such, can be the primary booth source. The connector will support larger capacity microSD cards. The microSD card is not provided with the board.

4.4.4 Boot Modes

Todo: Need info on BBAI boot mode settings

4.4.5 Power Management

Todo: Need info on BBAI power management

4.4.6 Connectivity

Todo: Add WiFi/Bluetooth/Ethernet

BeagleBone® AI supports the majority of the functions of the AM5729 SOC through connectors or expansion header pin accessibility. See section 7 for more information on expansion header pinouts. There are a few functions that are not accessible which are: (TBD)

Todo: This text needs to go somewhere.

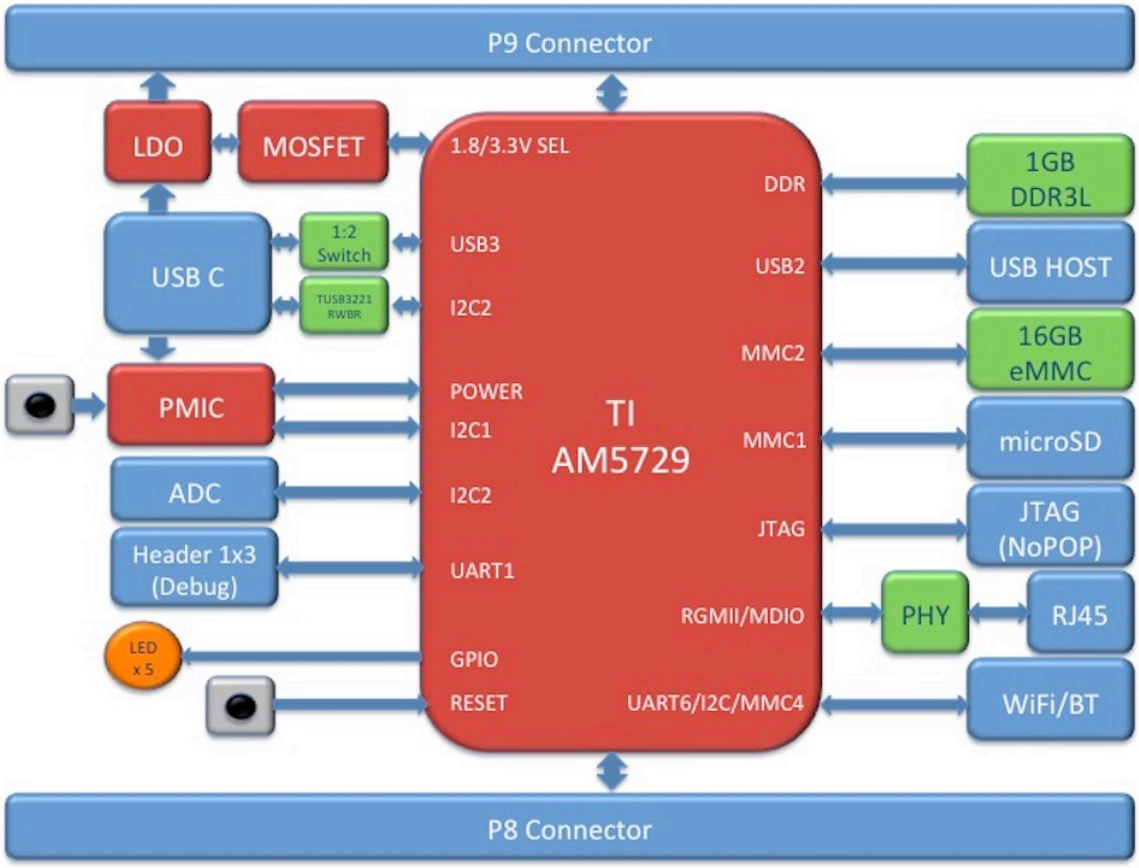
Table 4.1: On-board I2C Devices

Address	Identifier	Description
0x12	U3	TPS6590379 PMIC DVS
0x41	U78	STMPE811Q ADC and GPIO expander
0x47	U13	HD3SS3220 USB Type-C DRP port controller
0x50	U9	24LC32 board ID EEPROM
0x58	U3	TPS6590379 PMIC power registers
0x5a	U3	TPS6590379 PMIC interfaces and auxiliaries
0x5c	U3	TPS6590379 PMIC trimming and test
0x5e	U3	TPS6590379 PMIC OTP

4.5 Detailed Hardware Design

This section provides a detailed description of the Hardware design. This can be useful for interfacing, writing drivers, or using it to help modify specifics of your own design.

The figure below is the high level block diagram of BeagleBone® AI. For those who may be concerned, this is the same figure found in section 5. It is placed here again for convenience so it is closer to the topics to follow.



4.5.1 Power Section

Figure ? is the high level block diagram of the power section of the board.

(Block Diagram for Power)

TPS6590379 PMIC

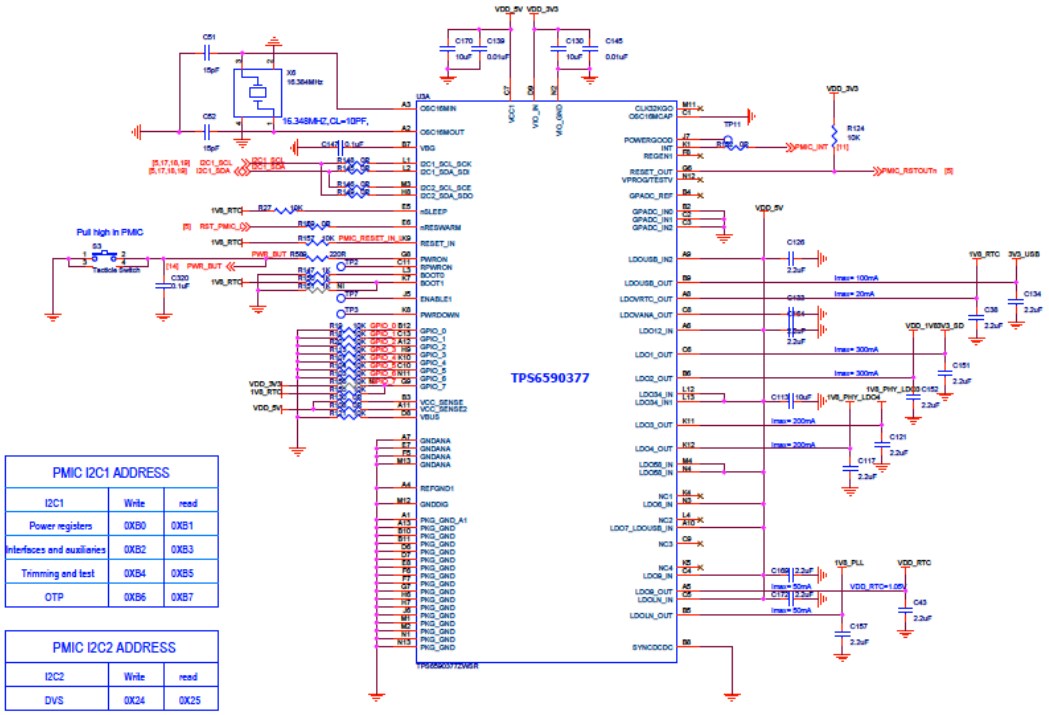
The Texas Instruments TPS6590379ZWSR device is an integrated power-management IC (PMIC) specifically designed to work well ARM Cortex A15 Processors, such as the AM5729 used on BeagleBone® AI. The datasheet is located here https://www.ti.com/lit/ds/symlink/tps659037.pdf

The device provides seven configurable step-down converters with up to 6 A of output current for memory, processor core, input-output (I/O), or preregulation of LDOs. One of these configurable step-down converters can be combined with another 3-A regulator to allow up to 9 A of output current. All of the step-down converters can synchronize to an external clock source between 1.7 MHz and 2.7 MHz, or an internal fallback clock at 2.2 MHz.

The TPS659037 device contains seven LDO regulators for external use. These LDO regulators can be supplied from either a system supply or a preregulated supply. The power-up and power-down controller is configurable and supports any power-up and power-down sequences (OTP based). The TPS659037 device includes a 32-kHz RC oscillator to sequence all resources during power up and power down. In cases where a fast start up is needed, a 16-MHz crystal oscillator is also included to quickly generate a stable 32-kHz for the system. All LDOs and SMPS converters can be controlled by the SPI or I2C interface, or by power request signals. In addition, voltage scaling registers allow transitioning the SMPS to different voltages by SPI, I2C, or roof and floor control.

One dedicated pin in each package can be configured as part of the power-up sequence to control external resources. General-purpose input-output (GPIO) functionality is available and two GPIOs can be configured as part of the power-up sequence to control external resources. Power request signals enable power mode

control for power optimization. The device includes a general-purpose sigma-delta analog-to-digital converter (GPADC) with three external input channels.



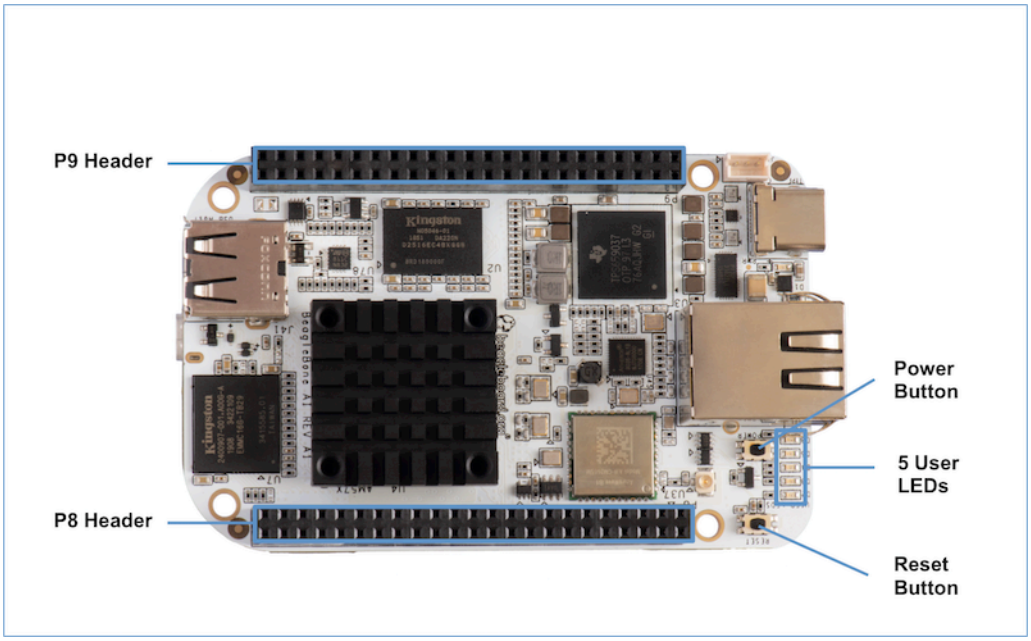
PMIC I2C1 ADDRESS		
I2C1	Write	read
Power registers	0XB0	0XB1
Interfaces and auxiliaries	0XB2	0XB3
Trimming and test	0XB4	0XB5
OTP	0XB6	0XB7

PMIC I2C2 ADDRESS		
I2C2	Write	read
DVS	0X24	0X25

USB-C Power

Below image shows how the USB-C power input is connected to the **TPS6590379**.

Power Button



4.5.2 eMMC Flash Memory (16GB)

eMMC Device

eMMC Circuit Design

Board ID

A board identifier is placed on the eMMC in the second linear boot partition (/dev/mmcblk1boot1). Reserved bytes up to 32k (0x8000) are filled with "FF".

Table 4.2: Board ID

Name	Size (bytes)	Contents
Header	4	MSB 0xEE3355AA LSB (stored LSB first)
Board Name	8	Name for board in ASCII "BBONE-AI" = BeagleBone AI
Version	4	Hardware version code for board in ASCII "00A1" = rev. A1
Serial Number	14	Serial number of the board. This is a 14 character string which is: WWYYEMAInnnnnn where: • WW = 2 digit week of the year of production • YY = 2 digit year of production • EM = Embest • AI = BeagleBone AI • nnnnnn = incrementing board number

```
debian@beaglebone:~$ sudo hexdump -C /dev/mmcblk1boot1
00000000  aa 55 33 ee 42 42 4f 4e  45 2d 41 49 30 30 41 31  |.U3.BBONE-
↪AI00A1|
00000010  31 39 33 33 45 4d 41 49  30 30 30 38 30 33 ff ff  |1933EMAI000803..
↪|
00000020  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................
↪|
*
00008000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................
↪|
*
00400000
```

4.5.3 Wireless Communication: 802.11 ac & Bluetooth: AzureWave AW-CM256SM

Datasheet https://storage.googleapis.com/wzukusers/user-26561200/documents/5b7d0fe3c3f29Ct6k0QI/AW-CM256SM_DS_Rev%2015_CYW.pdf Wireless connectivity is provided on BeagleBone® AI via the AzureWave Technologies AW-CM256SM IEEE 802.11a/b/g/n/ac Wi-Fi with Bluetooth 4.2 Combo Stamp Module.

This highly integrated wireless local area network (WLAN) solution combines Bluetooth 4.2 and provides a complete 2.4GHz Bluetooth system which is fully compliant to Bluetooth 4.2 and v2.1 that supports EDR of 2Mbps and 3Mbps for data and audio communications. It enables a high performance, cost effective, low power, compact solution that easily fits onto the SDIO and UART combo stamp module.

Compliant with the IEEE 802.11a/b/g/n/ac standard, AW-CM256SM uses Direct Sequence Spread Spectrum (DSSS), Orthogonal Frequency Division Multiplexing (OFDM), BPSK, QPSK, CCK and QAM baseband modulation technologies. Compare to 802.11n technology, 802.11ac provides a big improvement on speed and range.

The AW-CM256SM module adopts a Cypress solution. The module design is based on the Cypress CYP43455 single chip.

WLAN on the AzureWave AW-CM256SM

High speed wireless connection up to 433.3Mbps transmit/receive PHY rate using 80MHz bandwidth,

- 1 antennas to support 1(Transmit) and 1(Receive) technology and Bluetooth
- WCS (Wireless Coexistence System)
- Low power consumption and high performance
- Enhanced wireless security
- Fully speed operation with Piconet and Scatternet support
- 12mm(L) x 12mm(W) x1.65mm(H) LGA package
- Dual - band 2.4 GHz and 5GHz 802.11 a/b/g/n/ac
- External Crystal

Bluetooth on the AzureWave AW-CM256S

- 1 antennas to support 1(Transmit) and 1(Receive) technology and Bluetooth
- Fully qualified Bluetooth BT4.2
- Enhanced Data Rate(EDR) compliant for both 2Mbps and 3Mbps supported
- High speed UART and PCM for Bluetooth

4.5.4 HDMI

The HDMI interface is aligned with the HDMI TMDS single stream standard v1.4a (720p @60Hz to 1080p @24Hz) and the HDMI v1.3 (1080p @60Hz): 3 data channels, plus 1 clock channel is supported (differential).

TODO: Verify it isn't better than this. Doesn't seem right.

4.5.5 PRU-ICSS

The Texas Instruments AM5729 Sitara™ provides 2 Programmable Real-Time Unit Subsystem and Industrial Communciation Subsystems. (PRU-ICSS1 and PRU-ICSS2).

Within each PRU-ICSS are dual 32-bit Load / Store RISC CPU cores: Programmable Real-Time Units (PRU0 and PRU1), shared data and instruction memories, internal peripheral modules and an interrupt controller. Therefore the SoC is providing a total of 4 PRU 32-bit RISC CPU's:

- PRU-ICSS1 PRU0
- PRU-ICSS1 PRU1
- PRU-ICSS2 PRU0
- PRU-ICSS2 PRU1

The programmable nature of the PRUs, along with their access to pins, events and all SoC resources, provides flexibility in implementing fast real-time responses, specialized data handling operations, peripheral interfaces and in off-loading tasks from the other processor cores of the SoC.

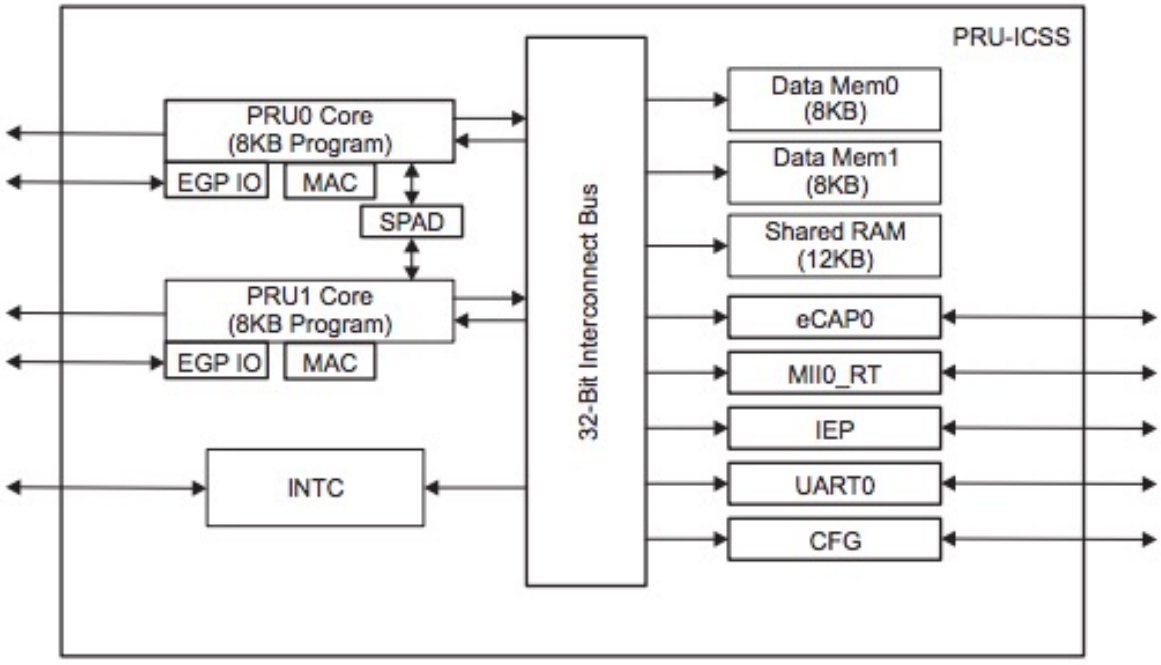
PRU-ICSS Features

Each of the 2 PRU-ICSS (PRU-ICSS1 and PRU-ICSS2) includes the following main features:

- 2 Independent programmable real-time (PRU) cores (PRU0 and PRU1)
- 21x Enhanced GPIs (EGPIs) and 21x Enhanced GPOs (EGPOs) with asynchronous capture and serial support per each PRU CPU core
- One Ethernet MII_RT module (PRU-ICSS_MII_RT) with two MII ports and configurable connections to PRUs
- 1 MDIO Port (PRU-ICSS_MII_MDIO)
- One Industrial Ethernet Peripheral (IEP) to manage/generate Industrial Ethernet functions
- 1 x 16550-compatible UART with a dedicated 192 MHz clock to support 12Mbps Profibus
- 1 Industrial Ethernet timer with 7/9 capture and 8 compare events
- 1 Enhanced Capture Module (ECAP)
- 1 Interrupt Controller (PRU-ICSS_INTC)
- A flexible power management support
- Integrated switched central resource with programmable priority
- Parity control supported by all memories

PRU-ICSS Block Diagram

Below is a high level block diagram of one of the PRU-ICSS Subsystems



4.5.6 PRU-ICSS Resources and FAQ's

Resources

- Great resources for PRU and BeagleBone® has been compiled here https://beagleboard.org/pru
- The PRU Cookbook provides examples and getting started information *PRU Cookbook*
- Detailed specification is available at http://processors.wiki.ti.com/index.php/PRU-ICSS

FAQ

- Q: Is it possible to configure the Ethernet MII to be accessed via a PRU MII?
- A: TBD

PRU-ICSS1 Pin Access

The table below shows which PRU-ICSS1 signals can be accessed on BeagleBone® AI and on which connector and pins they are accessible from. Some signals are accessible on the same pins. Signal Names reveal which PRU-ICSS Subsystem is being addressed. pr1 is PRU-ICSS1 and pr2 is PRU-ICSS2

Table 4.3: PRU-ICSS1 Pin Access

SIGNAL NAME	DESCRIPTION	TYPE	PROC	HEADER_PIN	MODE	HEADER_PIN	MODE
pr1_pru0_gpo0	PRU0 General-Purpose Output	O	AH6	NA			
pr1_pru0_gpo1	PRU0 General-Purpose Output	O	AH3	NA			
pr1_pru0_gpo2	PRU0 General-Purpose Output	O	AH5	NA			
pr1_pru0_gpo3	PRU0 General-Purpose Output	O	AG6	P8_12	MODE13		
pr1_pru0_gpo4	PRU0 General-Purpose Output	O	AH4	P8_11	MODE13		
pr1_pru0_gpo5	PRU0 General-Purpose Output	O	AG4	P9_15	MODE13		
pr1_pru0_gpo6	PRU0 General-Purpose Output	O	AG2	NA			
pr1_pru0_gpo7	PRU0 General-Purpose Output	O	AG3	NA			
pr1_pru0_gpo8	PRU0 General-Purpose Output	O	AG5	NA			
pr1_pru0_gpo9	PRU0 General-Purpose Output	O	AF2	NA			
pr1_pru0_gpo10	PRU0 General-Purpose Output	O	AF6	NA			
pr1_pru0_gpo11	PRU0 General-Purpose Output	O	AF3	NA			
pr1_pru0_gpo12	PRU0 General-Purpose Output	O	AF4	NA			
pr1_pru0_gpo13	PRU0 General-Purpose Output	O	AF1	NA			
pr1_pru0_gpo14	PRU0 General-Purpose Output	O	AE3	NA			
pr1_pru0_gpo15	PRU0 General-Purpose Output	O	AE5	NA			
pr1_pru0_gpo16	PRU0 General-Purpose Output	O	AE1	NA			
pr1_pru0_gpo17	PRU0 General-Purpose Output	O	AE2	P9_26	MODE13		
pr1_pru0_gpo18	PRU0 General-Purpose Output	O	AE6	NA			
pr1_pru0_gpo19	PRU0 General-Purpose Output	O	AD2	NA			
pr1_pru0_gpo20	PRU0 General-Purpose Output	O	AD3	NA			
pr1_pru0_gpi0	PRU0 General-Purpose Input	I	AH6	NA			
pr1_pru0_gpi1	PRU0 General-Purpose Input	I	AH3	NA			
pr1_pru0_gpi2	PRU0 General-Purpose Input	I	AH5	NA			
pr1_pru0_gpi3	PRU0 General-Purpose Input	I	AG6	P8_12	MODE12		
pr1_pru0_gpi4	PRU0 General-Purpose Input	I	AH4	P8_11	MODE12		
pr1_pru0_gpi5	PRU0 General-Purpose Input	I	AG4	P9_15	MODE12		
pr1_pru0_gpi6	PRU0 General-Purpose Input	I	AG2	NA			
pr1_pru0_gpi7	PRU0 General-Purpose Input	I	AG3	NA			
pr1_pru0_gpi8	PRU0 General-Purpose Input	I	AG5	NA			
pr1_pru0_gpi9	PRU0 General-Purpose Input	I	AF2	NA			
pr1_pru0_gpi10	PRU0 General-Purpose Input	I	AF6	NA			
pr1_pru0_gpi11	PRU0 General-Purpose Input	I	AF3	NA			
pr1_pru0_gpi12	PRU0 General-Purpose Input	I	AF4	NA			
pr1_pru0_gpi13	PRU0 General-Purpose Input	I	AF1	NA			

continues on next page

Table 4.3 – continued from previous page

SIGNAL NAME	DESCRIPTION	TYPE	PROC	HEADER_PIN	MODE	HEADER_PIN	MODE
pr1_pru0_gpi14	PRU0 General-Purpose Input	I	AE3	NA			
pr1_pru0_gpi15	PRU0 General-Purpose Input	I	AE5	NA			
pr1_pru0_gpi16	PRU0 General-Purpose Input	I	AE1	NA			
pr1_pru0_gpi17	PRU0 General-Purpose Input	I	AE2	P9_26	MODE12		
pr1_pru0_gpi18	PRU0 General-Purpose Input	I	AE6	NA			
pr1_pru0_gpi19	PRU0 General-Purpose Input	I	AD2	NA			
pr1_pru0_gpi20	PRU0 General-Purpose Input	I	AD3	NA			
pr1_pru1_gpo0	PRU1 General-Purpose Output	O	E2	NA			
pr1_pru1_gpo1	PRU1 General-Purpose Output	O	D2	P9_20	MODE13		
pr1_pru1_gpo2	PRU1 General-Purpose Output	O	F4	P9_19	MODE13		
pr1_pru1_gpo3	PRU1 General-Purpose Output	O	C1	P9_41	MODE13		
pr1_pru1_gpo4	PRU1 General-Purpose Output	O	E4	NA			
pr1_pru1_gpo5	PRU1 General-Purpose Output	O	F5	P8_18	MODE13		
pr1_pru1_gpo6	PRU1 General-Purpose Output	O	E6	P8_19	MODE13		
pr1_pru1_gpo7	PRU1 General-Purpose Output	O	D3	P8_13	MODE13		
pr1_pru1_gpo8	PRU1 General-Purpose Output	O	F6	NA			
pr1_pru1_gpo9	PRU1 General-Purpose Output	O	D5	P8_14	MODE13		
pr1_pru1_gpo10	PRU1 General-Purpose Output	O	C2	P9_42	MODE13		
pr1_pru1_gpo11	PRU1 General-Purpose Output	O	C3	P9_27	MODE13		
pr1_pru1_gpo12	PRU1 General-Purpose Output	O	C4	NA			
pr1_pru1_gpo13	PRU1 General-Purpose Output	O	B2	NA			
pr1_pru1_gpo14	PRU1 General-Purpose Output	O	D6	P9_14	MODE13		
pr1_pru1_gpo15	PRU1 General-Purpose Output	O	C5	P9_16	MODE13		
pr1_pru1_gpo16	PRU1 General-Purpose Output	O	A3	P8_15	MODE13		
pr1_pru1_gpo17	PRU1 General-Purpose Output	O	B3	P8_26	MODE13		
pr1_pru1_gpo18	PRU1 General-Purpose Output	O	B4	P8_16	MODE13		
pr1_pru1_gpo19	PRU1 General-Purpose Output	O	B5	NA			
pr1_pru1_gpo20	PRU1 General-Purpose Output	O	A4	NA			
pr1_pru1_gpi0	PRU1 General-Purpose Input	I	E2	NA			
pr1_pru1_gpi1	PRU1 General-Purpose Input	I	D2	P9_20	MODE12		
pr1_pru1_gpi2	PRU1 General-Purpose Input	I	F4	P9_19	MODE12		
pr1_pru1_gpi3	PRU1 General-Purpose Input	I	C1	P9_41	MODE12		
pr1_pru1_gpi4	PRU1 General-Purpose Input	I	E4	NA			
pr1_pru1_gpi5	PRU1 General-Purpose Input	I	F5	P8_18	MODE12		
pr1_pru1_gpi6	PRU1 General-Purpose Input	I	E6	P8_19	MODE12		
pr1_pru1_gpi7	PRU1 General-Purpose Input	I	D3	P8_13	MODE12		
pr1_pru1_gpi8	PRU1 General-Purpose Input	I	F6	NA			

continues on next page

Table 4.3 – continued from previous page

SIGNAL NAME	DESCRIPTION	TYPE	PROC	HEADER_PIN	MODE	HEADER_PIN	MODE
pr1_pru1_gpi9	PRU1 General-Purpose Input	I	D5	P8_14	MODE12		
pr1_pru1_gpi10	PRU1 General-Purpose Input	I	C2	P9_42	MODE12		
pr1_pru1_gpi11	PRU1 General-Purpose Input	I	C3	P9_27	MODE12		
pr1_pru1_gpi12	PRU1 General-Purpose Input	I	C4	NA			
pr1_pru1_gpi13	PRU1 General-Purpose Input	I	B2	NA			
pr1_pru1_gpi14	PRU1 General-Purpose Input	I	D6	P9_14	MODE12		
pr1_pru1_gpi15	PRU1 General-Purpose Input	I	C5	P9_16	MODE12		
pr1_pru1_gpi16	PRU1 General-Purpose Input	I	A3	P8_15	MODE12		
pr1_pru1_gpi17	PRU1 General-Purpose Input	I	B3	P8_26	MODE12		
pr1_pru1_gpi18	PRU1 General-Purpose Input	I	B4	P8_16	MODE12		
pr1_pru1_gpi19	PRU1 General-Purpose Input	I	B5	NA			
pr1_pru1_gpi20	PRU1 General-Purpose Input	I	A4	NA			
pr1_mii_mt0_clk	MII0 Transmit Clock	I	U5	NA			
pr1_mii0_txen	MII0 Transmit Enable	O	V3	NA			
pr1_mii0_txd3	MII0 Transmit Data	O	V5	NA			
pr1_mii0_txd2	MII0 Transmit Data	O	V4	NA			
pr1_mii0_txd1	MII0 Transmit Data	O	Y2	NA			
pr1_mii0_txd0	MII0 Transmit Data	O	W2	NA			
pr1_mii0_rxdv	MII0 Data Valid	I	V2	NA			
pr1_mii_mr0_clk	MII0 Receive Clock	I	Y1	NA			
pr1_mii0_rxd3	MII0 Receive Data	I	W9	NA			
pr1_mii0_rxd2	MII0 Receive Data	I	V9	NA			
pr1_mii0_crs	MII0 Carrier Sense	I	V7	NA			
pr1_mii0_rxer	MII0 Receive Error	I	U7	NA			
pr1_mii0_rxd1	MII0 Receive Data	I	V6	NA			
pr1_mii0_rxd0	MII0 Receive Data	I	U6	NA			
pr1_mii0_col	MII0 Collision Detect	I	V1	NA			
pr1_mii0_rxlink	MII0 Receive Link	I	U4	NA			
pr1_mii_mt1_clk	MII1 Transmit Clock	I	C1	P9_41	MODE11		
pr1_mii1_txen	MII1 Transmit Enable	O	E4	NA			
pr1_mii1_txd3	MII1 Transmit Data	O	F5	P8_18	MODE11		
pr1_mii1_txd2	MII1 Transmit Data	O	E6	P8_19	MODE11		
pr1_mii1_txd1	MII1 Transmit Data	O	D5	P8_14	MODE11		
pr1_mii1_txd0	MII1 Transmit Data	O	C2	P9_42	MODE11		
pr1_mii_mr1_clk	MII1 Receive Clock	I	C3	P9_27	MODE11		
pr1_mii1_rxdv	MII1 Data Valid	I	C4	NA			
pr1_mii1_rxd3	MII1 Receive Data	I	B2	NA			

continues on next page

Table 4.3 – continued from previous page

SIGNAL NAME	DESCRIPTION	TYPE	PROC	HEADER_PIN	MODE	HEADER_PIN	MODE
pr1_mii1_rxd2	MII1 Receive Data	I	D6	P9_14	MODE11		
pr1_mii1_rxd1	MII1 Receive Data	I	C5	P9_16	MODE11		
pr1_mii1_rxd0	MII1 Receive Data	I	A3	P8_15	MODE11		
pr1_mii1_rxer	MII1 Receive Error	I	B3	P8_26	MODE11		
pr1_mii1_rxlink	MII1 Receive Link	I	B4	P8_16	MODE11		
pr1_mii1_col	MII1 Collision Detect	I	B5	NA			
pr1_mii1_crs	MII1 Carrier Sense	I	A4	NA			
pr1_mdio_mdclk	MDIO Clock	O	D3	P8_13	MODE11		
pr1_mdio_data	MDIO Data	IO	F6	NA			
pr1_edc_latch0_in	Latch Input 0	I	AG3/E2	NA			
pr1_edc_latch1_in	Latch Input 1	I	AG5	NA			
pr1_edc_sync0_out	SYNC0 Output	O	AF2/D2	P9_20	MODE11		
pr1_edc_sync1_out	SYNC1 Output	O	AF6	NA			
pr1_edio_latch_in	Latch Input	I	AF3	NA			
pr1_edio_sof	Start Of Frame	O	AF4/F4	P9_19	MODE11		
pr1_edio_data_in0	Ethernet Digital Input	I	AF1/E1	NA			
pr1_edio_data_in1	Ethernet Digital Input	I	AE3/G2	NA			
pr1_edio_data_in2	Ethernet Digital Input	I	AE5/H7	NA			
pr1_edio_data_in3	Ethernet Digital Input	I	AE1/G1	NA			
pr1_edio_data_in4	Ethernet Digital Input	I	AE2/G6	P9_26	MODE10	P8_34	MODE12
pr1_edio_data_in5	Ethernet Digital Input	I	AE6/F2	P8_36	MODE12		
pr1_edio_data_in6	Ethernet Digital Input	I	AD2/F3	NA			
pr1_edio_data_in7	Ethernet Digital Input	I	AD3/D1	P8_15	MODE12		
p r1_edio_data_out0	Ethernet Digital Output	O	AF1/E1	NA			
p r1_edio_data_out1	Ethernet Digital Output	O	AE3/G2	NA			
p r1_edio_data_out2	Ethernet Digital Output	O	AE5/H7	NA			
p r1_edio_data_out3	Ethernet Digital Output	O	AE1/G1	NA			
p r1_edio_data_out4	Ethernet Digital Output	O	AE2/G6	P9_26	MODE11	P8_34	MODE13
p r1_edio_data_out5	Ethernet Digital Output	O	AE6/F2	P8_36	MODE13		
p r1_edio_data_out6	Ethernet Digital Output	O	AD2/F3	NA			
p r1_edio_data_out7	Ethernet Digital Output	O	AD3/D1	P8_15	MODE13		
pr1_uart0_cts_n	UART Clear-To-Send	I	G1/F11	P8_45	MODE10		
pr1_uart0_rts_n	UART Ready-To-Send	O	G6/G10	P8_34	MODE11	P8_46	MODE10
pr1_uart0_rxd	UART Receive Data	I	F2/F10	P8_36	MODE11	P8_43	MODE10
pr1_uart0_txd	UART Transmit Data	O	F3/G11	P8_44	MODE10		
pr1_ecap0_ ecap_capin_apwm_o	Capture Input/PWM Output	IO	D1/E9	P8_15	MODE11	P8_41	MODE10

PRU-ICSS2 Pin Access

The table below shows which PRU-ICSS2 signals can be accessed on BeagleBone® AI and on which connector and pins they are accessible from. Some signals are accessible on the same pins. Signal Names reveal which PRU-ICSS Subsystem is being addressed. pr1 is PRU-ICSS1 and pr2 is PRU-ICSS2

Table 4.4: PRU-ICSS2 Pin Access

SIGNAL NAME	DESCR IPTION	TYPE	PROC	HEAD ER_PIN	MODE	HEAD ER_PIN	MODE
p r2_pru 0_gpo0	PRU0 Gen eral-P urpose Output	O	G 11/AC5	P8_44	MODE13		
p r2_pru 0_gpo1	PRU0 Gen eral-P urpose Output	O	E9/AB4	P8_41	MODE13		
p r2_pru 0_gpo2	PRU0 Gen eral-P urpose Output	O	F9/AD4	P8_42	MODE13	P8_21	MODE13
p r2_pru 0_gpo3	PRU0 Gen eral-P urpose Output	O	F8/AC4	P8_39	MODE13	P8_20	MODE13
p r2_pru 0_gpo4	PRU0 Gen eral-P urpose Output	O	E7/AC7	P8_40	MODE13	P8_25	MODE13
p r2_pru 0_gpo5	PRU0 Gen eral-P urpose Output	O	E8/AC6	P8_37	MODE13	P8_24	MODE13
p r2_pru 0_gpo6	PRU0 Gen eral-P urpose Output	O	D9/AC9	P8_38	MODE13	P8_5	MODE13
p r2_pru 0_gpo7	PRU0 Gen eral-P urpose Output	O	D7/AC3	P8_36	MODE13	P8_6	MODE13
p r2_pru 0_gpo8	PRU0 Gen eral-P urpose Output	O	D8/AC8	P8_34	MODE13	P8_23	MODE13
p r2_pru 0_gpo9	PRU0 Gen eral-P urpose Output	O	A5/AD6	P8_35	MODE13	P8_22	MODE13
pr 2_pru0 _gpo10	PRU0 Gen eral-P urpose Output	O	C6/AB8	P8_33	MODE13	P8_3	MODE13
pr 2_pru0 _gpo11	PRU0 Gen eral-P urpose Output	O	C8/AB5	P8_31	MODE13	P8_4	MODE13
pr 2_pru0 _gpo12	PRU0 Gen eral-P urpose Output	O	C7/B18	P8_32	MODE13		
pr 2_pru0 _gpo13	PRU0 Gen eral-P urpose Output	O	B7/F15	P8_45	MODE13		
pr 2_pru0 _gpo14	PRU0 Gen eral-P urpose Output	O	B8/B19	P9_11	MODE13	P9_11	MODE13
pr 2_pru0 _gpo15	PRU0 Gen eral-P urpose Output	O	A7/C17	P8_17	MODE13	P9_13	MODE13
pr 2_pru0 _gpo16	PRU0 Gen eral-P urpose Output	O	A8/C15	P8_27	MODE13		
pr 2_pru0 _gpo17	PRU0 Gen eral-P urpose Output	O	C9/A16	P8_28	MODE13		
pr 2_pru0 _gpo18	PRU0 Gen eral-P urpose Output	O	A9/A19	P8_29	MODE13		
pr 2_pru0 _gpo19	PRU0 Gen eral-P urpose Output	O	B9/A18	P8_30	MODE13		
pr 2_pru0 _gpo20	PRU0 Gen eral-P urpose Output	O	A 10/F14	P8_46	MODE13	P8_8	MODE13
p r2_pru 0_gpi0	PRU0 Gen eral-P urpose Input	I	G 11/AC5	P8_44	MODE12		
p r2_pru 0_gpi1	PRU0 Gen eral-P urpose Input	I	E9/AB4	P8_41	MODE12		
p r2_pru 0_gpi2	PRU0 Gen eral-P urpose Input	I	F9/AD4	P8_42	MODE12	P8_21	MODE12
p r2_pru 0_gpi3	PRU0 Gen eral-P urpose Input	I	F8/AC4	P8_39	MODE12	P8_20	MODE12
p r2_pru 0_gpi4	PRU0 Gen eral-P urpose Input	I	E7/AC7	P8_40	MODE12	P8_25	MODE12
p r2_pru 0_gpi5	PRU0 Gen eral-P urpose Input	I	E8/AC6	P8_37	MODE12	P8_24	MODE12
p r2_pru 0_gpi6	PRU0 Gen eral-P urpose Input	I	D9/AC9	P8_38	MODE12	P8_5	MODE12
p r2_pru 0_gpi7	PRU0 Gen eral-P urpose Input	I	D7/AC3	P8_36	MODE12	P8_6	MODE12
p r2_pru 0_gpi8	PRU0 Gen eral-P urpose Input	I	D8/AC8	P8_34	MODE12	P8_23	MODE12
p r2_pru 0_gpi9	PRU0 Gen eral-P urpose Input	I	A5/AD6	P8_35	MODE12	P8_22	MODE12
pr 2_pru0 _gpi10	PRU0 Gen eral-P urpose Input	I	C6/AB8	P8_33	MODE12	P8_3	MODE12
pr 2_pru0 _gpi11	PRU0 Gen eral-P urpose Input	I	C8/AB5	P8_31	MODE12	P8_4	MODE12
pr 2_pru0 _gpi12	PRU0 Gen eral-P urpose Input	I	C7/B18	P8_32	MODE12		
pr 2_pru0 _gpi13	PRU0 Gen eral-P urpose Input	I	B7/F15	P8_45	MODE12		

continues on next page

Table 4.4 – continued from previous page

SIGNAL NAME	DESCR IPTION	TYPE	PROC	HEAD ER_PIN	MODE	HEAD ER_PIN	MODE
pr 2_pru0 _gpi14	PRU0 Gen eral-P urpose Input	I	B8/B19	P9_11	MODE12	P9_11	MODE12
pr 2_pru0 _gpi15	PRU0 Gen eral-P urpose Input	I	A7/C17	P8_17	MODE12	P9_13	MODE12
pr 2_pru0 _gpi16	PRU0 Gen eral-P urpose Input	I	A8/C15	P8_27	MODE12		
pr 2_pru0 _gpi17	PRU0 Gen eral-P urpose Input	I	C9/A16	P8_28	MODE12		
pr 2_pru0 _gpi18	PRU0 Gen eral-P urpose Input	I	A9/A19	P8_29	MODE12		
pr 2_pru0 _gpi19	PRU0 Gen eral-P urpose Input	I	B9/A18	P8_30	MODE12		
pr 2_pru0 _gpi20	PRU0 Gen eral-P urpose Input	I	A 10/F14	P8_46	MODE12	P8_8	MODE12
p r2_pru 1_gpo0	PRU1 Gen eral-P urpose Output	O	V1/D17	P8_32	MODE13		
p r2_pru 1_gpo1	PRU1 Gen eral-P urpose Output	O	U4/AA3	NA			
p r2_pru 1_gpo2	PRU1 Gen eral-P urpose Output	O	U3/AB9	NA			
p r2_pru 1_gpo3	PRU1 Gen eral-P urpose Output	O	V2/AB3	NA			
p r2_pru 1_gpo4	PRU1 Gen eral-P urpose Output	O	Y1/AA4	NA			
p r2_pru 1_gpo5	PRU1 Gen eral-P urpose Output	O	W9/D18	P9_25	MODE13		
p r2_pru 1_gpo6	PRU1 Gen eral-P urpose Output	O	V9/E17	P8_9	MODE13		
p r2_pru 1_gpo7	PRU1 Gen eral-P urpose Output	O	V7/C14	P9_31	MODE13		
p r2_pru 1_gpo8	PRU1 Gen eral-P urpose Output	O	U7/G12	P9_18	MODE13		
p r2_pru 1_gpo9	PRU1 Gen eral-P urpose Output	O	V6/F12	P9_17	MODE13		
pr 2_pru1 _gpo10	PRU1 Gen eral-P urpose Output	O	U6/B12	P9_31	MODE13		
pr 2_pru1 _gpo11	PRU1 Gen eral-P urpose Output	O	U5/A11	P9_29	MODE13		
pr 2_pru1 _gpo12	PRU1 Gen eral-P urpose Output	O	V5/B13	P9_30	MODE13		
pr 2_pru1 _gpo13	PRU1 Gen eral-P urpose Output	O	V4/A12	P9_26	MODE13		
pr 2_pru1 _gpo14	PRU1 Gen eral-P urpose Output	O	V3/E14	P9_42	MODE13		
pr 2_pru1 _gpo15	PRU1 Gen eral-P urpose Output	O	Y2/A13	P8_10	MODE13		
pr 2_pru1 _gpo16	PRU1 Gen eral-P urpose Output	O	W2/G14	P8_7	MODE13		
pr 2_pru1 _gpo17	PRU1 Gen eral-P urpose Output	O	E11	P8_27	MODE13		
pr 2_pru1 _gpo18	PRU1 Gen eral-P urpose Output	O	F11	P8_45	MODE13		
pr 2_pru1 _gpo19	PRU1 Gen eral-P urpose Output	O	G10	P8_46	MODE13		
pr 2_pru1 _gpo20	PRU1 Gen eral-P urpose Output	O	F10	P8_43	MODE13		
p r2_pru 1_gpi0	PRU1 Gen eral-P urpose Input	I	V1/D17	P8_32	MODE12		
p r2_pru 1_gpi1	PRU1 Gen eral-P urpose Input	I	U4/AA3	NA			
p r2_pru 1_gpi2	PRU1 Gen eral-P urpose Input	I	U3/AB9	NA			
p r2_pru 1_gpi3	PRU1 Gen eral-P urpose Input	I	V2/AB3	NA			
p r2_pru 1_gpi4	PRU1 Gen eral-P urpose Input	I	Y1/AA4	NA			
p r2_pru 1_gpi5	PRU1 Gen eral-P urpose Input	I	W9/D18	P9_25	MODE12		
p r2_pru 1_gpi6	PRU1 Gen eral-P urpose Input	I	V9/E17	P8_9	MODE12		
p r2_pru 1_gpi7	PRU1 Gen eral-P urpose Input	I	V7/C14	P9_31	MODE12		
p r2_pru 1_gpi8	PRU1 Gen eral-P urpose Input	I	U7/G12	P9_18	MODE12		

continues on next page

Table 4.4 – continued from previous page

SIGNAL NAME	DESCR IPTION	TYPE	PROC	HEAD ER_PIN	MODE	HEAD ER_PIN	MODE
p r2_pru 1_gpi9	PRU1 Gen eral-P urpose Input	I	V6/F12	P9_17	MODE12		
pr 2_pru1 _gpi10	PRU1 Gen eral-P urpose Input	I	U6/B12	P9_31	MODE12		
pr 2_pru1 _gpi11	PRU1 Gen eral-P urpose Input	I	U5/A11	P9_29	MODE12		
pr 2_pru1 _gpi12	PRU1 Gen eral-P urpose Input	I	V5/B13	P9_30	MODE12		
pr 2_pru1 _gpi13	PRU1 Gen eral-P urpose Input	I	V4/A12	P9_28	MODE12		
pr 2_pru1 _gpi14	PRU1 Gen eral-P urpose Input	I	V3/E14	P9_42	MODE12		
pr 2_pru1 _gpi15	PRU1 Gen eral-P urpose Input	I	Y2/A13	P8_10	MODE12		
pr 2_pru1 _gpi16	PRU1 Gen eral-P urpose Input	I	W2/G14	P8_7	MODE12		
pr 2_pru1 _gpi17	PRU1 Gen eral-P urpose Input	I	E11	P8_27	MODE12		
pr 2_pru1 _gpi18	PRU1 Gen eral-P urpose Input	I	F11	P8_45	MODE12		
pr 2_pru1 _gpi19	PRU1 Gen eral-P urpose Input	I	G10	P8_46	MODE12		
pr 2_pru1 _gpi20	PRU1 Gen eral-P urpose Input	I	F10	P8_43	MODE12		
pr2_e dc_lat ch0_in	Latch Input 0	I	F9	P8_42	MODE10		
pr2_e dc_lat ch1_in	Latch Input 1	I	F8	P8_39	MODE10		
pr2_e dc_syn c0_out	SYNC0 Output	O	E7	P8_40	MODE10		
pr2_e dc_syn c1_out	SYNC1 Output	O	E8	P8_37	MODE10		
pr2_e dio_la tch_in	Latch Input	I	D9	P8_38	MODE10		
pr2_ed io_sof	Start Of Frame	O	D7	P8_36	MODE10		
pr2 _uart0 _cts_n	UART C lear-T o-Send	I	D8	P8_34	MODE10		
pr2 _uart0 _rts_n	UART R eady-T o-Send	O	A5	P8_35	MODE10		
p r2_uar t0_rxd	UART R eceive Data	I	C6	P8_33	MODE10		
p r2_uar t0_txd	UART Tr ansmit Data	O	C8	P8_31	MODE10		
pr2 _ecap0 _ecap_ capin_ apwm_o	C apture Inp ut/PWM output	IO	C7	P8_32	MODE10		
pr2_e dio_da ta_in0	Et hernet D igital Input	I	B7	P8_45	MODE10		
pr2_e dio_da ta_in1	Et hernet D igital Input	I	B8	P9_11	MODE10		
pr2_e dio_da ta_in2	Et hernet D igital Input	I	A7	P8_17	MODE10		
pr2_e dio_da ta_in3	Et hernet D igital Input	I	A8	P8_27	MODE10		
pr2_e dio_da ta_in4	Et hernet D igital Input	I	C9	P8_28	MODE10		
pr2_e dio_da ta_in5	Et hernet D igital Input	I	A9	P8_29	MODE10		
pr2_e dio_da ta_in6	Et hernet D igital Input	I	B9	P8_30	MODE10		
pr2_e dio_da ta_in7	Et hernet D igital Input	I	A10	P8_46	MODE10		
pr2_ed io_dat a_out0	Et hernet D igital Output	O	B7	P8_45	MODE11		
pr2_ed io_dat a_out1	Et hernet D igital Output	O	B8	P9_11	MODE11		
pr2_ed io_dat a_out2	Et hernet D igital Output	O	A7	P8_17	MODE11		
pr2_ed io_dat a_out3	Et hernet D igital Output	O	A8	P8_27	MODE11		
pr2_ed io_dat a_out4	Et hernet D igital Output	O	C9	P8_28	MODE11		
pr2_ed io_dat a_out5	Et hernet D igital Output	O	A9	P8_29	MODE11		

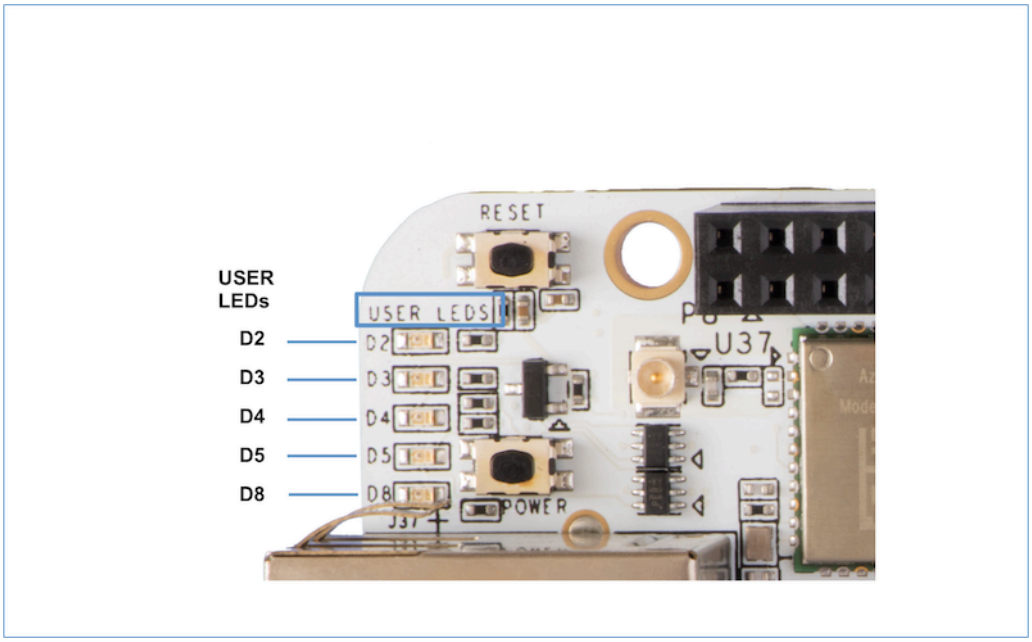
continues on next page

Table 4.4 – continued from previous page

SIGNAL NAME	DESCR IPTION	TYPE	PROC	HEAD ER_PIN	MODE	HEAD ER_PIN	MODE
pr2_ed io_dat a_out6	Et hernet D igital Output	O	B9	P8_30	MODE11		
pr2_ed io_dat a_out7	Et hernet D igital Output	O	A10	P8_46	MODE11		
pr2_mi i1_col	MII1 Col lision Detect	I	D18	P9_25	MODE11		
pr2_mi i1_crs	MII1 C arrier Sense	I	E17	P8_9	MODE11		
pr 2_mdio _mdclk	MDIO Clock	O	C 14/AB3	P9_31	MODE11		
p r2_mdi o_data	MDIO Data	IO	D 14/AA4	P9_29	MODE11		
p r2_mii 0_rxer	MII0 R eceive Error	I	G12	P9_18	MODE11		
pr2 _mii_m t0_clk	MII0 Tr ansmit Clock	I	F12	P9_17	MODE11		
p r2_mii 0_txen	MII0 Tr ansmit Enable	O	B12	P9_31	MODE11		
p r2_mii 0_txd3	MII0 Tr ansmit Data	O	A11	P9_29	MODE11		
p r2_mii 0_txd2	MII0 Tr ansmit Data	O	B13	P9_30	MODE11		
p r2_mii 0_txd1	MII0 Tr ansmit Data	O	A12	P9_28	MODE11		
p r2_mii 0_txd0	MII0 Tr ansmit Data	O	E14	P9_42	MODE11		
pr2 _mii_m r0_clk	MII0 R eceive Clock	I	A13	P8_10	MODE11		
p r2_mii 0_rxdv	MII0 Data Valid	I	G14	P8_7	MODE11		
p r2_mii 0_rxd3	MII0 R eceive Data	I	F14	P8_8	MODE11		
p r2_mii 0_rxd2	MII0 R eceive Data	I	A19	NA			
p r2_mii 0_rxd1	MII0 R eceive Data	I	A18	NA			
p r2_mii 0_rxd0	MII0 R eceive Data	I	C15	NA			
pr2 _mii0_ rxlink	MII0 R eceive Link	I	A16	NA			
pr2_mi i0_crs	MII0 C arrier Sense	I	B18	NA			
pr2_mi i0_col	MII0 Col lision Detect	I	F15	NA			
p r2_mii 1_rxer	MII1 R eceive Error	I	B19	P9_11	MODE11		
pr2 _mii1_ rxlink	MII1 R eceive Link	I	C17	P9_13	MODE11		
pr2 _mii_m t1_clk	MII1 Tr ansmit Clock	I	AC5	NA			
p r2_mii 1_txen	MII1 Tr ansmit Enable	O	AB4	NA			
p r2_mii 1_txd3	MII1 Tr ansmit Data	O	AD4	P8_21	MODE11		
p r2_mii 1_txd2	MII1 Tr ansmit Data	O	AC4	P8_20	MODE11		
p r2_mii 1_txd1	MII1 Tr ansmit Data	O	AC7	P8_25	MODE11		
p r2_mii 1_txd0	MII1 Tr ansmit Data	O	AC6	P8_24	MODE11		
pr2 _mii_m r1_clk	MII1 R eceive Clock	I	AC9	P8_5	MODE11		
p r2_mii 1_rxdv	MII1 Data Valid	I	AC3	P8_6	MODE11		
p r2_mii 1_rxd3	MII1 R eceive Data	I	AC8	P8_23	MODE11		
p r2_mii 1_rxd2	MII1 R eceive Data	I	AD6	P8_22	MODE11		
p r2_mii 1_rxd1	MII1 R eceive Data	I	AB8	P8_3	MODE11		
p r2_mii 1_rxd0	MII1 R eceive Data	I	AB5	P8_4	MODE11		
end	end	end	end	end	end	end	end

4.5.7 User LEDs

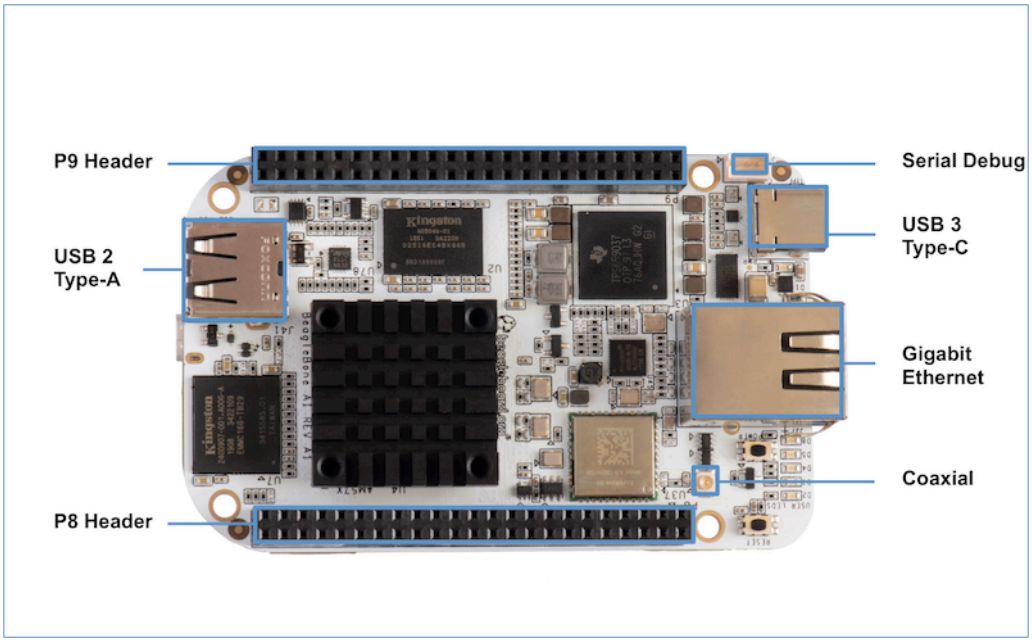
There are 5 User Programmable LEDs on BeagleBone® AI. These are connected to GPIO pins on the processor.

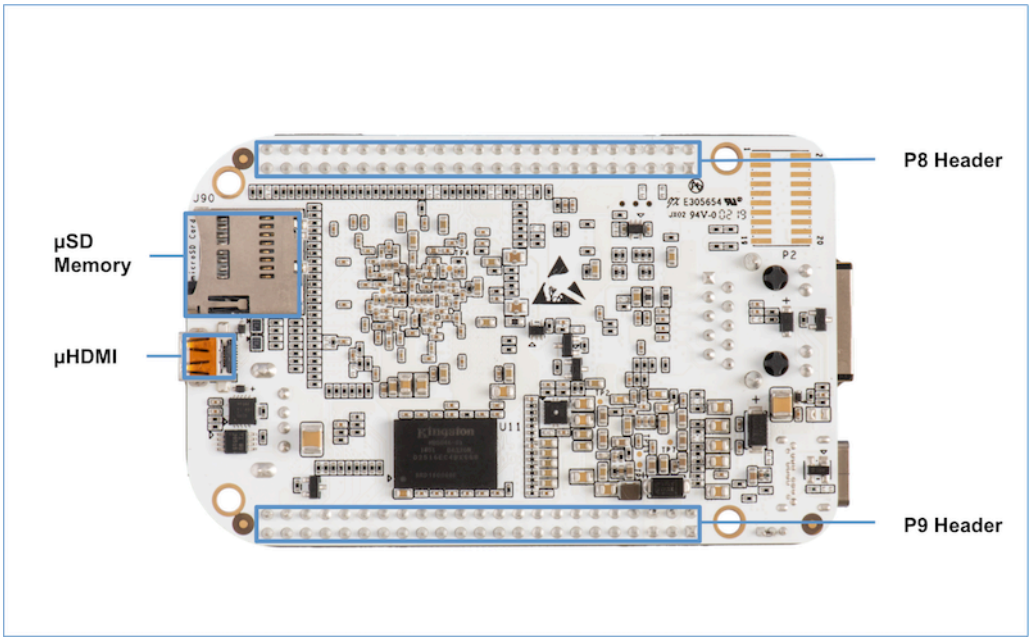


The table shows the signals used to control the LEDs from the processor. Each LED is user programmable. However, there is a Default Functions assigned in the device tree for BeagleBone® AI:

LED	GPIO SIGNAL	DEFAULT FUNCTION
D2	GPIO3_17	Heartbeat When Linux is Running
D3	GPIO5_5	microSD Activity
D4	GPIO3_15	CPU Activity
D5	GPIO3_14	eMMC Activity
D8	GPIO3_7	WiFi/Bluetooth Activity

4.6 Connectors





4.6.1 Expansion Connectors

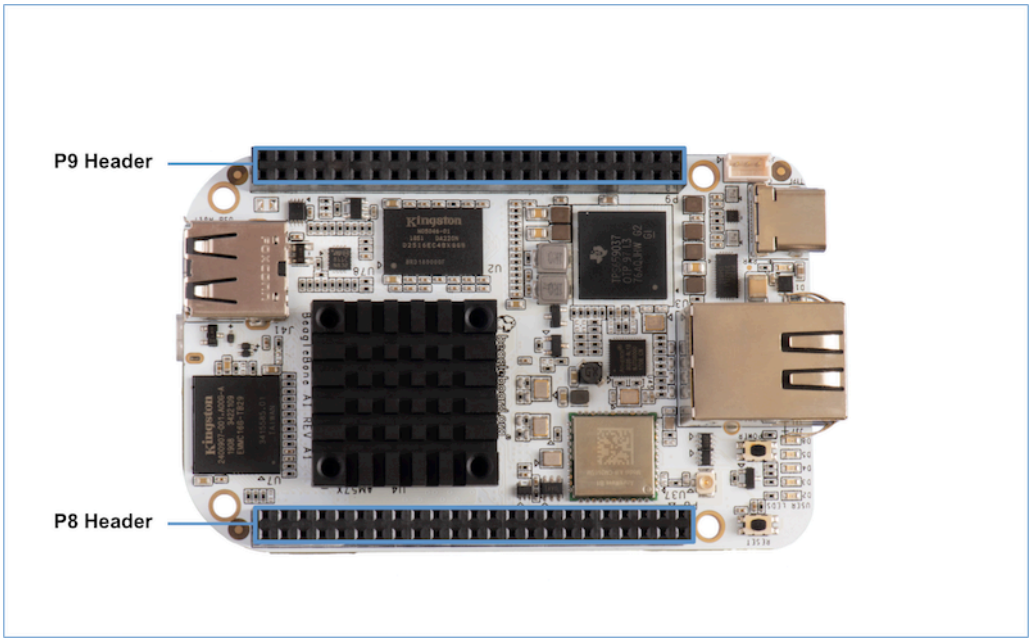
The expansion interface on the board is comprised of two 46 pin connectors, the P8 and P9 Headers. All signals on the expansion headers are **3.3V** unless otherwise indicated.

Note: Do not connect 5V logic level signals to these pins or the board will be damaged.

Note: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

Figure ? shows the location of the expansion connectors.



The location and spacing of the expansion headers are the same as on BeagleBone Black.

Connector P8

The following tables show the pinout of the **P8** expansion header. The SW is responsible for setting the default function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **GPIO** row is the expected gpio identifier number in the Linux kernel.

The **BALL** row is the pin number on the processor.

The **REG** row is the offset of the control register for the processor pin.

The **MODE #** rows are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

If included, the **2nd BALL** row is the pin number on the processor for a second processor pin connected to the same pin on the expansion header. Similarly, all row headings starting with **2nd** refer to data for this second processor pin.

Note: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

Table 4.5: P8.01-P8.02

P8.01	P8.02
GND	GND

Table 4.6: P8.03-P8.05

	P8.03	P8.04	P8.05
GPIO	24	25	193
BALL	AB8	AB5	AC9
REG	0x179C	0x17A0	0x178C
MODE 0	mmc3_dat6	mmc3_dat7	mmc3_dat2
1	spi4_d0	spi4_cs0	spi3_cs0
2	uart10_ctsn	uart10_rtsn	uart5_ctsn
3			
4	vin2b_de1	vin2b_clk1	vin2b_d3
5			
6			
7			
8			
9	vin5a_hsync0	vin5a_vsync0	vin5a_d3
10	ehrpwm3_tripzone_input	eCAP3_in_PWM3_out	eQEP3_index
11	pr2_mii1_rxd1	pr2_mii1_rxd0	pr2_mii_mr1_clk
12	pr2_pru0_gpi10	pr2_pru0_gpi11	pr2_pru0_gpi6
13	pr2_pru0_gpo10	pr2_pru0_gpo11	pr2_pru0_gpo6
14	gpio1_24	gpio1_25	gpio7_1
15	Driver off	Driver off	Driver off

Table 4.7: P8.06-P8.09

	P8.06	P8.07	P8.08	P8.09
GPIO	194	165	166	178
BALL	AC3	G14	F14	E17
REG	0x1790	0x16EC	0x16F0	0x1698
MODE0	mmc3_dat3	mcasp1_axr14	mcasp1_axr15	xref_clk1
1	spi3_cs1	mcasp7_aclkx	mcasp7_fsx	mcasp2_axr9
2	uart5_rtsn	mcasp7_aclkr	mcasp7_fsr	mcasp1_axr5
3				mcasp2_ahclkx
4	vin2b_d2			mcasp6_ahclkx
5				
6				
7		vin6a_d9	vin6a_d8	vin6a_clk0
8				
9	vin5a_d2			
10	eQEP3_strobe	timer11	timer12	timer14
11	pr2_mii1_rxdv	pr2_mii0_rxdv	pr2_mii0_rxd3	pr2_mii1_crs
12	pr2_pru0_gpi7	pr2_pru1_gpi16	pr2_pru0_gpi20	pr2_pru1_gpi6
13	pr2_pru0_gpo7	pr2_pru1_gpo16	pr2_pru0_gpo20	pr2_pru1_gpo6
14	gpio7_2	gpio6_5	gpio6_6	gpio6_18
15	Driver off	Driver off	Driver off	Driver off

Table 4.8: P8.10-P8.13

	P8.10	P8.11	P8.12	P8.13
GPIO	164	75	74	107
BALL	A13	AH4	AG6	D3
REG	0x16E8	0x1510	0x150C	0x1590
MODE 0	mcasp1_axr13	vin1a_d7	vin1a_d6	vin2a_d10
1	mcasp7_axr1			
2				
3		vout3_d0	vout3_d1	mdio_mclk
4		vout3_d16	vout3_d17	vout2_d13
5				
6				
7	vin6a_d10			
8				
9				kbd_col7
10	timer10	eQEP2B_in	eQEP2A_in	ehrpwm2B
11	pr2_mii_mr0_clk			pr1_mdio_mdclk
12	pr2_pru1_gpi15	pr1_pru0_gpi4	pr1_pru0_gpi3	pr1_pru1_gpi7
13	pr2_pru1_gpo15	pr1_pru0_gpo4	pr1_pru0_gpo3	pr1_pru1_gpo7
14	gpio6_4	gpio3_11	gpio3_10	gpio4_11
15	Driver off	Driver off	Driver off	Driver off

Table 4.9: P8.14-P8.16

	P8.14	P8.15	P8.16
GPIO	109	99	125
BALL	D5	D1	B4
REG	0x1598	0x1570	0x15BC
MODE 0	vin2a_d12	vin2a_d2	vin2a_d21
1			
2			vin2b_d2
3	rgmii1_txc		rgmii1_rxd2
4	vout2_d11	vout2_d21	vout2_d2
5		emu12	vin3a_fld0
6			vin3a_d13
7			
8	mii1_rxclk	uart10_rxd	mii1_col
9	kbd_col8	kbd_row6	
10	eCAP2_in_PWM2_out	eCAP1_in_PWM1_out	

continues on next page

Table 4.9 – continued from previous page

	P8.14	P8.15	P8.16
11	pr1_mii1_txd1	pr1_ecap0_ecap_capin_apwm_o	pr1_mii1_rxlink
12	pr1_pru1_gpi9	pr1_edio_data_in7	pr1_pru1_gpi18
13	pr1_pru1_gpo9	pr1_edio_data_out7	pr1_pru1_gpo18
14	gpio4_13	gpio4_3	gpio4_29
15	Driver off	Driver off	Driver off
2nd BALL		A3	
2nd REG		0x15B4	
2nd MODE 0		vin2a_d19	
2nd 1			
2nd 2		vin2b_d4	
2nd 3		rgmii1_rxctl	
2nd 4		vout2_d4	
2nd 5			
2nd 6		vin3a_d11	
2nd 7			
2nd 8		mii1_txer	
2nd 9			
2nd 10		ehrpwm3_tripzone_input	
2nd 11		pr1_mii1_rxd0	
2nd 12		pr1_pru1_gpi16	
2nd 13		pr1_pru1_gpo16	
2nd 14		gpio4_27	
2nd 15		Driver off	

Table 4.10: P8.17-P8.19

	P8.17	P8.18	P8.19
GPIO	242	105	106
BALL	A7	F5	E6
REG	0x1624	0x1588	0x158C
MODE 0	vout1_d18	vin2a_d8	vin2a_d9
1			
2	emu4		
3	vin4a_d2		
4	vin3a_d2	vout2_d15	vout2_d14
5	obs11	emu18	emu19
6	obs27		
7			
8		mii1_rxd3	mii1_rxd0
9		kbd_col5	kbd_col6
10	pr2_edio_data_in2	eQEP2_strobe	ehrpwm2A
11	pr2_edio_data_out2	pr1_mii1_txd3	pr1_mii1_txd2
12	pr2_pru0_gpi15	pr1_pru1_gpi5	pr1_pru1_gpi6
13	pr2_pru0_gpo15	pr1_pru1_gpo5	pr1_pru1_gpo6
14	gpio8_18	gpio4_9	gpio4_10
15	Driver off	Driver off	Driver off

Table 4.11: P8.20-P8.22

	P8.20	P8.21	P8.22
GPIO	190	189	23
BALL	AC4	AD4	AD6
REG	0x1780	0x177C	0x1798
MODE 0	mmc3_cmd	mmc3_clk	mmc3_dat5
1	spi3_sclk		spi4_d1
2			uart10_txd
3			
4	vin2b_d6	vin2b_d7	vin2b_d0
5			
6			
7			
8			
9	vin5a_d6	vin5a_d7	vin5a_d0
10	eCAP2_in_PWM2_out	ehrpwm2_tripzone_input	ehrpwm3B
11	pr2_mii1_txd2	pr2_mii1_txd3	pr2_mii1_rxd2
12	pr2_pru0_gpi3	pr2_pru0_gpi2	pr2_pru0_gpi9
13	pr2_pru0_gpo3	pr2_pru0_gpo2	pr2_pru0_gpo9
14	gpio6_30	gpio6_29	gpio1_23
15	Driver off	Driver off	Driver off

Table 4.12: P8.23-P8.26

	P8.23	P8.24	P8.25	P8.26
GPIO	22	192	191	124
BALL	AC8	AC6	AC7	B3
REG	0x1794	0x1788	0x1784	0x15B8
MODE 0	mmc3_dat4	mmc3_dat1	mmc3_dat0	vin2a_d20
1	spi4_sclk	spi3_d0	spi3_d1	
2	uart10_rxd	uart5_txd	uart5_rxd	vin2b_d3
3				rgmii1_rxd3
4	vin2b_d1	vin2b_d4	vin2b_d5	vout2_d3
5				vin3a_de0
6				vin3a_d12
7				
8				mii1_rxer
9	vin5a_d1	vin5a_d4	vin5a_d5	
10	ehrpwm3A	eQEP3B_in	eQEP3A_in	eCAP3_in_PWM3_out
11	pr2_mii1_rxd3	pr2_mii1_txd0	pr2_mii1_txd1	pr1_mii1_rxer
12	pr2_pru0_gpi8	pr2_pru0_gpi5	pr2_pru0_gpi4	pr1_pru1_gpi17
13	pr2_pru0_gpo8	pr2_pru0_gpo5	pr2_pru0_gpo4	pr1_pru1_gpo17
14	gpio1_22	gpio7_0	gpio6_31	gpio4_28
15	Driver off	Driver off	Driver off	Driver off

Table 4.13: P8.27-P8.29

	P8.27	P8.28	P8.29
GPIO	119	115	118
BALL	E11	D11	C11
REG	0x15D8	0x15C8	0x15D4
MODE 0	vout1_vsync	vout1_clk	vout1_hsync
1			
2			
3	vin4a_vsync0	vin4a_fld0	vin4a_hsync0
4	vin3a_vsync0	vin3a_fld0	vin3a_hsync0
5			
6			
7			
8	spi3_sclk	spi3_cs0	spi3_d0
9			
10			

continues on next page

Table 4.13 – continued from previous page

	P8.27	P8.28	P8.29
11			
12	pr2_pru1_gpi17		
13	pr2_pru1_gpo17		
14	gpio4_23	gpio4_19	gpio4_22
15	Driver off	Driver off	Driver off
2nd BALL	A8	C9	A9
2nd REG	0x1628	0x162C	0x1630
2nd MODE0	vout1_d19	vout1_d20	vout1_d21
2nd 1			
2nd 2	emu15	emu16	emu17
2nd 3	vin4a_d3	vin4a_d4	vin4a_d5
2nd 4	vin3a_d3	vin3a_d4	vin3a_d5
2nd 5	obs12	obs13	obs14
2nd 6	obs28	obs29	obs30
2nd 7			
2nd 8			
2nd 9			
2nd 10	pr2_edio_data_in3	pr2_edio_data_in4	pr2_edio_data_in5
2nd 11	pr2_edio_data_out3	pr2_edio_data_out4	pr2_edio_data_out5
2nd 12	pr2_pru0_gpi16	pr2_pru0_gpi17	pr2_pru0_gpi18
2nd 13	pr2_pru0_gpo16	pr2_pru0_gpo17	pr2_pru0_gpo18
2nd 14	gpio8_19	gpio8_20	gpio8_21
2nd 15	Driver off	Driver off	Driver off

Table 4.14: P8.30-P8.32

	P8.30	P8.31	P8.32
GPIO	116	238	239
BALL	B10	C8	C7
REG	0x15CC	0x1614	0x1618
MODE 0	vout1_de	vout1_d14	vout1_d15
1			
2		emu13	emu14
3	vin4a_de0	vin4a_d14	vin4a_d15
4	vin3a_de0	vin3a_d14	vin3a_d15
5		obs9	obs10
6		obs25	obs26
7			
8	spi3_d1		
9			
10		pr2_uart0_txd	pr2_ecap0_ecap_capin_apwm_o
11			
12		pr2_pru0_gpi11	pr2_pru0_gpi12
13		pr2_pru0_gpo11	pr2_pru0_gpo12
14	gpio4_20	gpio8_14	gpio8_15
15	Driver off	Driver off	Driver off
2nd BALL	B9	G16	D17
2nd REG	0x1634	0x173C	0x1740
2nd MODE 0	vout1_d22	mcasp4_axr0	mcasp4_axr1
2nd 1			
2nd 2	emu18	spi3_d0	spi3_cs0
2nd 3	vin4a_d6	uart8_ctsn	uart8_rtsn
2nd 4	vin3a_d6	uart4_rxd	uart4_txd

continues on next page

Table 4.14 – continued from previous page

	P8.30	P8.31	P8.32
2nd 5	obs15		
2nd 6	obs31	vout2_d18	vout2_d19
2nd 7			
2nd 8		vin4a_d18	vin4a_d19
2nd 9		vin5a_d13	vin5a_d12
2nd 10	pr2_edio_data_in6		
2nd 11	pr2_edio_data_out6		
2nd 12	pr2_pru0_gpi19		pr2_pru1_gpi0
2nd 13	pr2_pru0_gpo19		pr2_pru1_gpo0
2nd 14	gpio8_22		
2nd 15	Driver off	Driver off	Driver off

Table 4.15: P8.33-P8.35

	P8.33	P8.34	P8.35
GPIO	237	235	236
BALL	C6	D8	A5
REG	0x1610	0x1608	0x160C
MODE 0	vout1_d13	vout1_d11	vout1_d12
1			
2	emu12	emu10	emu11
3	vin4a_d13	vin4a_d11	vin4a_d12
4	vin3a_d13	vin3a_d11	vin3a_d12
5	obs8	obs6	obs7
6	obs24	obs22	obs23
7		obs_dmarq2	
8			
9			
10	pr2_uart0_rxd	pr2_uart0_cts_n	pr2_uart0_rts_n
11			
12	pr2_pru0_gpi10	pr2_pru0_gpi8	pr2_pru0_gpi9
13	pr2_pru0_gpo10	pr2_pru0_gpo8	pr2_pru0_gpo9
14	gpio8_13	gpio8_11	gpio8_12
15	Driver off	Driver off	Driver off
2nd BALL	AF9	G6	AD9
2nd REG	0x14E8	0x1564	0x14E4
2nd MODE0	vin1a_fld0	vin2a_vsync0	vin1a_de0
2nd 1	vin1b_vsync1		vin1b_hsync1
2nd 2			
2nd 3		vin2b_vsync1	vout3_d17
2nd 4	vout3_clk	vout2_vsync	vout3_de
2nd 5	uart7_txd	emu9	uart7_rxd
2nd 6			
2nd 7	timer15	uart9_txd	timer16
2nd 8	spi3_d1	spi4_d1	spi3_sclk
2nd 9	kbd_row1	kbd_row3	kbd_row0
2nd 10	eQEP1B_in	ehrpwm1A	eQEP1A_in
2nd 11		pr1_uart0_rts_n	
2nd 12		pr1_edio_data_in4	
2nd 13		pr1_edio_data_out4	
2nd 14	gpio3_1	gpio4_0	gpio3_0
2nd 15	Driver off	Driver off	Driver off

Table 4.16: P8.36-P8.38

	P8.36	P8.37	P8.38
GPIO	234	232	233
BALL	D7	E8	D9
REG	0x1604	0x15FC	0x1600
MODE 0	vout1_d10	vout1_d8	vout1_d9
1			
2	emu3	uart6_rxd	uart6_txd
3	vin4a_d10	vin4a_d8	vin4a_d9
4	vin3a_d10	vin3a_d8	vin3a_d9
5	obs5		
6	obs21		
7	obs_irq2		
8			
9			
10	pr2_edio_sof	pr2_edc_sync1_out	pr2_edio_latch_in
11			
12	pr2_pru0_gpi7	pr2_pru0_gpi5	pr2_pru0_gpi6
13	pr2_pru0_gpo7	pr2_pru0_gpo5	pr2_pru0_gpo6
14	gpio8_10	gpio8_8	gpio8_9
15	Driver off	Driver off	Driver off
2nd BALL	F2	A21	C18
2nd REG	0x1568	0x1738	0x1734
2nd MODE 0	vin2a_d0	mcasp4_fsx	mcasp4_aclkx
2nd 1		mcasp4_fsr	mcasp4_aclkr
2nd 2		spi3_d1	spi3_sclk
2nd 3		uart8_txd	uart8_rxd
2nd 4	vout2_d23	i2c4_scl	i2c4_sda
2nd 5	emu10		
2nd 6		vout2_d17	vout2_d16
2nd 7	uart9_ctsn		
2nd 8	spi4_d0	vin4a_d17	vin4a_d16
2nd 9	kbd_row4	vin5a_d14	vin5a_d15
2nd 10	ehrpwm1B		
2nd 11	pr1_uart0_rxd		
2nd 12	pr1_edio_data_in5		
2nd 13	pr1_edio_data_out5		
2nd 14	gpio4_1		
2nd 15	Driver off	Driver off	Driver off

Table 4.17: P8.39-P8.41

	P8.39	P8.40	P8.41
GPIO	230	231	228
BALL	F8	E7	E9
REG	0x15F4	0x15F8	0x15EC
MODE 0	vout1_d6	vout1_d7	vout1_d4
1			
2	emu8	emu9	emu6
3	vin4a_d22	vin4a_d23	vin4a_d20
4	vin3a_d22	vin3a_d23	vin3a_d20
5	obs4		obs2
6	obs20		obs18
7			
8			
9			
10	pr2_edc_latch1_in	pr2_edc_sync0_out	pr1_ecap0_ecap_capin_apwm_o
11			
12	pr2_pru0_gpi3	pr2_pru0_gpi4	pr2_pru0_gpi1
13	pr2_pru0_gpo3	pr2_pru0_gpo4	pr2_pru0_gpo1
14	gpio8_6	gpio8_7	gpio8_4
15	Driver off	Driver off	Driver off

Table 4.18: P8.42-P8.44

	P8.42	P8.43	P8.44
GPIO	229	226	227
BALL	F9	F10	G11
REG	0x15F0	0x15E4	0x15E8
MODE 0	vout1_d5	vout1_d2	vout1_d3
1			
2	emu7	emu2	emu5
3	vin4a_d21	vin4a_d18	vin4a_d19
4	vin3a_d21	vin3a_d18	vin3a_d19
5	obs3	obs0	obs1
6	obs19	obs16	obs17
7		obs_irq1	obs_dmarq1
8			
9			
10	pr2_edc_latch0_in	pr1_uart0_rxd	pr1_uart0_txd
11			
12	pr2_pru0_gpi2	pr2_pru1_gpi20	pr2_pru0_gpi0
13	pr2_pru0_gpo2	pr2_pru1_gpo20	pr2_pru0_gpo0
14	gpio8_5	gpio8_2	gpio8_3
15	Driver off	Driver off	Driver off

Table 4.19: P8.45-P8.46

	P8.45	P8.46
GPIO	224	225
BALL	F11	G10
REG	0x15DC	0x15E0
MODE 0	vout1_d0	vout1_d1
1		
2	uart5_rxd	uart5_txd
3	vin4a_d16	vin4a_d17
4	vin3a_d16	vin3a_d17
5		
6		
7		
8	spi3_cs2	
9		
10	pr1_uart0_cts_n	pr1_uart0_rts_n

continues on next page

Table 4.19 – continued from previous page

	P8.45	P8.46
11		
12	pr2_pru1_gpi18	pr2_pru1_gpi19
13	pr2_pru1_gpo18	pr2_pru1_gpo19
14	gpio8_0	gpio8_1
15	Driver off	Driver off
2nd BALL	B7	A10
2nd REG	0x161C	0x1638
2nd MODE 0	vout1_d16	vout1_d23
2nd 1		
2nd 2	uart7_rxd	emu19
2nd 3	vin4a_d0	vin4a_d7
2nd 4	vin3a_d0	vin3a_d7
2nd 5		
2nd 6		
2nd 7		
2nd 8		spi3_cs3
2nd 9		
2nd 10	pr2_edio_data_in0	pr2_edio_data_in7
2nd 11	pr2_edio_data_out0	pr2_edio_data_out7
2nd 12	pr2_pru0_gpi13	pr2_pru0_gpi20
2nd 13	pr2_pru0_gpo13	pr2_pru0_gpo20
2nd 14	gpio8_16	gpio8_23
2nd 15	Driver off	Driver off

TODO: Notes regarding the resistors on muxed pins.

Connector P9

The following tables show the pinout of the **P9** expansion header. The SW is responsible for setting the default function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **GPIO** row is the expected gpio identifier number in the Linux kernel.

The **BALL** row is the pin number on the processor.

The **REG** row is the offset of the control register for the processor pin.

The **MODE #** rows are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

If included, the **2nd BALL** row is the pin number on the processor for a second processor pin connected to the same pin on the expansion header. Similarly, all row headings starting with **2nd** refer to data for this second processor pin.

NOTES:

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

In the table are the following notations:

PWR_BUT is a 5V level as pulled up internally by the TPS6590379. It is activated by pulling the signal to GND.

TODO: (Actually, on BeagleBone AI, I believe PWR_BUT is pulled to 3.3V, but activation is still done by pulling the signal to GND. Also, a quick grounding of PWR_BUT will trigger a system event where shutdown can occur, but there is no hardware power-off function like on BeagleBone Black via this signal. It does, however, act as a hardware power-on.)

TODO: (On BeagleBone Black, SYS_RESET was a bi-directional signal, but it is only an output from BeagleBone AI to capes on BeagleBone AI.)

Table 4.20: P9.01-P9.05

P9.01	P9.02	P9.03	P9.04	P9.05
GND	GND	VOUT_3V3	VOUT_3V3	VIN

Table 4.21: P9.06-P9.10

P9.06	P9.07 | P9.08 | P9.09 | P9.10			
VIN	VOUT_SYS	VOUT_SYS	RESET#	RESET#

Table 4.22: P9.11-P9.13

	P9.11	P9.12	P9.13
GPIO	241	128	172
BALL	B19	B14	C17
REG	0x172C	0x16AC	0x1730
MODE 0	mcasp3_axr0	mcasp1_aclkr	mcasp3_axr1
1		mcasp7_axr2	
2	mcasp2_axr14		mcasp2_axr15
3	uart7_ctsn		uart7_rtsn
4	uart5_rxd		uart5_txd
5			
6		vout2_d0	
7	vin6a_d1		vin6a_d0
8		vin4a_d0	
9			vin5a_fld0
10		i2c4_sda	
11	pr2_mii1_rxer		pr2_mii1_rxlink
12	pr2_pru0_gpi14		pr2_pru0_gpi15
13	pr2_pru0_gpo14		pr2_pru0_gpo15
14		gpio5_0	
15	Driver off	Driver off	Driver off
2nd BALL	B8		AB10**
2nd REG	0x1620		0x1680
2nd MODE 0	vout1_d17		usb1_drvvbus
2nd 1			
2nd 2	uart7_txd		
2nd 3	vin4a_d1		
2nd 4	vin3a_d1		
2nd 5			
2nd 6			
2nd 7			timer16
2nd 8			
2nd 9			
2nd 10	pr2_edio_data_in1		
2nd 11	pr2_edio_data_out1		
2nd 12	pr2_pru0_gpi14		
2nd 13	pr2_pru0_gpo14		
2nd 14	gpio8_17		gpio6_12

continues on next page

Table 4.22 – continued from previous page

	P9.11	P9.12	P9.13
2nd 15	Driver off		Driver off

Table 4.23: P9.14-P9.16

	P9.14	P9.15	P9.16
GPIO	121	76	122
BALL	D6	AG4	C5
REG	0x15AC	0x1514	0x15B0
MODE 0	vin2a_d17	vin1a_d8	vin2a_d18
1		vin1b_d7	
2	vin2b_d6		vin2b_d5
3	rgmii1_txd0		rgmii1_rxc
4	vout2_d6	vout3_d15	vout2_d5
5			
6	vin3a_d9		vin3a_d10
7			
8	mii1_txd2		mii1_txd3
9		kbd_row2	
10	ehrpwm3A	eQEP2_index	ehrpwm3B
11	pr1_mii1_rxd2		pr1_mii1_rxd1
12	pr1_pru1_gpi14	pr1_pru0_gpi5	pr1_pru1_gpi15
13	pr1_pru1_gpo14	pr1_pru0_gpo5	pr1_pru1_gpo15
14	gpio4_25	gpio3_12	gpio4_26
15	Driver off	Driver off	Driver off

Table 4.24: P9.17-P9.19

	P9.17	P9.18	P9.19
GPIO	209	208	195
BALL	B24	G17	R6
REG	0x17CC	0x17C8	0x1440
MODE 0	spi2_cs0	spi2_d0	gpmc_a0
1	uart3_rtsn	uart3_ctsn	
2	uart5_txd	uart5_rxd	vin3a_d16
3			vout3_d16
4			vin4a_d0
5			
6			vin4b_d0
7			i2c4_scl
8			uart5_rxd
9			
10			
11			
12			
13			
14	gpio7_17	gpio7_16	gpio7_3
15	Driver off	Driver off	Driver off
2nd BALL	F12	G12	F4
2nd REG	0x16B8	0x16B4	0x157C
2nd MODE 0	mcasp1_axr1	mcasp1_axr0	vin2a_d5
2nd 1			
2nd 2			
2nd 3	uart6_txd	uart6_rxd	
2nd 4			vout2_d18
2nd 5			emu15
2nd 6			
2nd 7	vin6a_hsync0	vin6a_vsync0	

continues on next page

Table 4.24 – continued from previous page

	P9.17	P9.18	P9.19
2nd 8			uart10_rtsn
2nd 9			kbd_col2
2nd 10	i2c5_scl	i2c5_sda	eQEP2A_in
2nd 11	pr2_mii_mt0_clk	pr2_mii0_rxer	pr1_edio_sof
2nd 12	pr2_pru1_gpi9	pr2_pru1_gpi8	pr1_pru1_gpi2
2nd 13	pr2_pru1_gpo9	pr2_pru1_gpo8	pr1_pru1_gpo2
2nd 14	gpio5_3	gpio5_2	gpio4_6
2nd 15	Driver off	Driver off	Driver off

Table 4.25: P9.20-P9.22

	P9.20	P9.21	P9.22
GPIO	196	67	179
BALL	T9	AF8	B26
REG	0x1444	0x14F0	0x169C
MODE 0	gpmc_a1	vin1a_vsync0	xref_clk2
1		vin1b_de1	mcasp2_axr10
2	vin3a_d17		mcasp1_axr6
3	vout3_d17		mcasp3_ahclkx
4	vin4a_d1	vout3_vsync	mcasp7_ahclkx
5		uart7_rtsn	
6	vin4b_d1		vout2_clk
7	i2c4_sda	timer13	
8	uart5_txd	spi3_cs0	vin4a_clk0
9			
10		eQEP1_strobe	timer15
11			
12			
13			
14	gpio7_4	gpio3_3	gpio6_19
15	Driver off	Driver off	Driver off
2nd BALL	D2	B22	A26
2nd REG	0x1578	0x17C4	0x17C0
2nd MODE 0	vin2a_d4	spi2_d1	spi2_sclk
2nd 1		uart3_txd	uart3_rxd
2nd 2			
2nd 3			
2nd 4	vout2_d19		
2nd 5	emu14		
2nd 6			
2nd 7			
2nd 8	uart10_ctsn		
2nd 9	kbd_col1		
2nd 10	ehrpwm1_synco		
2nd 11	pr1_edc_sync0_out		
2nd 12	pr1_pru1_gpi1		
2nd 13	pr1_pru1_gpo1		
2nd 14	gpio4_5	gpio7_15	gpio7_14
2nd 15	Driver off	Driver off	Driver off

Table 4.26: P9.23-P9.25

	P9.23	P9.24	P9.25
GPIO	203	175	177
BALL	A22	F20	D18
REG	0x17B4	0x168C	0x1694
MODE 0	spi1_cs1	gpio6_15	xref_clk0
1		mcasp1_axr9	mcasp2_axr8
2	sata1_led	dcan2_rx	mcasp1_axr4
3	spi2_cs1	uart10_txd	mcasp1_ahclkx
4			mcasp5_ahclkx
5			
6		vout2_vsync	
7			vin6a_d0
8		vin4a_vsync0	hdq0
9		i2c3_scl	clkout2
10		timer2	timer13
11			pr2_mii1_col
12			pr2_pru1_gpi5
13			pr2_pru1_gpo5
14	gpio7_11	gpio6_15	gpio6_17
15	Driver off	Driver off	Driver off

Table 4.27: P9.26-P9.29

	P9.26	P9.27	P9.28	P9.29
GPIO	174	111	113	139
BALL	E21	C3	A12	A11
REG	0x1688	0x15A0	0x16E0	0x16D8
MODE 0	gpio6_14	vin2a_d14	mcasp1_axr11	mcasp1_axr9
1	mcasp1_axr8		mcasp6_fsx	mcasp6_axr1
2	dcan2_tx		mcasp6_fsr	
3	uart10_rxd	rgmii1_txd3	spi3_cs0	spi3_d1
4		vout2_d9		
5				
6	vout2_hsync			
7			vin6a_d12	vin6a_d14
8	vin4a_hsync0	mii1_txclk		
9	i2c3_sda			
10	timer1	eQEP3B_in	timer8	timer6
11		pr1_mii_mr1_clk	pr2_mii0_txd1	pr2_mii0_txd3
12		pr1_pru1_gpi11	pr2_pru1_gpi13	pr2_pru1_gpi11
13		pr1_pru1_gpo11	pr2_pru1_gpo13	pr2_pru1_gpo11
14	gpio6_14	gpio4_15	gpio4_17	gpio5_11
15	Driver off	Driver off	Driver off	Driver off
2nd BALL	AE2	J14		D14
2nd REG	0x1544	0x16B0		0x16A8
2nd MODE 0	vin1a_d20	mcasp1_fsr		mcasp1_fsx
2nd 1	vin1b_d3	mcasp7_axr3		
2nd 2				
2nd 3				
2nd 4	vout3_d3			
2nd 5				
2nd 6	vin3a_d4	vout2_d1		
2nd 7				vin6a_de0
2nd 8		vin4a_d1		
2nd 9	kbd_col5			
2nd 10	pr1_edio_data_in4	i2c4_scl		i2c3_scl
2nd 11	pr1_edio_data_out4			pr2_mdio_data

continues on next page

Table 4.27 – continued from previous page

	P9.26	P9.27	P9.28	P9.29
2nd 12	pr1_pru0_gpi17			
2nd 13	pr1_pru0_gpo17			
2nd 14	gpio3_24	gpio5_1		gpio7_30
2nd 15	Driver off	Driver off		Driveroff

Table 4.28: P9.30-P9.31

	P9.30	P9.31
GPIO	140	138
BALL	B13	B12
REG	0x16DC	0x16D4
MODE 0	mcasp1_axr10	mcasp1_axr8
1	mcasp6_aclkx	mcasp6_axr0
2	mcasp6_aclkr	
3	spi3_d0	spi3_sclk
4		
5		
6		
7	vin6a_d13	vin6a_d15
8		
9		
10	timer7	timer5
11	pr2_mii0_txd2	pr2_mii0_txen
12	pr2_pru1_gpi12	pr2_pru1_gpi10
13	pr2_pru1_gpo12	pr2_pru1_gpo10
14	gpio5_12	gpio5_10
15	Driver off	Driver off
2nd BALL		C14
2nd REG		0x16A4
2nd MODE 0		mcasp1_aclkx
2nd 1		
2nd 2		
2nd 3		
2nd 4		
2nd 5		
2nd 6		
2nd 7		vin6a_fld0
2nd 8		
2nd 9		
2nd 10		i2c3_sda
2nd 11		pr2_mdio_mdclk
2nd 12		pr2_pru1_gpi7
2nd 13		pr2_pru1_gpo7
2nd 14		gpio7_31
2nd 15		Driver off

Todo: This table needs entries

Table 4.29: P9.32-P9.40

	P9.32	P9.33	P9.34	P9.35	P9.36	P9.37	P9.38	P9.39	P9.40
Row 1	P9.32	P9.33	P9.34	P9.35	P9.36	P9.37	P9.38	P9.39	P9.40

Table 4.30: P9.41-P9.42

	P9.41	P9.42
GPIO	180	114
BALL	C23	E14
REG	0x16A0	0x16E4
MODE 0	xref_clk3	mcasp1_axr12
1	mcasp2_axr11	mcasp7_axr0
2	mcasp1_axr7	
3	mcasp4_ahclkx	spi3_cs1
4	mcasp8_ahclkx	
5		
6	vout2_de	
7	hdq0	vin6a_d11
8	vin4a_de0	
9	clkout3	
10	timer16	timer9
11		pr2_mii0_txd0
12		pr2_pru1_gpi14
13		pr2_pru1_gpo14
14	gpio6_20	gpio4_18
15	Driver off	Driver off
2nd BALL	C1	C2
2nd REG	0x1580	0x159C
2nd MODE 0	vin2a_d6	vin2a_d13
2nd 1		
2nd 2		
2nd 3		rgmii1_txctl
2nd 4	vout2_d17	vout2_d10
2nd 5	emu16	
2nd 6		
2nd 7		
2nd 8	mii1_rxd1	mii1_rxdv
2nd 9	kbd_col3	kbd_row8
2nd 10	eQEP2B_in	eQEP3A_in
2nd 11	pr1_mii_mt1_clk	pr1_mii1_txd0
2nd 12	pr1_pru1_gpi3	pr1_pru1_gpi10
2nd 13	pr1_pru1_gpo3	pr1_pru1_gpo10
2nd 14	gpio4_7	gpio4_14
2nd 15	Driver off	Driver off

Todo: Table entries needed

Table 4.31: P9.43-P9.46

	P9.43	P9.44	P9.45	P9.46
Row 1	P9.43	P9.44	P9.45	P9.46

4.6.2 Serial Debug

Todo: Need info on BealgeBone AI serial debug

4.6.3 USB 3 Type-C

Todo: Need info on BealgeBone AI USB Type-C connection

4.6.4 USB 2 Type-A

Todo: Need info on BealgeBone AI USB Type-A connection

4.6.5 Gigabit Ethernet

Todo: Need info on BealgeBone AI USB Gigabit Ethernet connection

4.6.6 Coaxial

Todo: Need info on BealgeBone AI u.FL antenna connection

4.6.7 microSD Memory

Todo: Need info on BealgeBone AI uSD card slot

4.6.8 microHDMI

Todo: Need info on BealgeBone AI uHDMI connection

4.7 Cape Board Support

There is a Cape Headers Google Spreadsheet which has a lot of detail regarding various boards and cape add-on boards.

See also https://elinux.org/Beagleboard:BeagleBone_cape_interface_spec

TODO

4.7.1 BeagleBone® Black Cape Compatibility

TODO

See https://elinux.org/Beagleboard:BeagleBone_cape_interface_spec for now.

4.7.2 EEPROM

TODO

4.7.3 Pin Usage Consideration

TODO

4.7.4 GPIO

TODO

4.7.5 I2C

TODO

4.7.6 UART or PRU UART

This section is about both UART pins on the header and PRU UART pins on the headers we will include a chart and later some code

Table 4.32: UART

Function	Pin	ABC Ball	Pinctrl Register	Mode
uart3_txd	P9.21	B22	0x17C4	1
uart3_rxd	P9.22	A26	0x17C0	1
uart5_txd	P9.13	C17	0x1730	4
uart5_rxd	P9.11	B19	0x172C	4
uart5_ctsn	P8.05	AC9	0x178C	2
uart5_rtsn	P8.06	AC3	0x1790	2
uart8_txd	P8.37	A21	0x1738	3
uart8_rxd	P8.38	C18	0x1734	3
uart8_ctsn	P8.31	G16	0x173C	3
uart8_rtsn	P8.32	D17	0x1740	3
uart10_txd	P9.24	F20	0x168C	3
uart10_rxd	P9.26	E21	0x1688	3
uart10_ctsn	P8.03	AB8	0x179C	2
uart10_rtsn	P8.04	AB5	0x17A0	2
uart10_txd	P9.24	F20	0x168C	3
uart10_rxd	P9.26	E21	0x1688	3
uart10_ctsn	P9.20	D2	0x1578	8
uart10_rtsn	P9.19	F4	0x157C	8

Table 4.33: PRU UART

Function	Pin	ABC Ball	Pinctrl Register	Mode
pr2_uart0_txd	P8.31	C8	0x1614	10
pr2_uart0_rxd	P8.33	C6	0x1610	10
pr2_uart0_cts_n	P8.34	D8	0x1608	10
pr2_uart0_rts_n	P8.35	A5	0x160C	10
pr1_uart0_rxd	P8.43	F10	0x15E4	10
pr1_uart0_txd	P8.44	G11	0x15E8	10
pr1_uart0_cts_n	P8.45	F11	0x15DC	10
pr1_uart0_rts_n	P8.46	G10	0x15E0	10

TODO

4.7.7 SPI

TODO

4.7.8 Analog

TODO

4.7.9 PWM, TIMER, eCAP or PRU PWM/eCAP

TODO

4.7.10 eQEP

TODO

4.7.11 CAN

TODO

4.7.12 McASP (audio serial like I2S and AC97)

TODO

4.7.13 MMC

TODO

4.7.14 LCD

TODO

4.7.15 PRU GPIO

TODO

4.7.16 CLKOUT

TODO

4.7.17 Expansion Connector Headers

TODO: discuss header options for working with the expansion connectors per https://git.beagleboard.org/beagleboard/beaglebone-black/-/wikis/System-Reference-Manual#section-7-1

4.7.18 Signal Usage

TODO

4.7.19 Cape Power

TODO

4.7.20 Mechanical

TODO

4.8 Additional Support Information

Todo: Reference https://docs.beagleboard.org/latest/intro/support/index.html and https://beagleboard.org/resources

Related TI documentation: http://www.ti.com/tool/BEAGLE-3P-BBONE-AI

4.8.1 REGULATORY, COMPLIANCE, AND EXPORT INFORMATION

- Country of origin: PRC
- FCC: 2ATUT-BBONE-AI
- CE: TBD
- CNHTS: 8543909000
- USHTS: 8473301180
- MXHTS: 84733001
- TARIC: 8473302000
- ECCN: 5A992.C
- CCATS: Z1613110/G180570
- RoHS/REACH: TBD
- Volatility: TBD

4.8.2 Mechanical Information

- Board Dimensions: 8.9cm x 5.4cm x 1.5cm
- Board Net Weight 48g
- Packaging Dimensions: 13.8cm x 10cm x 4cm
- Gross Weight (including packaging): 110g
- 3D STEP model: https://git.beagleboard.org/beagleboard/beaglebone-ai/-/tree/master/Mechanical

4.8.3 Change History

Rev A0

Initial prototype revision. Not taken to production. eMMC flash image provided by Embest.

Rev A1

Second round prototype.

- Fixed size of mounting holes.
- Added LED for WiFi status.
- Added microHDMI.
- Changed eMMC voltage from 3.3V to 1.8V to support HS200.
- Changed eMMC from 4GB to 16GB.
- Changed serial debug header from 6-pin 100mil pitch to 3-pin 1.5mm pitch.
- Switched expansion header from UART4 to UART5. The UART4 pins were used for the microHDMI.

eMMC flash image provided by Embest.

Rev A1a

Alpha pilot-run units and initial production.

- Added pull-down resistor on serial debug header RX line.

Alpha pilot-run eMMC flash image: https://debian.beagleboard.org/images/bbai-pilot-20190408.img.xz

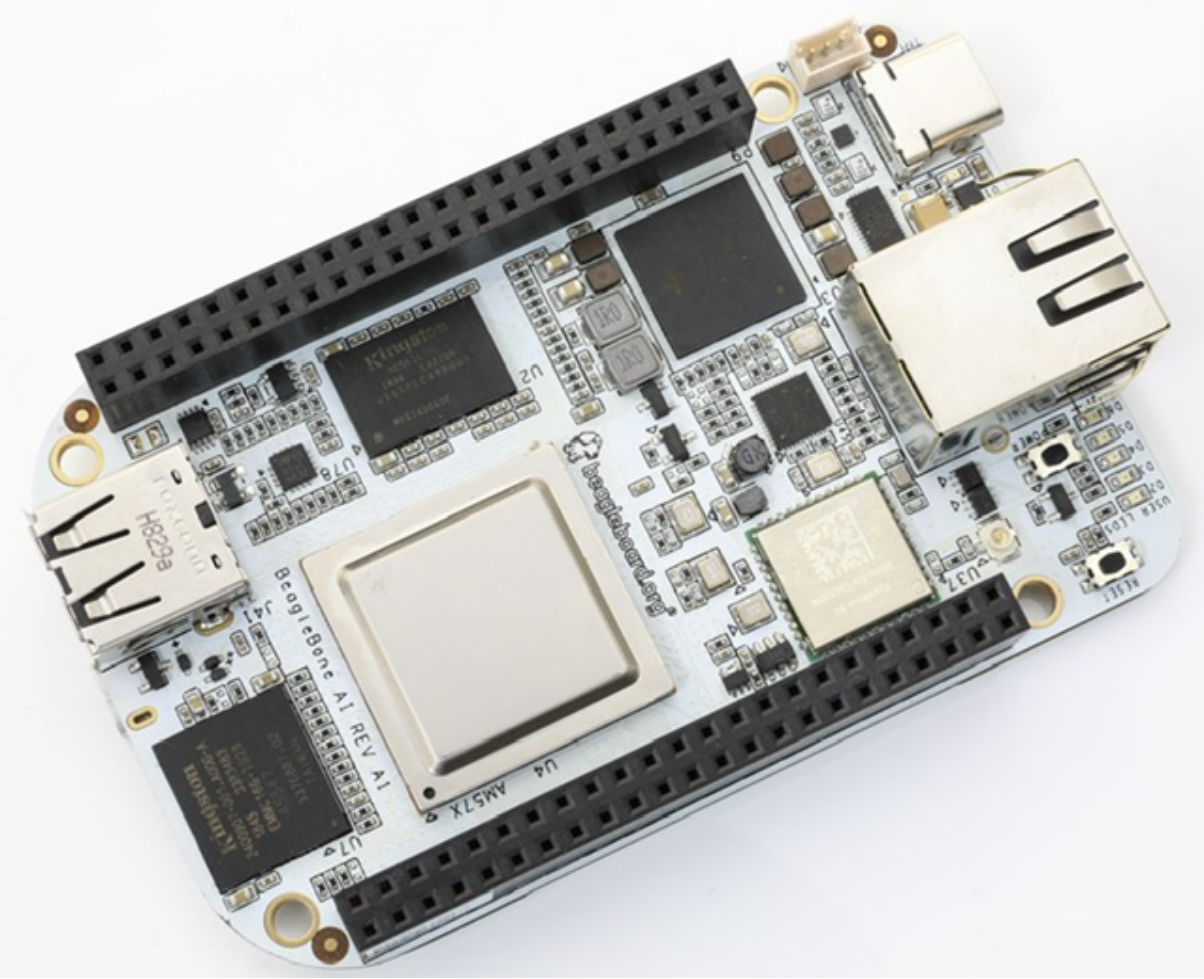
Production eMMC flash image: http://debian.beagleboard.org/images/am57xx-eMMC-flasher-debian-9.9-lxqt-armhf-2019-08-03-4gb.img.xz

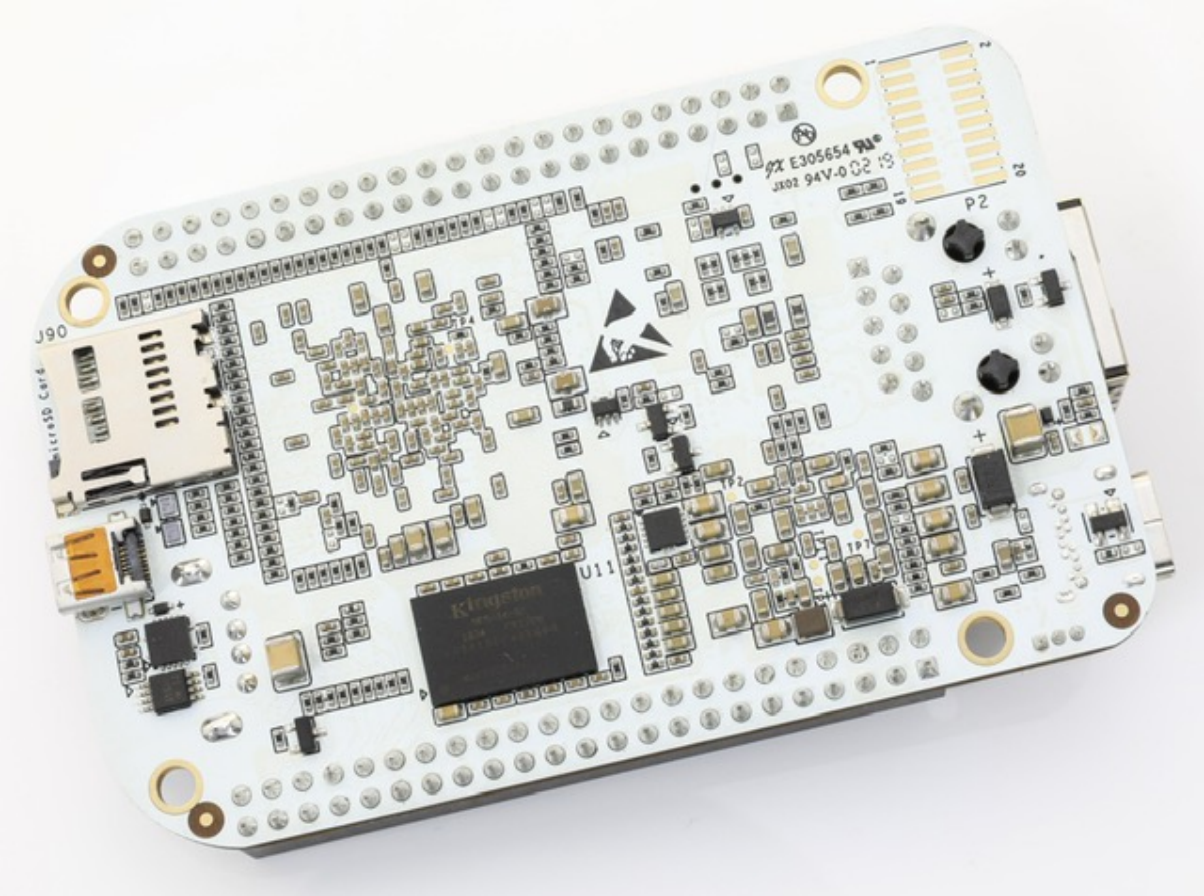
Rev A2

Proposed changes.

- HW: need pull-down on console uart RX line.
- HW: position of microSD may impact existing case designs.
- HW: P9.13 does not have a GPIO.
- HW: HDMI hotplug detection not working.
- HW: add extra DCAN.
- HW: wire mods required to enable JTAG.
- HW: Small I2C nvmem/eeprom for board identifier.

4.8.4 Pictures




E305654
94V-0
P2
J90
U11
Kingston

Chapter 5

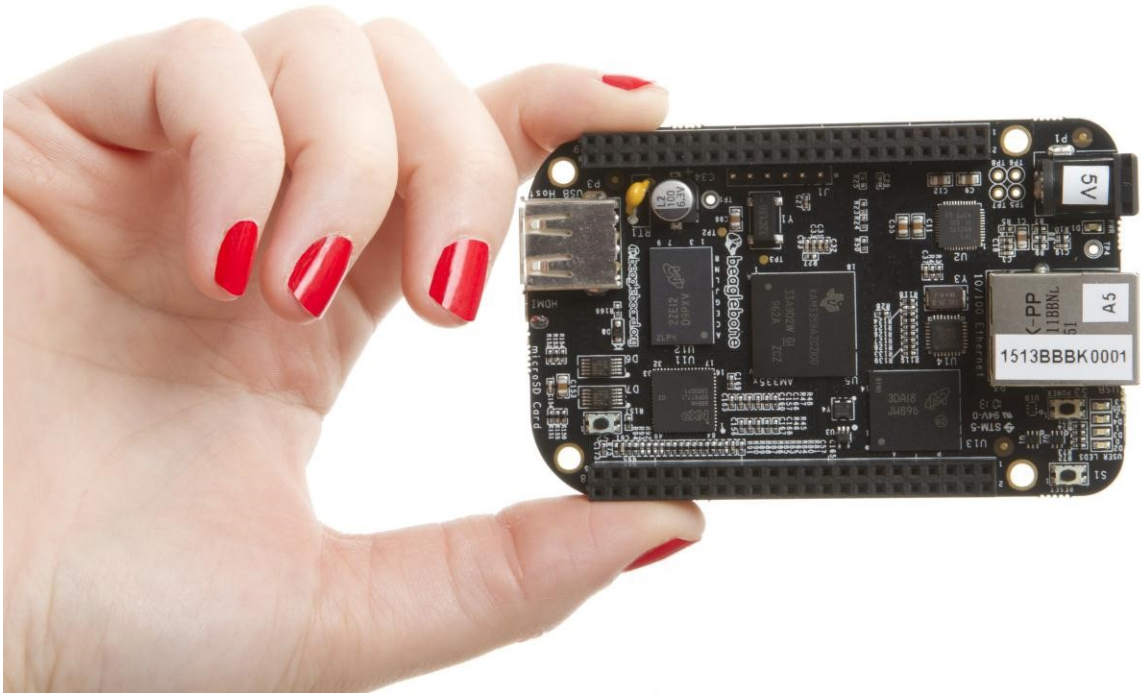
BeagleBone Black

BeagleBone Black is a low-cost, community-supported development platform for developers and hobbyists. Boot Linux in under 10 seconds and get started on development in less than 5 minutes with just a single USB cable.



License Terms

- This documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page
- For export, emissions and other compliance, see *Support Information*



5.1 Introduction

This document is the *System Reference Manual* for the BeagleBone Black and covers its use and design. The board will primarily be referred to in the remainder of this document simply as the board, although it may also be referred to as the BeagleBone Black as a reminder. There are also references to the original BeagleBone as well, and will be referenced as simply BeagleBone.

This design is subject to change without notice as we will work to keep improving the design as the product matures based on feedback and experience. Software updates will be frequent and will be independent of the hardware revisions and as such not result in a change in the revision number.

Make sure you check the docs repository frequently for the most up to date information.

https://git.beagleboard.org/docs/docs.beagleboard.io/-/tree/main/beaglebone-black

5.2 Change History

This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

5.2.1 Document Change History

Table 5.1: AsciiDoc Change History

Rev	Changes	Date	By
A4	Preliminary	January 4, 2013	GC
A5	Production release	January 8.2013	GC

continues on next page

Table 5.1 – continued from previous page

Rev	Changes	Date	By
A5.1	1. Added information on Power button and the battery access points. 2. Final production released version.	April 1 2013	GC
A5.2	1. Edited version. 2. Added numerous pictures of the Rev A5A board.	April 23 2013	GC
A5.3	1. Updated serial number locations. 2. Corrected the feature table for 4 UARTS 3. Corrected eMMC pin table to match other tables in the manual.	April 30, 2013	GC
A5.4	1. Corrected revision listed in section 2. Rev A5A is the initial production release. 2. Added all the locations of the serial numbers 3. Made additions to the compatibility list. 4. Corrected «table-7» for LED GPIO pins. 5. Fixed several typos. 6. Added some additional information about LDOs and Step-Down converters. 7. Added short section on HDMI.	May 12, 2013	GC
A5.5	1. Release of the A5B version. 2. The LEDS were dimmed by changing the resistors. 3. The serial termination mode was incorporated into the PCB.	May 20, 2013	GC
A5.6	1. Added information on Rev A5C 2. Added PRU/ICSS options to tables for P8 and P9. 3. Added section on USB Host Correct modes on «table-15». 4. Fixed a few typos	June 16, 2013	GC
A5.7	1. Updated assembly revision to A6. 2. PCB change to add buffer to the reset line and ground the oscillator GND pin. 3. Added resistor on PCB for connection of OSC_GND to board GND.	August 9, 2013	GC
A6	1. Added Rev A6 changes.	October 11, 2013	GC
A6A	1. Added Rev A6A changes	December 17, 2013	GC
B	1. Changed the processor to the AM3358BZCZ	January 20, 2013	GC

continues on next page

Table 5.1 – continued from previous page

Rev	Changes	Date	By
C	1. Changed the eMMC from 2GB to 4GB. 2. Added additional supplier to DDR2 and eMMC.	March 21,2014	GC
C.1	1. Added note to recommend powering off the board with the power	March 22, 2014	GC
C.2	Numerous community edits and format changes to asciidoc.	May 6, 2020	JK
C.3	Added information for board rev C3.	August 24, 2021	JK

5.2.2 Board Changes

Rev C3a

PCB revision C.

- New USB Type-A connector.

Rev C3

PCB revision C.

- Updated microSD card cage due to availability. See https://git.beagleboard.org/beagleboard/beaglebone-black/-/issues/6. Added series resistors and depopulated C5.
- Added reset option (GPIO1_8) for Ethernet PHY to avoid possible start-up issue. See https://git.beagleboard.org/beagleboard/beaglebone-black/-/issues/4.
- Added series resistors to MMC1 lines and depopulated C24.
- Connected pin A6 of J5 on U13 (eMMC IC) to DGND.
- Changed USB1_VBUS series resistor to 0 ohm.
- Change required PCB revision to C.

Initial boxes mistakenly say rev C1.

Rev C2

PCB revision B6.

- Update memories based on availability. See https://github.com/beagleboard/beaglebone-black/commit/74914bd01efeb61376ec3dda4bf9143ad2bb635c.
  - DDR3:
    - Kingston D2516EC4BXGGB-U
  - eMMC:
    - Kingston MMC04G-M627-X02U

Rev C1

PCB revision B6.

- Update memories based on availability. See https://github.com/beagleboard/beaglebone-black/commit/5787736d816832cc8cc9629d19f334b6a12e67f9.
  - DDR3:
    - Micron MT41K256M16TW-107:P
  - eMMC:
    - Micron MTFC4GACAJCN-1M WT
    - Kingston EMMC04G-S100-A08U

Rev C

- Changed the eMMC from 2GB to 4GB.

2GB devices are getting harder to get as they are being phased out. This required us to move to 4GB. We now have two sources for the device. This will however, require an increase in the price of the board.

Rev B

- Changed the processor to the AM3358BZCZ100.

Rev A6A

- Added connection from 32KHz OSC_GND to system ground and changed C106 to 1uF.
- Changes C25 to 2.2uF. This resolved an issue we were seeing in a few boards where the board would not boot in 1 in 20 tries.
- Change required PCB revision to B6.

Rev A6

- In random instances there could be a glitch in the SYS_RESETn signal from the processor where the SYS_RESETn signal was taken high for a momentary amount of time before it was supposed to. To prevent this, the signal was ORed with the PORZn (Power On reset).
- Noise issues were observed in other design where the clock oscillator was getting hit due to a suspected issue in ground bounce. A zero ohm resistor was added to connect the OSC_GND to the system ground.

There are no new features added as a result of these changes.

Rev A5C

We were seeing some fallout in production test where we were seeing some jitter on the HDMI display test. It started showing up on our second production run. R46, R47, R48 were changed to 0 ohm from 33 ohm. R45 was taken from 330 ohm to 22 ohm.

We do not know of any boards that were shipped with this issue as this issue was caught in production test. No impact on features or functionality resulted from this change.

Rev A5B

There is no operational difference between the Rev A5A and the Rev A5B. There were two changes made to the A5B version.

- Due to complaints about the brightness of the LEDs keeping people awake at night, the LEDs were dimmed. Resistors were changed from 820 ohms to 4.75K ohms.

- The PCB revision was updated to incorporate the hand mod that was being done on the board during manufacturing. The resistor was incorporated into the next revision of the PCB.

The highest supported resolution is now listed as 1920x1080@24Hz. This was not a result of any hardware changes but only updated software. The A5A version also supports this resolution.

Rev A5A

This is the initial production release of the board. We will be tracking changes from this point forward.

5.3 Connecting Up Your BeagleBone Black

This section provides instructions on how to hook up your board. Two scenarios will be discussed:

1. Tethered to a PC and
2. As a standalone development platform in a desktop PC configuration.

5.3.1 What's In the Box

In the box you will find three main items as shown in «figure-1».

- BeagleBone Black
- miniUSB to USB Type A Cable
- Instruction card with link to the support WIKI address.

This is sufficient for the tethered scenario and creates an out of box experience where the board can be used immediately with no other equipment needed.



Fig. 5.1: In the Box

5.3.2 Main Connection Scenarios

This section will describe how to connect the board for use. This section is basically a slightly more detailed description of the Quick Start Guide that came in the box. There is also a Quick Start Guide document on the board that should also be referred to. The intent here is that someone looking to purchase the board will be able to read this section and get a good idea as to what the initial set up will be like.

The board can be configured in several different ways, but we will discuss the two most common scenarios as described in the Quick Start Guide card that comes in the box.

- Tethered to a PC via the USB cable
  - Board is accessed as a storage drive
  - Or a RNDIS Ethernet connection.
- Standalone desktop
  - Display
  - Keyboard and mouse
  - External 5V power supply

Each of these configurations is discussed in general terms in the following sections.

For an up-to-date list of confirmed working accessories please go to BeagleBone_Black_Accessories

5.3.3 Tethered To A PC

In this configuration, the board is powered by the PC via the provided USB cable–no other cables are required. The board is accessed either as a USB storage drive or via the browser on the PC. You need to use either Firefox or Chrome on the PC, Internet Explorer will not work properly. «figure-2» shows this configuration.

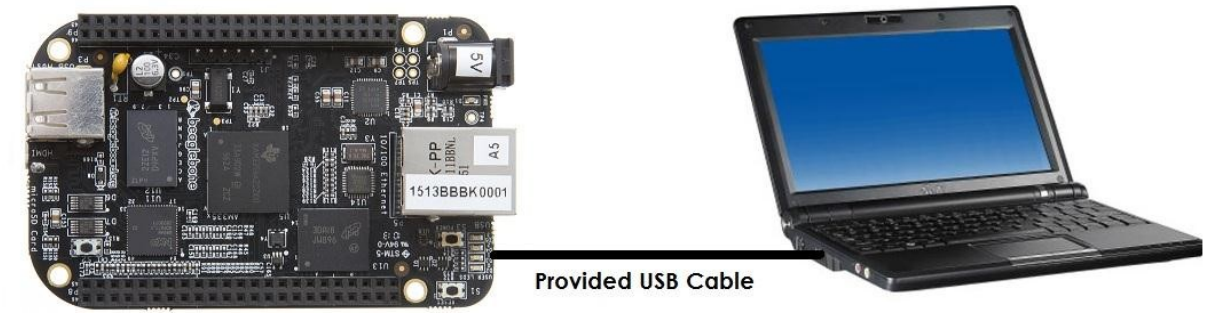


Fig. 5.2: Tethered Configuration

All the power for the board is provided by the PC via the USB cable. In some instances, the PC may not be able to supply sufficient power for the board. In that case, an external 5VDC power supply can be used, but this should rarely be necessary.

Connect the Cable to the Board

1. Connect the small connector on the USB cable to the board as shown in *figure-3*. The connector is on the bottom side of the board.
2. Connect the large connector of the USB cable to your PC or laptop USB port.
3. The board will power on and the power LED will be on as shown in figure below.
4. When the board starts to the booting process started by the process of applying power, the LEDs will come on in sequence as shown in *figure-5* below. It will take a few seconds for the status LEDs to come on, so be patient. The LEDs will be flashing in an erratic manner as it begins to boot the Linux kernel.

Accessing the Board as a Storage Drive

The board will appear around a USB Storage drive on your PC after the kernel has booted, which will take around 10 seconds. The kernel on the board needs to boot before the port gets enumerated. Once the board appears as a storage drive, do the following:

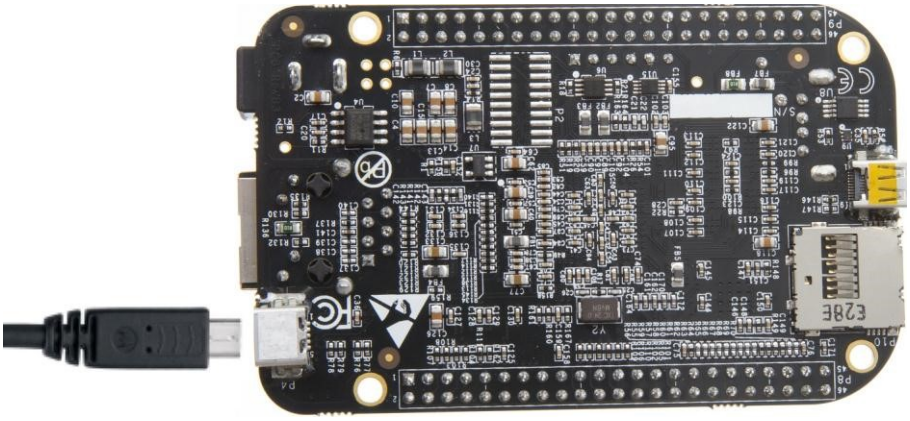


Fig. 5.3: USB Connection to the Board

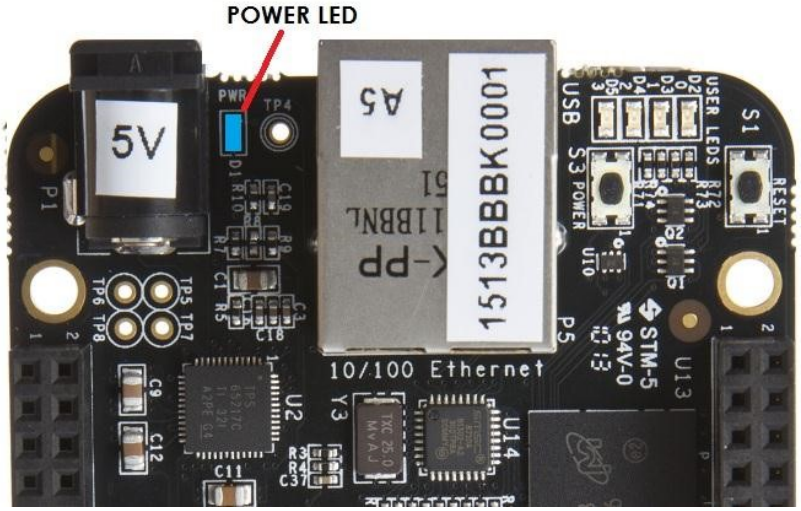


Fig. 5.4: Board Power LED

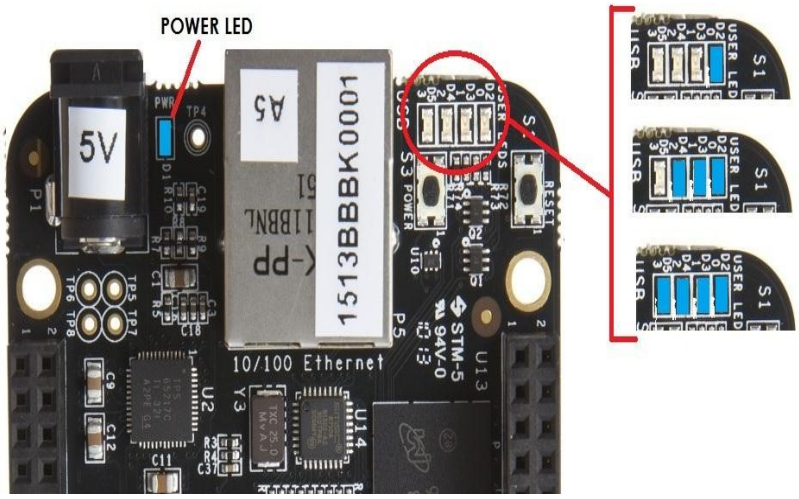


Fig. 5.5: Board Boot Status

1. Open the USB Drive folder.
2. Click on the file named *start.htm*
3. The file will be opened by your browser on the PC and you should get a display showing the Quick Start Guide.
4. Your board is now operational! Follow the instructions on your PC screen.

5.3.4 Standalone w/Display and Keyboard/Mouse

In this configuration, the board works more like a PC, totally free from any connection to a PC as shown in «figure-6». It allows you to create your code to make the board do whatever you need it to do. It will however require certain common PC accessories. These accessories and instructions are described in the following section.

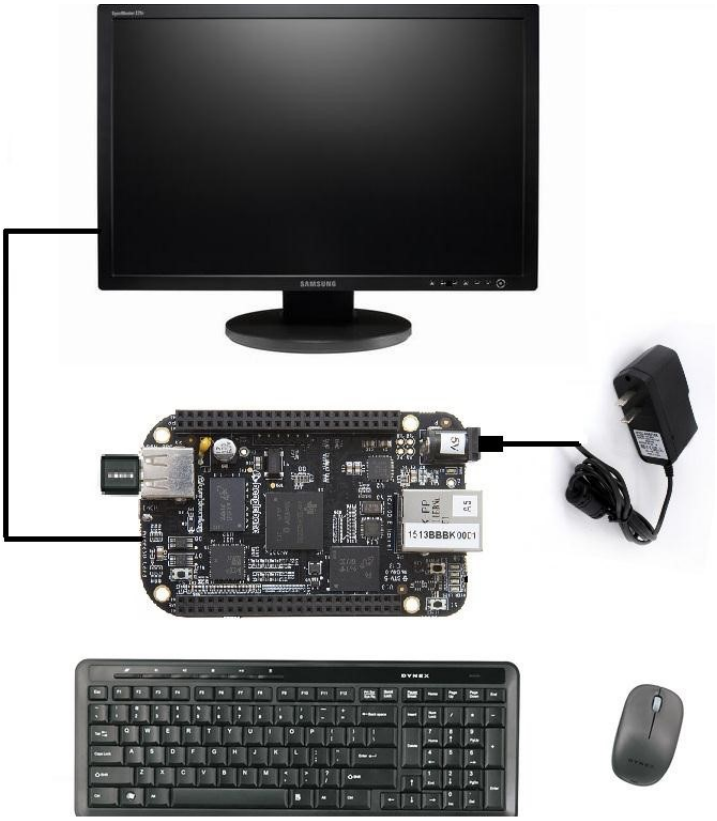


Fig. 5.6: Desktop Configuration

Optionally an Ethernet cable can also be used for network access.

Required Accessories

In order to use the board in this configuration, you will need the following accessories:

- 1 x 5VDC 1A power supply
- 1 x HDMI monitor or a DVI-D monitor. (*NOTE:* Only HDMI will give you audio capability).
- 1 x Micro HDMI to HDMI cable or a Micro HDMI to DVI-D adapter.
- 1 x USB wireless keyboard and mouse combo.
- 1 x USB HUB (OPTIONAL). The board has only one USB host port, so you may need to use a USB Hub if your keyboard and mouse requires two ports.

For an up-to-date list of confirmed working accessories please go to BeagleBone_Black_Accessories

Connecting Up the Board

1. Connect the big end of the HDMI cable as shown in *figure-7* to your HDMI monitor. Refer to your monitor Owner's Manual for the location of your HDMI port. If you have a DVI-D Monitor go to *Step 3*, otherwise proceed to *Step 4* .



Fig. 5.7: Connect microHDMI Cable to the Monitor

2. If you have a DVI-D monitor you must use a DVI-D to HDMI adapter in addition to your HDMI cable. An example is shown in *figure-8* below from two perspectives. If you use this configuration, you will not have audio support.



Fig. 5.8: DVI-D to HDMI Adapter

3. If you have a single wireless keyboard and mouse combination such as seen in *figure-9* below, you need to plug the receiver in the USB host port of the board as shown in *figure-10* .



Fig. 5.9: Wireless Keyboard and Mouse Combo

If you have a wired USB keyboard requiring two USB ports, you will need a HUB similar to the ones shown in figure below . You may want to have more than one port for other devices. Note that the board can only supply up to 500mA, so if you plan to load it down, it will need to be externally powered.

4. Connect the Ethernet Cable

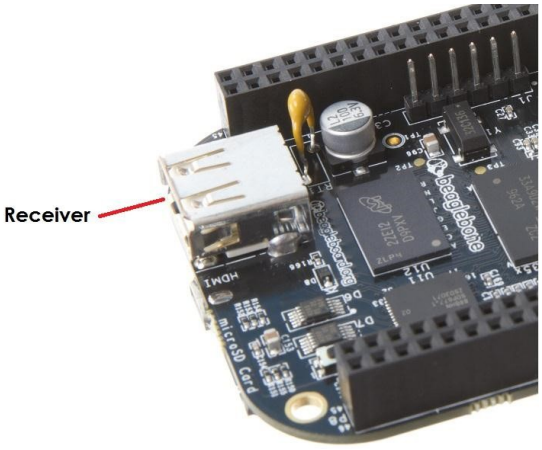


Fig. 5.10: Connect Keyboard and Mouse Receiver to the Board



Fig. 5.11: Keyboard and Mouse Hubs

If you decide you want to connect to your local area network, an Ethernet cable can be used. Connect the Ethernet Cable to the Ethernet port as shown in figure below . Any standard 100M Ethernet cable should work.

5. The final step is to plug in the DC power supply to the DC power jack as shown in figure below.
6. The cable needed to connect to your display is a microHDMI to HDMI. Connect the microHDMI connector end to the board at this time. The connector is on the bottom side of the board as shown in *figure-14* below.

The connector is fairly robust, but we suggest that you not use the cable as a leash for your Beagle. Take proper care not to put too much stress on the connector or cable.

7. Booting the Board

As soon as the power is applied to the board, it will start the booting up process. When the board starts to boot the LEDs will come on in sequence as shown in *figure-15* below. It will take a few seconds for the status LEDs to come on, so be patient. The LEDs will be flashing in an erratic manner as it boots the Linux kernel.

While the four user LEDs can be overwritten and used as desired, they do have specific meanings in the image that is shipped with the board once the Linux kernel has booted.

- *USER0* is the heartbeat indicator from the Linux kernel.
- *USER1* turns on when the microSD card is being accessed
- *USER2* is an activity indicator. It turns on when the kernel is not in the idle loop.
- *USER3* turns on when the onboard eMMC is being accessed.

8. A Booted System
    a. The board will have a mouse pointer appear on the screen as it enters the Linux boot step. You may have to move the physical mouse to get the mouse pointer to appear. The system can come up in the suspend mode with the HDMI port in a sleep mode.

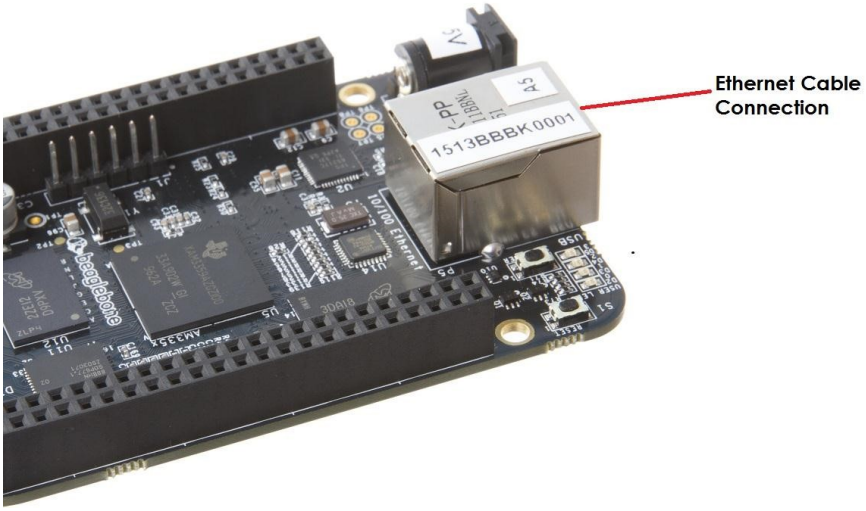


Fig. 5.12: Ethernet Cable Connection

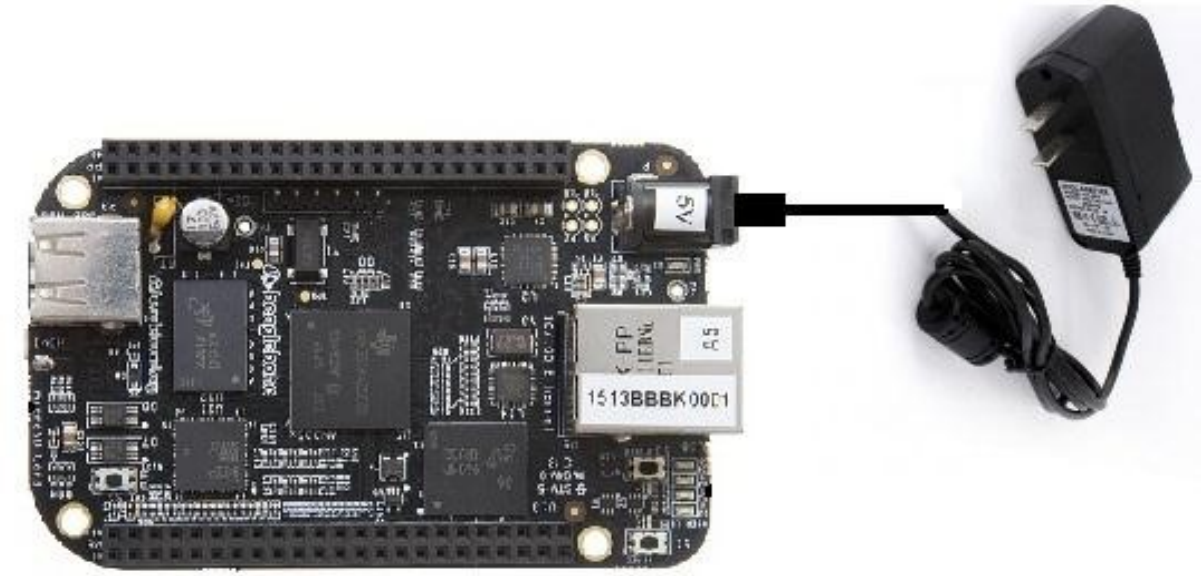


Fig. 5.13: External DC Power

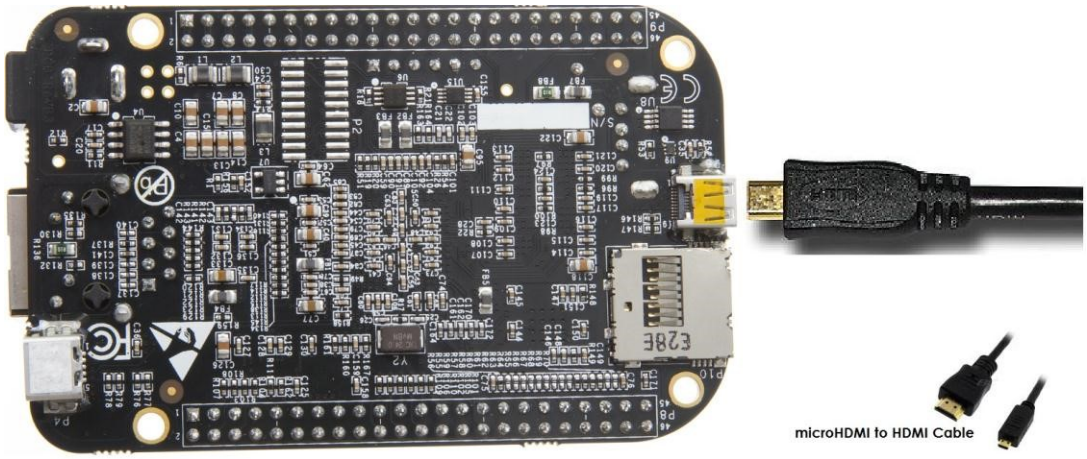


Fig. 5.14: Connect microHDMI Cable to the Board

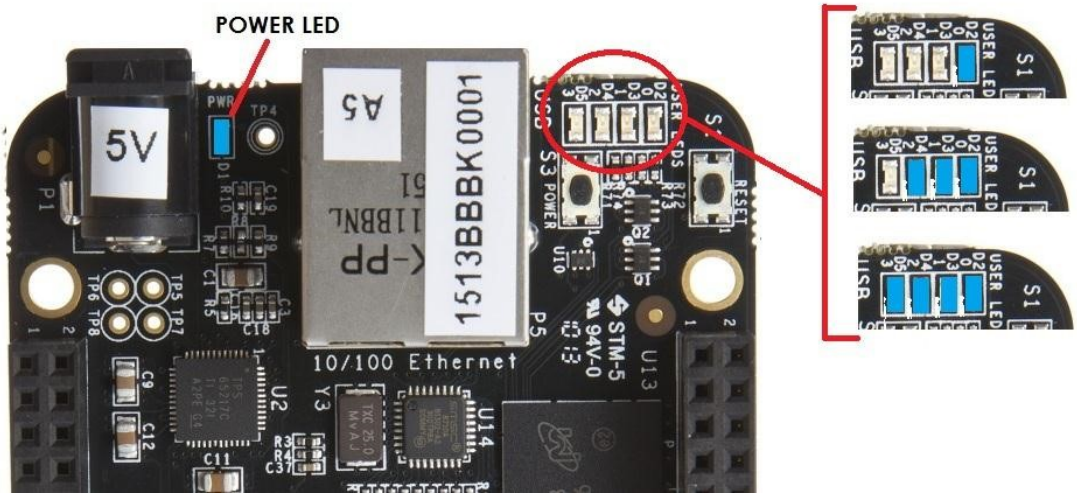


Fig. 5.15: Board Boot Status

b. After a minute or two a login screen will appear. You do not have to do anything at this point.

c. After a minute or two the desktop will appear. It should be similar to the one shown in figure-1. HOWEVER, it will change from one release to the next, so do not expect your system to look exactly like the one in the figure, but it will be very similar.

d. And at this point you are ready to go! *figure-16* shows the desktop after booting.



Fig. 5.16: Desktop Screen

9. Powering Down

   A. Press the power button momentarily.

   B. The system will power down automatically.

   C. Remove the power jack.

5.4 BeagleBone Black Overview

The BeagleBone Black is the latest addition to the BeagleBoard.org family and like its predecessors, is designed to address the Open Source Community, early adopters, and anyone interested in a low cost ARM Cortex-A8 based processor.

It has been equipped with a minimum set of features to allow the user to experience the power of the processor and is not intended as a full development platform as many of the features and interfaces supplied by the processor are not accessible from the BeagleBone Black via onboard support of some interfaces. It is not a complete product designed to do any particular function. It is a foundation for experimentation and learning how to program the processor and to access the peripherals by the creation of your own software and hardware.

It also offers access to many of the interfaces and allows for the use of add-on boards called capes, to add many different combinations of features. A user may also develop their own board or add their own circuitry.

BeagleBone Black is manufactured and warranted by partners listed at https://beagleboard.org/logo for the benefit of the community and its supporters.

Jason Kridner of Texas Instruments handles the community promotions and is the spokesman for BeagleBoard.org.

The board is designed by Gerald Coley of EmProDesign, a charter member of the BeagleBoard.org community.

The PCB layout up through PCB revision B was done by Circuitco and Circuitco is the sole funder of its development and transition to production. Later PCB revisions have been made by Embest, a subsidiary of Avent.

The Software is written and supported by the thousands of community members, including Jason Kridner, employee of Texas Instruments, and Robert Nelson, employee of DigiKey.

5.4.1 BeagleBone Compatibility

The board is intended to be compatible with the original BeagleBone as much as possible. There are several areas where there are differences between the two designs. These differences are listed below, along with the reasons for the differences.

- Sitara AM3358BZCZ100, 1GHZ, processor.
  - Sorry, we just had to make it faster.
- 512MB DDR3L
  - *Cost reduction*
  - Performance boost
  - Memory size increase
  - Lower power
- No Serial port by default
  - *Cost reduction*
  - Can be added by buying a TTL to USB Cable that is widely available
  - Single largest cost reduction action taken
- No JTAG emulation over USB
  - *Cost reduction* JTAG header is not populated, but can easily be mounted.
  - EEPROM Reduced from 32KB to 4KB
  - *Cost Reduction*
- Onboard Managed NAND (eMMC)
  - 4GB

  - *Cost reduction*
  - Performance boost x8 vs. x4 bits
  - Performance boost due to deterministic properties vs. microSD card
- GPMC bus may not be accessible from the expansion headers in some cases
  - Result of eMMC on the main board
  - Signals are still routed to the expansion connector
  - If eMMC is not used, signals can be used via expansion if eMMC is held in reset
- There may be 10 less GPIO pins available
  - Result of eMMC
  - If eMMC is not used, could still be used
- The power expansion header, for battery and backlight, has been removed
  - _*Cost reduction* , space reduction
  - Four pins were added to provide access to the battery charger function.
- HDMI interface onboard
  - Feature addition
  - Audio and video capable
  - Micro HDMI
- No three function USB cable
  - *Cost reduction*
- GPIO3_21 has a 24.576 MHZ clock on it.
  - This is required by the HDMI Framer for Audio purposes. We needed to run a clock into the processor to generate the correct clock frequency. The pin on the processor was already routed to the expansion header. In order not to remove this feature on the expansion header, it was left connected. In order to use the pin as a GPIO pin, you need to disable the clock. While this disables audio to the HDMI, the fact that you want to use this pin for something else, does the same thing.

5.4.2 BeagleBone Black Features and Specification

This section covers the specifications and features of the board and provides a high level description of the major components and interfaces that make up the board. table below provides a list of the features.

Table 5.2: BeagleBone Black Features

	Feature
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS
Graphics Engine	SGX530 3D, 20M Polygons/S
SDRAM Memory	512MB DDR3L 800MHZ
Onboard Flash	4GB, 8bit Embedded MMC
PMIC	TPS65217C PMIC regulator and one additional LDO.
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header
Power Source	miniUSB USB or DC Jack
PCB	3.4” x 2.1”
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB

continues on next page

Table 5.2 – continued from previous page

	Feature
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated
Ethernet	10/100, RJ45
SD/MMC Connector	microSD , 3.3V
User Input	1. Reset Button 2. Boot Button 3. Power Button
Video Out	1. 16b HDMI, 1280x1024 (MAX) 2. 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support
Audio	Via HDMI Interface, Stereo
Expansion Connectors	1. Power 5V, 3.3V , VDD_ADC(1.8V) 2. 3.3V I/O on all signals 3. McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 4. AIN _(1.8V MAX)_, 4 Timers, 4 Serial Ports, CAN0, 5. EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)
Weight	1.4 oz (39.68 grams)
Power	Refer to *section-6-1-7*

5.4.3 Board Component Locations

This section describes the key components on the board. It provides information on their location and function. Familiarize yourself with the various components on the board.

Connectors, LEDs, and Switches

figure below shows the locations of the connectors, LEDs, and switches on the PCB layout of the board.

- *DC Power* is the main DC input that accepts 5V power.
- *Power Button* alerts the processor to initiate the power down sequence and is used to power down the board.
- *10/100 Ethernet* is the connection to the LAN.
- *Serial Debug* is the serial debug port.
- *USB Client* is a miniUSB connection to a PC that can also power the board.
- *BOOT switch* can be used to force a boot from the microSD card if the power is cycled on the board, removing power and reapplying the power to the board..
- There are four blue **LED**'s that can be used by the user.
- *Reset Button* allows the user to reset the processor.
- *microSD* slot is where a microSD card can be installed.
- *microHDMI* connector is where the display is connected to.
- *USB Host* can be connected different USB interfaces such as Wi-Fi, BT, Keyboard, etc.

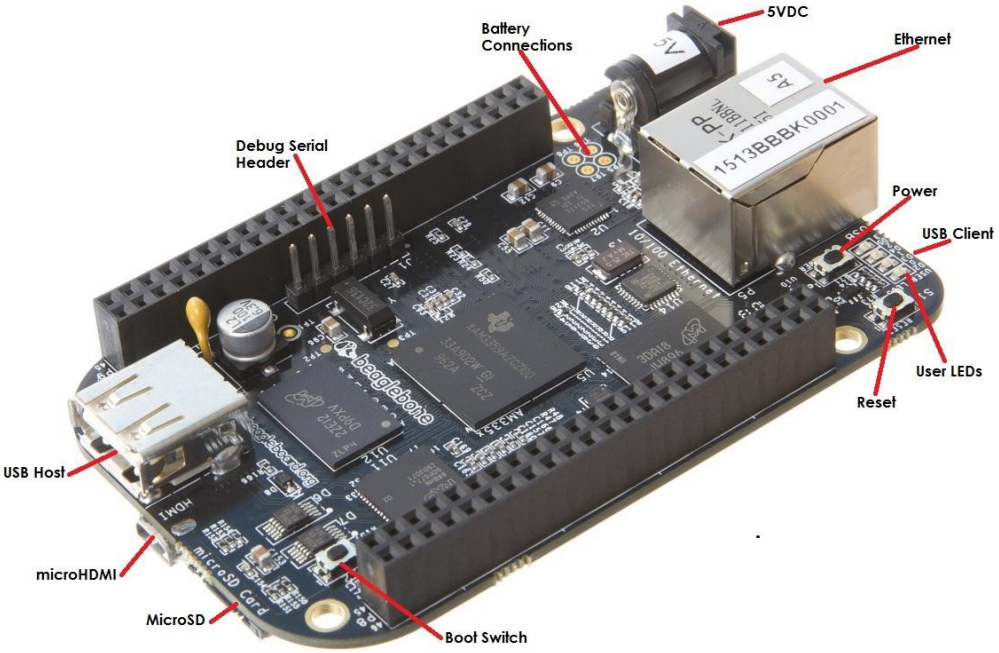


Fig. 5.17: Connectors, LEDs and Switches

Key Components

figure below shows the locations of the key components on the PCB layout of the board.

- *Sitara AM3358BZCZ100* is the processor for the board.
- *Micron 512MB DDR3L* or**Kingston 512mB DDR3** is the Dual Data Rate RAM memory.
- *TPS65217C PMIC* provides the power rails to the various components on the board.
- *SMSC Ethernet PHY* is the physical interface to the network.
- *Micron eMMC* is an onboard MMC chip that holds up to 4GB of data.
- *HDMI* Framer provides control for an HDMI or DVI-D display with an adapter.

5.5 BeagleBone Black High Level Specification

This section provides the high level specification of the BeagleBone Black.

5.5.1 Block Diagram

5.5.2 Processor

The revision B and later boards have moved to the Sitara AM3358BZCZ100 device.

5.5.3 Memory

Described in the following sections are the three memory devices found on the board.

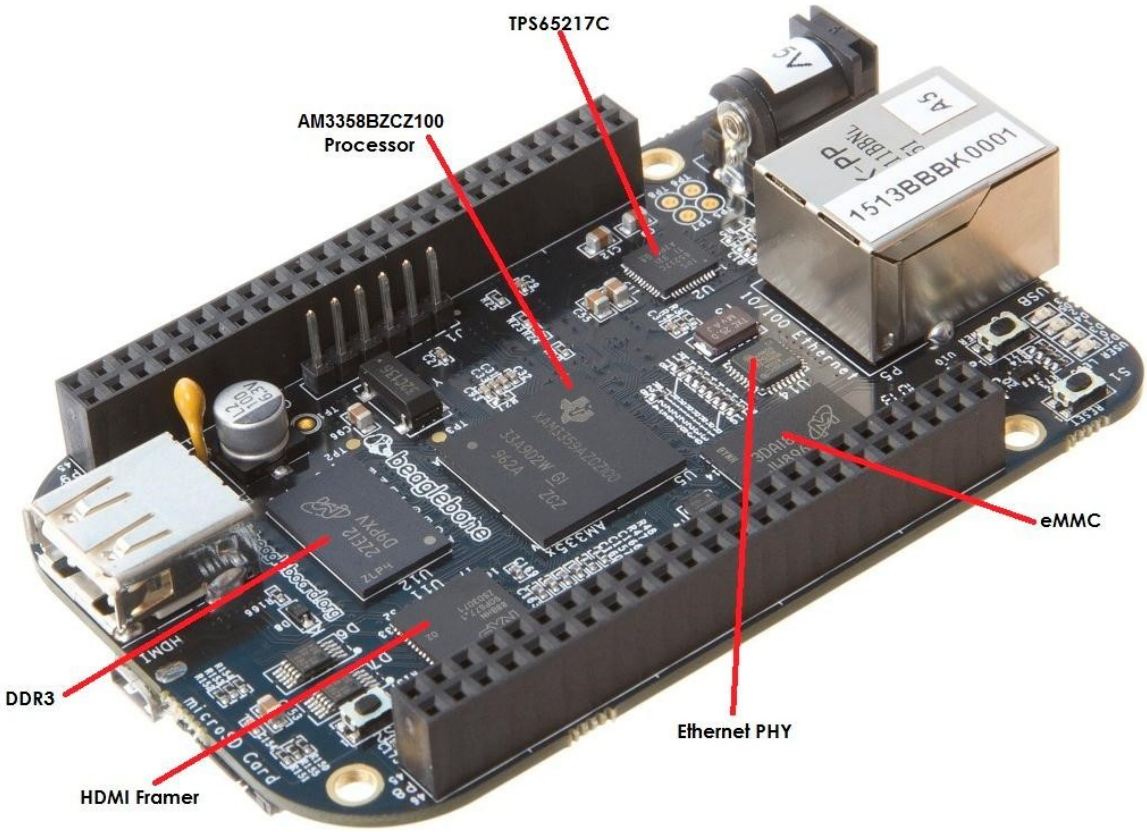


Fig. 5.18: Key Components

512MB DDR3L

A single 256Mb x16 DDR3L 4Gb (512MB) memory device is used. The memory used is one of two devices:

- MT41K256M16HA-125 from Micron
- D2516EC4BXGGB from Kingston

It will operate at a clock frequency of 400MHz yielding an effective rate of 800MHZ on the DDR3L bus allowing for 1.6GB/S of DDR3L memory bandwidth.

4KB EEPROM

A single 4KB EEPROM is provided on I2C0 that holds the board information. This information includes board name, serial number, and revision information. This is the not the same as the one used on the original BeagleBone. The device was changed for cost reduction reasons. It has a test point to allow the device to be programmed and otherwise to provide write protection when not grounded.

4GB Embedded MMC

A single 4GB embedded MMC (eMMC) device is on the board. The device connects to the MMC1 port of the processor, allowing for 8bit wide access. Default boot mode for the board will be MMC1 with an option to change it to MMC0, the SD card slot, for booting from the SD card as a result of removing and reapplying the power to the board. Simply pressing the reset button will not change the boot mode. MMC0 cannot be used in 8Bit mode because the lower data pins are located on the pins used by the Ethernet port. This does not interfere with SD card operation but it does make it unsuitable for use as an eMMC port if the 8 bit feature is needed.

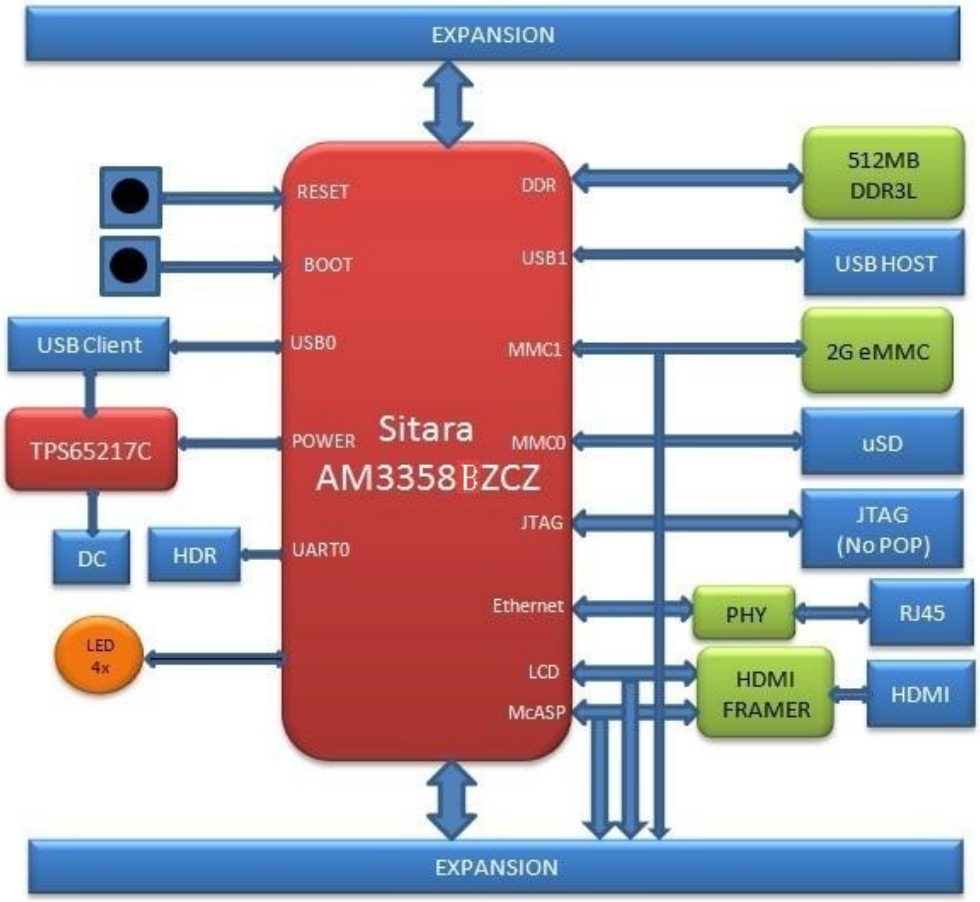


Fig. 5.19: BeagleBone Black Key Components

MicroSD Connector

The board is equipped with a single microSD connector to act as the secondary boot source for the board and, if selected as such, can be the primary boot source. The connector will support larger capacity microSD cards. The microSD card is not provided with the board. Booting from MMC0 will be used to flash the eMMC in the production environment or can be used by the user to update the SW as needed.

Boot Modes

As mentioned earlier, there are four boot modes:

- **eMMC Boot:** This is the default boot mode and will allow for the fastest boot time and will enable the board to boot out of the box using the pre-flashed OS image without having to purchase an microSD card or an microSD card writer.
- **SD Boot:** This mode will boot from the microSD slot. This mode can be used to override what is on the eMMC device and can be used to program the eMMC when used in the manufacturing process or for field updates.
- **Serial BooT:** This mode will use the serial port to allow downloading of the software direct. A separate USB to serial cable is required to use this port.
- **USB Boot:** This mode supports booting over the USB port.

Software to support USB and serial boot modes is not provided by beagleboard.org.Please contact TI for support of this feature.

A switch is provided to allow switching between the modes.

- Holding the boot switch down during a removal and reapplication of power without a microSD card inserted will force the boot source to be the USB port and if nothing is detected on the USB client port, it will go to the serial port for download.
- Without holding the switch, the board will boot try to boot from the eMMC. If it is empty, then it will try booting from the microSD slot, followed by the serial port, and then the USB port.
- If you hold the boot switch down during the removal and reapplication of power to the board, and you have a microSD card inserted with a bootable image, the board will boot from the microSD card.

NOTE: Pressing the RESET button on the board will NOT result in a change of the_ _boot mode. You MUST remove power and reapply power to change the boot mode.The boot pins are sampled during power on reset from the PMIC to the processor.The reset button on the board is a warm reset only and will not force a boot mode change.

5.5.4 Power Management

The *TPS65217C* power management device is used along with a separate LDO to provide power to the system. The**TPS65217C** version provides for the proper voltages required for the DDR3L. This is the same device as used on the original BeagleBone with the exception of the power rail configuration settings which will be changed in the internal EEPROM to the *TPS65217C* to support the new voltages.

DDR3L requires 1.5V instead of 1.8V on the DDR2 as is the case on the original BeagleBone. The 1.8V regulator setting has been changed to 1.5V for the DDR3L. The LDO3 3.3V rail has been changed to 1.8V to support those rails on the processor. LDO4 is still 3.3V for the 3.3V rails on the processor. An external *LDOTLV70233* provides the 3.3V rail for the rest of the board.

5.5.5 PC USB Interface

The board has a miniUSB connector that connects the USB0 port to the processor. This is the same connector as used on the original BeagleBone.

5.5.6 Serial Debug Port

Serial debug is provided via UART0 on the processor via a single 1x6 pin header. In order to use the interface a USB to TTL adapter will be required. The header is compatible with the one provided by FTDI and can be purchased for about $$12 to $$20 from various sources. Signals supported are TX and RX. None of the handshake signals are supported.

5.5.7 USB1 Host Port

On the board is a single USB Type A female connector with full LS/FS/HS Host support that connects to USB1 on the processor. The port can provide power on/off control and up to 500mA of current at 5V. Under USB power, the board will not be able to supply the full 500mA, but should be sufficient to supply enough current for a lower power USB device supplying power between 50 to 100mA.

You can use a wireless keyboard/mouse configuration or you can add a HUB for standard keyboard and mouse interfacing.

5.5.8 Power Sources

The board can be powered from four different sources:

- A USB port on a PC
- A 5VDC 1A power supply plugged into the DC connector.
- A power supply with a USB connector.
- Expansion connectors

The USB cable is shipped with each board. This port is limited to 500mA by the Power Management IC. It is possible to change the settings in the *TPS65217C* to increase this current, but only after the initial boot. And, at that point the PC most likely will complain, but you can also use a dual connector USB cable to the PC to get to 1A.

The power supply is not provided with the board but can be easily obtained from numerous sources. A 1A supply is sufficient to power the board, but if there is a cape plugged into the board or you have a power hungry device or hub plugged into the host port, then more current may needed from the DC supply.

Power routed to the board via the expansion header could be provided from power derived on a cape. The DC supply should be well regulated and 5V +/-.25V.

5.5.9 Reset Button

When pressed and released, causes a reset of the board. The reset button used on the BeagleBone Black is a little larger than the one used on the original BeagleBone. It has also been moved out to the edge of the board so that it is more accessible.

5.5.10 Power Button

A power button is provided near the reset button close to the Ethernet connector. This button takes advantage of the input to the PMIC for power down features. While a lot of capes have a button, it was decided to add this feature to the board to ensure everyone had access to some new features. These features include:

- Interrupt is sent to the processor to facilitate an orderly shutdown to save files and to un-mount drives.
- Provides ability to let processor put board into a sleep mode to save power.
- Can alert processor to wake up from sleep mode and restore state before sleep was entered.

If you hold the button down longer than 8 seconds, the board will power off if you release the button when the power LED turns off. If you continue to hold it, the board will power back up completing a power cycle.

We recommend that you use this method to power down the board. It will also help prevent contamination of the SD card or the eMMC.

If you do not remove the power jack, you can press the button again and the board will power up.

5.5.11 Indicators

There are a total of five blue LEDs on the board.

- One blue power LED indicates that power is applied and the power management IC is up. If this LED flashes when applying power, it means that an excess current flow was detected and the PMIC has shut down.
- Four blue LEDs that can be controlled via the SW by setting GPIO pins.

In addition, there are two LEDs on the RJ45 to provide Ethernet status indication. One is yellow (100M Link up if on) and the other is green (Indicating traffic when flashing).

5.5.12 CTI JTAG Header

A place for an optional 20 pin CTI JTAG header is provided on the board to facilitate the SW development and debugging of the board by using various JTAG emulators. This header is not supplied standard on the board. To use this, a connector will need to be soldered onto the board.

If you need the JTAG connector you can solder it on yourself. No other components are needed. The connector is made by Samtec and the part number is FTR-110-03-G-D-06. You can purchase it from http://www.digikey.com/

5.5.13 HDMI Interface

A single HDMI interface is connected to the 16 bit LCD interface on the processor. The 16b interface was used to preserve as many expansion pins as possible to allow for use by the user. The NXP TDA19988BHN is used to convert the LCD interface to HDMI and convert the audio as well. The signals are still connected to the expansion headers to enable the use of LCD expansion boards or access to other functions on the board as needed.

The HDMI device does not support HDCP copy protection. Support is provided via EDID to allow the SW to identify the compatible resolutions. Currently the following resolutions are supported via the software:

- 1280 x 1024
- 1440 x 900
- 1024 x 768
- 1280 x 720

5.5.14 Cape Board Support

The BeagleBone Black has the ability to accept up to four expansion boards or capes that can be stacked onto the expansion headers. The word cape comes from the shape of the board as it is fitted around the Ethernet connector on the main board. This notch acts as a key to ensure proper orientation of the cape.

The majority of capes designed for the original BeagleBone will work on the BeagleBone Black. The two main expansion headers will be populated on the board. There are a few exceptions where certain capabilities may not be present or are limited to the BeagleBone Black. These include:

- GPMC bus may NOT be available due to the use of those signals by the eMMC. If the eMMC is used for booting only and the file system is on the microSD card, then these signals could be used.

- Another option is to use the microSD or serial boot modes and not use the eMMC.
- The power expansion header is not on the BeagleBone Black so those functions are not supported.

For more information on cape support refer to *BeagleBone Black Mechanical* section.

5.6 Detailed Hardware Design

This section provides a detailed description of the Hardware design. This can be useful for interfacing, writing drivers, or using it to help modify specifics of your own design.

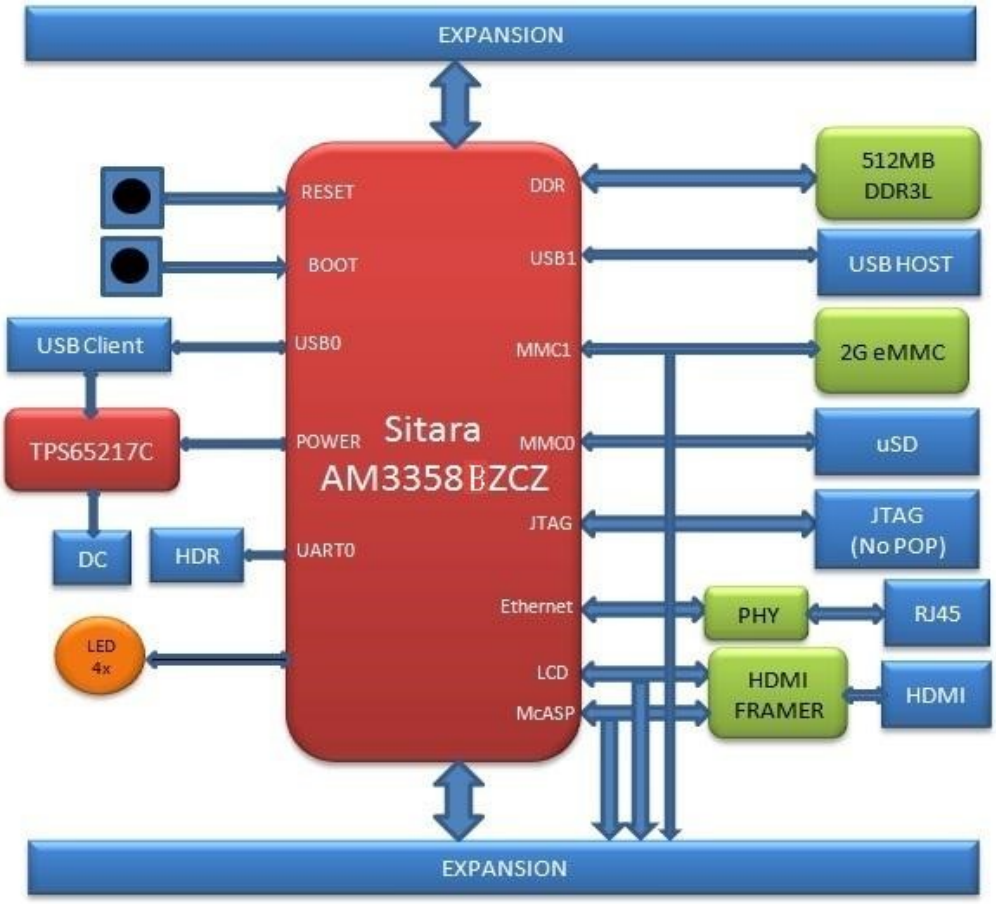


Fig. 5.20: BeagleBone Black Block Diagram

5.6.1 Power Section

This section describes the power section of the design and all the functions performed by the *TPS65217C*.

TPS65217C PMIC

The main Power Management IC (PMIC) in the system is the *TPS65217C* which is a single chip power management IC consisting of a linear dual-input power path, three step-down converters, and four LDOs. LDO stands for Low Drop Out. If you want to know more about an LDO, you can go to http://en.wikipedia.org/wiki/Low-dropout_regulator .If you want to learn more about step-down converters, you can go to

http://en.wikipedia.org/wiki/DC-to-DC_converter

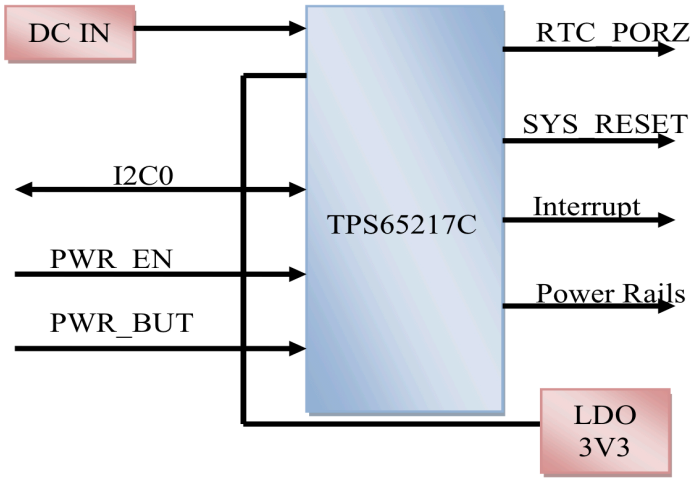


Fig. 5.21: High Level Power Block Diagram

The system is supplied by a USB port or DC adapter. Three high-efficiency 2.25MHz step-down converters are targeted at providing the core voltage, MPU, and memory voltage for the board.

The step-down converters enter a low power mode at light load for maximum efficiency across the widest possible range of load currents. For low-noise applications the devices can be forced into fixed frequency PWM using the I2C interface. The step-down converters allow the use of small inductors and capacitors to achieve a small footprint solution size.

LDO1 and LDO2 are intended to support system standby mode. In normal operation, they can support up to 100mA each. LDO3 and LDO4 can support up to 285mA each.

By default only LDO1 is always ON but any rail can be configured to remain up in SLEEP state. In particular the DCDC converters can remain up in a low-power PFM mode to support processor suspend mode. The *TPS65217C* offers flexible power-up and power-down sequencing and several house-keeping functions such as power-good output, pushbutton monitor, hardware reset function and temperature sensor to protect the battery.

For more information on the *TPS65217C*, refer to http://www.ti.com/product/tps65217C

DC Input

A 5VDC supply can be used to provide power to the board. The power supply current depends on how many and what type of add-on boards are connected to the board. For typical use, a 5VDC supply rated at 1A should be sufficient. If heavier use of the expansion headers or USB host port is expected, then a higher current supply will be required.

The connector used is a 2.1MM center positive x 5.5mm outer barrel. The 5VDC rail is connected to the expansion header. It is possible to power the board via the expansion headers from an add-on card. The 5VDC is also available for use by the add-on cards when the power is supplied by the 5VDC jack on the board.

USB Power

The board can also be powered from the USB port. A typical USB port is limited to 500mA max. When powering from the USB port, the VDD_5V rail is not provided to the expansion headers, so capes that require the 5V rail to supply the cape direct, bypassing the *TPS65217C*, will not have that rail available for use. The 5VDC supply from the USB port is provided on the SYS_5V, the one that comes from the**TPS65217C**, rail of the expansion header for use by a cape. *Figure 24* is the connection of the USB power input on the PMIC.

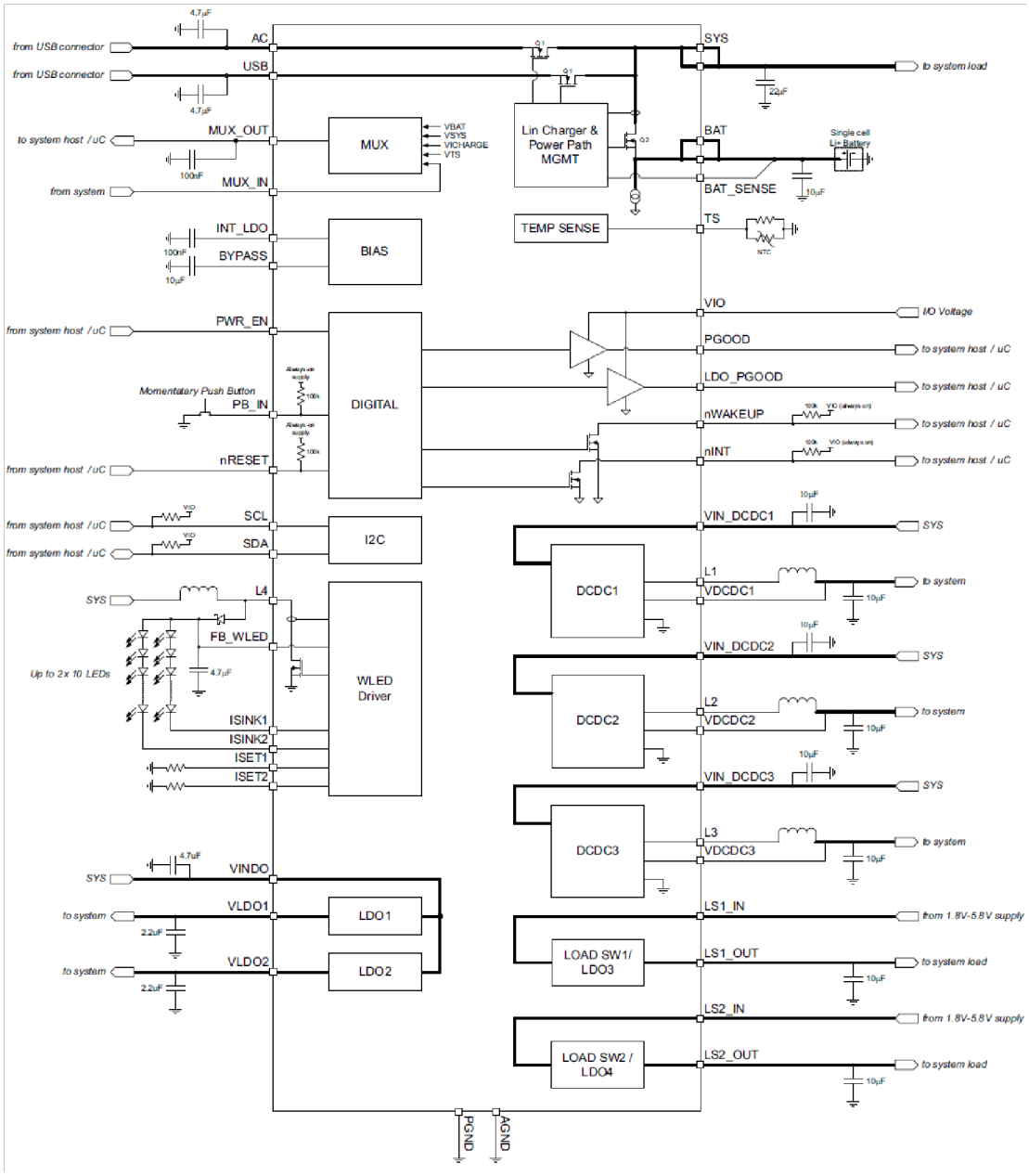


Fig. 5.22: TPS65217C Block Diagram

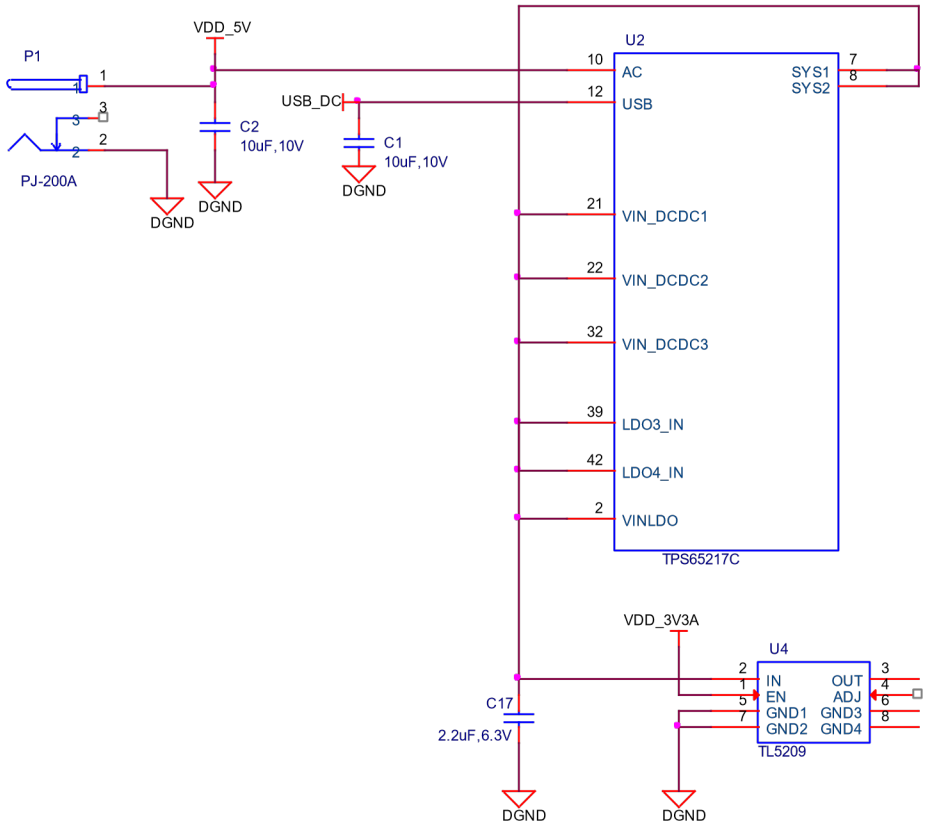


Fig. 5.23: TPS65217 DC Connection

Power Selection

The selection of either the 5VDC or the USB as the power source is handled internally to the *TPS65217C* and automatically switches to 5VDC power if both are connected. SW can change the power configuration via the I2C interface from the processor. In addition, the SW can read the**TPS65217C** and determine if the board is running on the 5VDC input or the USB input. This can be beneficial to know the capability of the board to supply current for things like operating frequency and expansion cards.

It is possible to power the board from the USB input and then connect the DC power supply. The board will switch over automatically to the DC input.

Power Button

A power button is connected to the input of the *TPS65217C*. This is a momentary switch, the same type of switch used for reset and boot selection on the board.

If you push the button the *TPS65217C* will send an interrupt to the processor. It is up to the processor to then pull the**PMIC_POWER_EN** pin low at the correct time to power down the board. At this point, the PMIC is still active, assuming that the power input was not removed. Pressing the power button will cause the board to power up again if the processor puts the board in the power off mode.

In power off mode, the RTC rail is still active, keeping the RTC powered and running off the main power input. If you remove that power, then the RTC will not be powered. You also have the option of using the battery holes on the board to connect a battery if desired as discussed in the next section.

If you push and hold the button for greater than 8 seconds, the PMIC will power down. But you must release the button when the power LED turns off. Holding the button past that point will cause the board to power cycle.

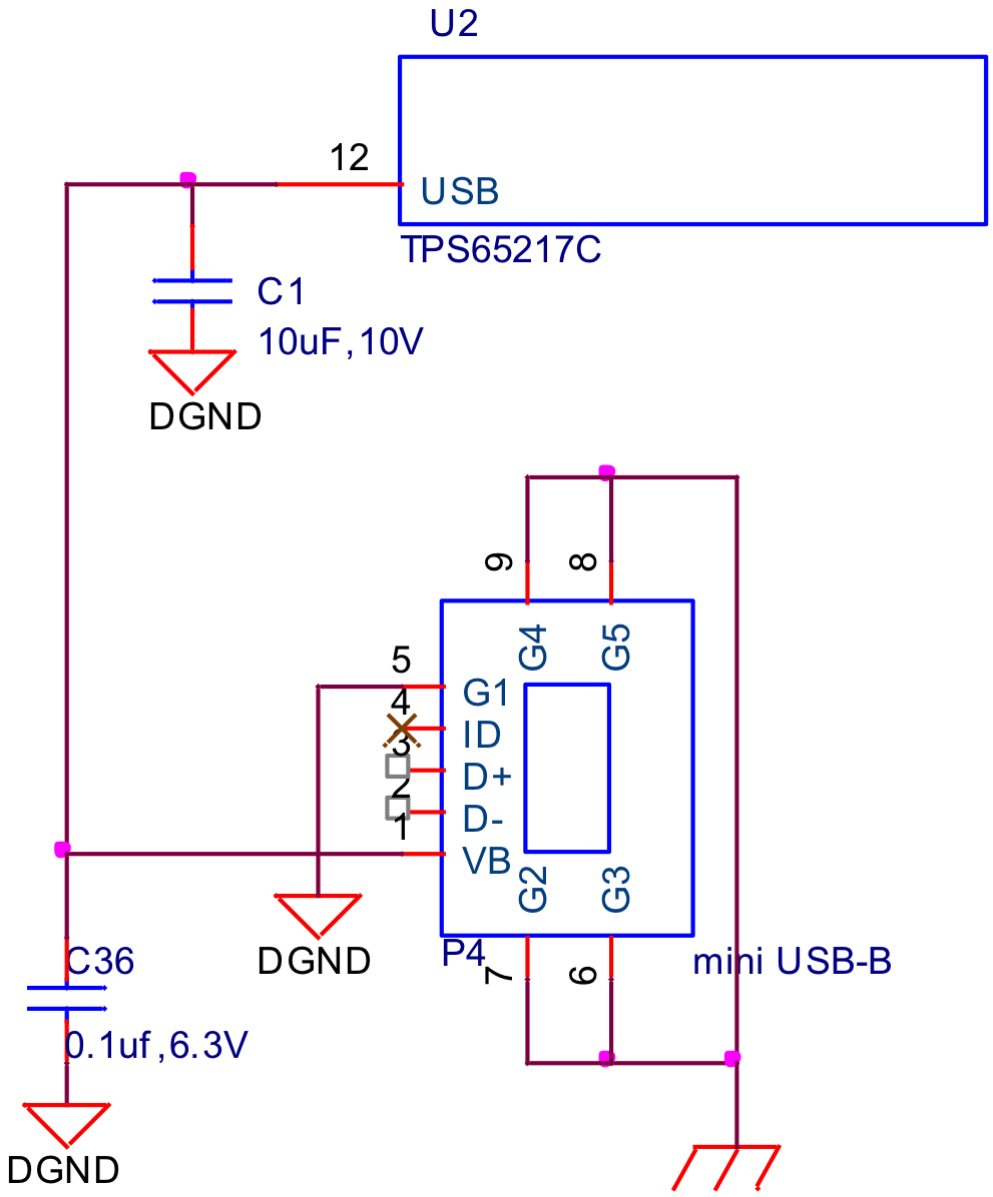


Fig. 5.24: USB Power Connections

Battery Access Pads

Four pads are provided on the board to allow access to the battery pins on the *TPS65217C*. The pads can be loaded with a 4x4 header or you may just wire a battery into the pads. In addition they could provide access via a cape if desired. The four signals are listed below in *table-3* .

Table 5.3: BeagleBone Black Battery Pins

PIN	DESIGNATION	FUNCTION
BAT	TP5	Battery connection point
SENSE	TP6	Battery voltage sense input, connect to BAT directly at the battery terminal.
TS	TP7	Temperature sense input. Connect to NTC thermistor to sense battery temperature.
GND	TP8	System ground.

There is no fuel gauge function provided by the *TPS65217C*. That would need to be added if that function was required. If you want to add a fuel gauge, an option is to use 1-wire SPI or I2C device. You will need to add this using the expansion headers and place it on an expansion board.

NOTE: Refer to the TPS65217C documentation + before connecting anything to these pins.

Power Consumption

The power consumption of the board varies based on power scenarios and the board boot processes. Measurements were taken with the board in the following configuration:

- DC powered and USB powered
- HDMI monitor connected
- USB HUB
- 4GB USB flash drive
- Ethernet connected @ 100M
- Serial debug cable connected

Table 5.4: BeagleBone Black Power Consumption(mA@5V)

MODE	USB	DC	DC+USB
Reset	TBD	TBD	TBD
Idling @ UBoot	210	210	210
Kernel Booting (Peak)	460	460	460
Kernel Idling	350	350	350
Kernel Idling Display Blank	280	280	280
Loading a Webpage	430	430	430

The current will fluctuate as various activates occur, such as the LEDs on and microSD/eMMC accesses.

Processor Interfaces

The processor interacts with the *TPS65217C* via several different signals. Each of these signals is described below.

I2C0

I2C0 is the control interface between the processor and the *TPS65217C*. It allows the processor to control the registers inside the **TPS65217C** for such things as voltage scaling and switching of the input rails.

PMIC_POWR_EN

On power up the *VDD_RTC* rail activates first. After the RTC circuitry in the processor has activated it instructs the**TPS65217C** to initiate a full power up cycle by activating the *PMIC_POWR_EN* signal by taking it HI. When powering down, the processor can take this pin low to start the power down process.

LDO_GOOD

This signal connects to the *RTC_PORZn* signal, RTC power on reset. The small *n indicates that the signal is an active low signal. Word processors seem to be unable to put a bar over a word so the* **n* is commonly used in electronics. As the RTC circuitry comes up first, this signal indicates that the LDOs, the 1.8V VRTC rail, is up and stable. This starts the power up process.

PMIC_PGOOD

Once all the rails are up, the *PMIC_PGOOD* signal goes high. This releases the **PORZn** signal on the processor which was holding the processor reset.

WAKEUP

The WAKEUP signal from the *TPS65217C* is connected to the **EXT_WAKEUP** signal on the processor. This is used to wake up the processor when it is in a sleep mode. When an event is detected by the *TPS65217C*, such as the power button being pressed, it generates this signal.

PMIC_INT

The *PMIC_INT* signal is an interrupt signal to the processor. Pressing the power button will send an interrupt to the processor allowing it to implement a power down mode in an orderly fashion, go into sleep mode, or cause it to wake up from a sleep mode. All of these require SW support.

Power Rails

VRTC Rail

The *VRTC* rail is a 1.8V rail that is the first rail to come up in the power sequencing. It provides power to the RTC domain on the processor and the I/O rail of the **TPS65217C**. It can deliver up to 250mA maximum.

VDD_3V3A Rail

The *VDD_3V3A* rail is supplied by the **TPS65217C** and provides the 3.3V for the processor rails and can provide up to 400mA.

VDD_3V3B Rail

The current supplied by the *VDD_3V3A* rail is not sufficient to power all of the 3.3V rails on the board. So a second LDO is supplied, U4, a **TL5209A**, which sources the *VDD_3V3B* rail. It is powered up just after the *VDD_3V3A* rail.

VDD_1V8 Rail

The *VDD_1V8* rail can deliver up to 400mA and provides the power required for the 1.8V rails on the processor and the HDMI framer. This rail is not accessible for use anywhere else on the board.

VDD_CORE Rail

The *VDD_CORE* rail can deliver up to 1.2A at 1.1V. This rail is not accessible for use anywhere else on the board and connects only to the processor. This rail is fixed at 1.1V and should not be adjusted by SW using the PMIC. If you do, then the processor will no longer work.

VDD_MPU Rail

The *VDD_MPU* rail can deliver up to 1.2A. This rail is not accessible for use anywhere else on the board and connects only to the processor. This rail defaults to 1.1V and can be scaled up to allow for higher frequency operation. Changing of the voltage is set via the I2C interface from the processor.

VDDS_DDR Rail

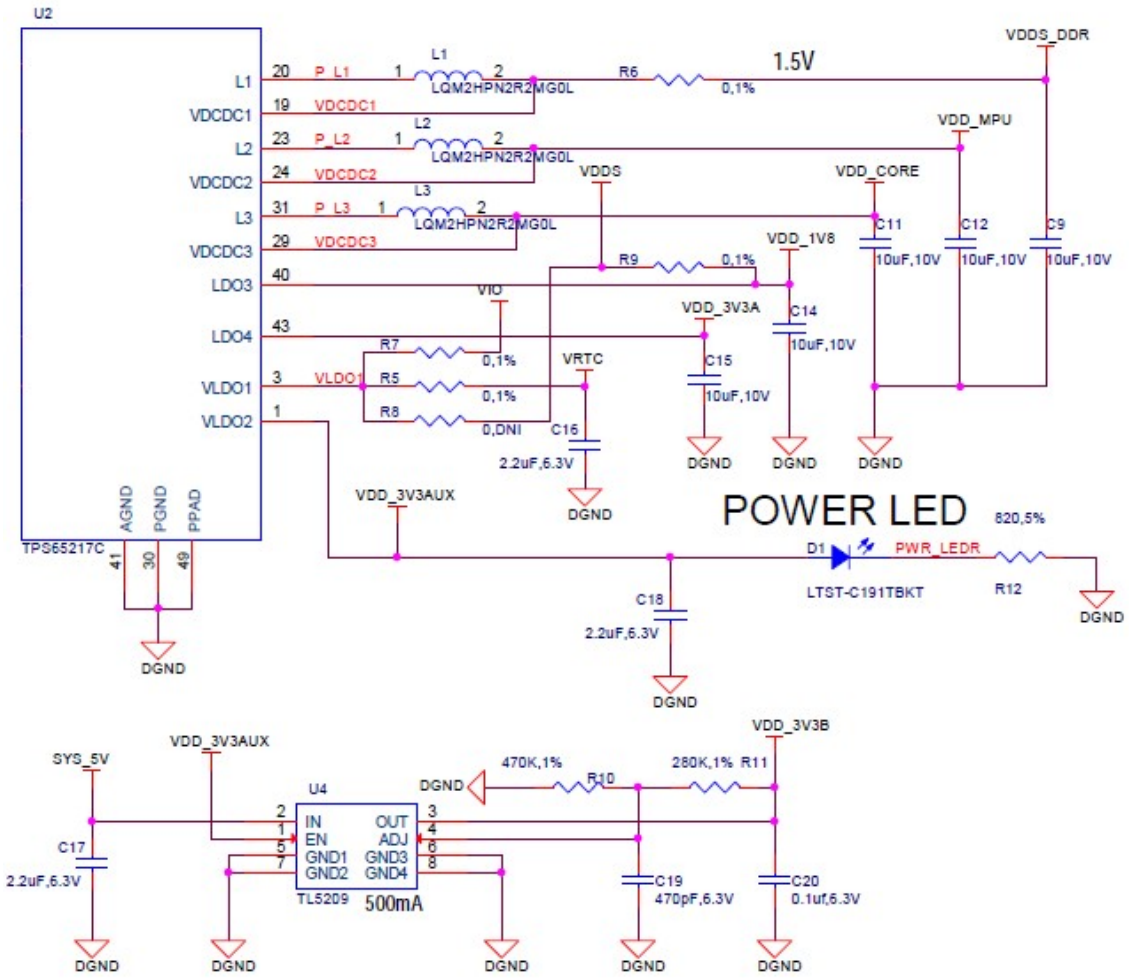


Fig. 5.25: Power Rails

The *VDDS_DDR* rail defaults to**1.5V** to support the DDR3L rails and can deliver up to 1.2A. It is possible to adjust this voltage rail down to *1.35V* for lower power operation of the DDR3L device. Only DDR3L devices can support this voltage setting of 1.35V.

Power Sequencing

The power up process is consists of several stages and events. *figure-26* describes the events that make up the power up process for the processor from the PMIC. This diagram is used elsewhere to convey additional information. I saw no need to bust it up into smaller diagrams. It is from the processor datasheet supplied by Texas Instruments.

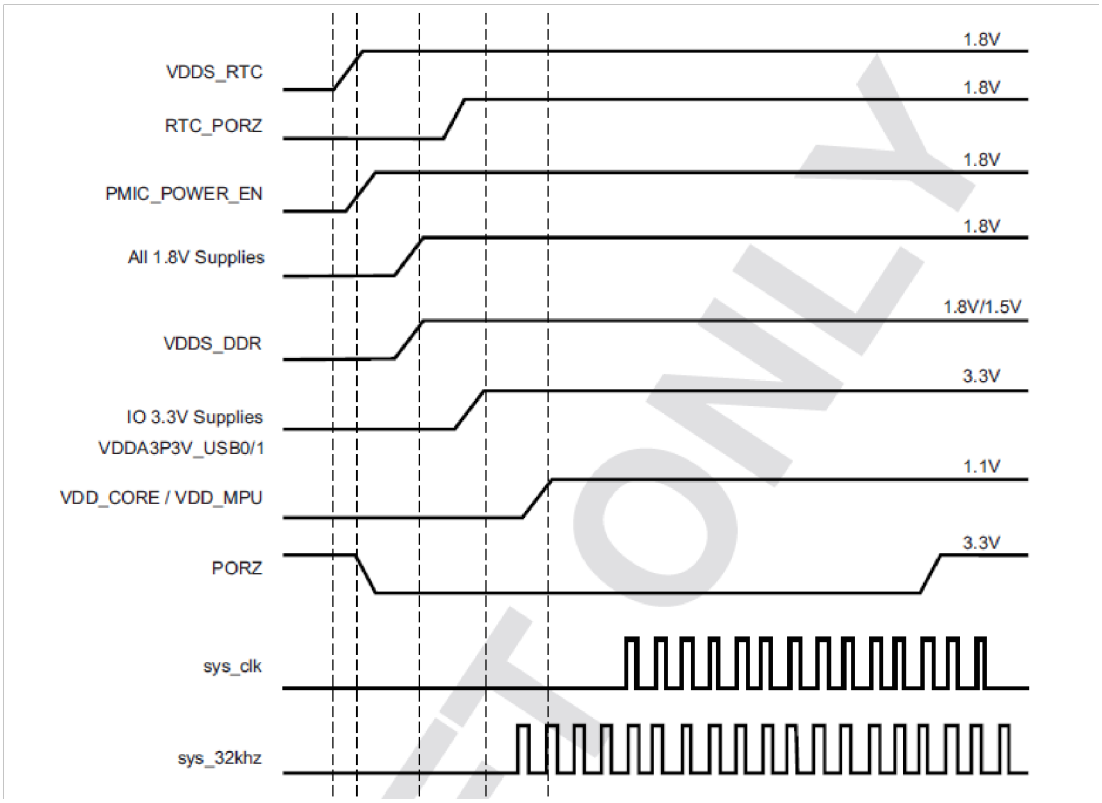


Fig. 5.26: Power Rail Power Up Sequencing

figure-27 the voltage rail sequencing for the **TPS65217C** as it powers up and the voltages on each rail. The power sequencing starts at 15 and then goes to one. That is the way the *TPS65217C* is configured. You can refer to the TPS65217C datasheet for more information.

TPS65217C (Targeted at AM335x - ZCZ)	
VOLTAGE (V)	**SEQUENCE (STROBE)**
1.5	1
1.1	5
1.1	5
1.8	15
3.3	3
1.8 (LDO, 400 mA)	2
3.3 (LDO, 400 mA)	4

Fig. 5.27: TPS65217C Power Sequencing Timing

Power LED

The power LED is a blue LED that will turn on once the *TPS65217C* has finished the power up procedure. If you ever see the LED flash once, that means that the**TPS65217C** started the process and encountered an issue that caused it to shut down. The connection of the LED is shown in *figure-25*.

TPS65217C Power Up Process

Figure below shows the interface between the **TPS65217C** and the processor. It is a cut from the PDF form of the schematic and reflects what is on the schematic.

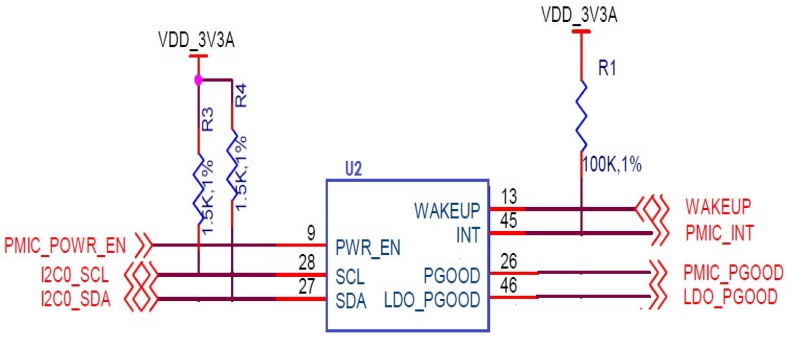


Fig. 5.28: Power Processor Interfaces

When voltage is applied, DC or USB, the *TPS65217C* connects the power to the SYS output pin which drives the switchers and LDOs in the **TPS65217C**.

At power up all switchers and LDOs are off except for the *VRTC LDO* (1.8V), which provides power to the VRTC rail and controls the **RTC_PORZn** input pin to the processor, which starts the power up process of the processor. Once the RTC rail powers up, the *RTC_PORZn* pin, driven by the *LDO_PGOOD* signal from the *TPS65217C*, of the processor is released.

Once the *RTC_PORZn* reset is released, the processor starts the initialization process. After the RTC stabilizes, the processor launches the rest of the power up process by activating the **PMIC_POWER_EN** signal that is connected to the *TPS65217C* which starts the *TPS65217C* power up process.

The *LDO_PGOOD* signal is provided by the**TPS65217C** to the processor. As this signal is 1.8V from the *TPS65217C* by virtue of the *TPS65217C* VIO rail being set to 1.8V, and the *RTC_PORZ* signal on the processor is 3.3V, a voltage level shifter, *U4*, is used. Once the LDOs and switchers are up on the *TPS65217C*, this signal goes active releasing the processor. The LDOs on the *TPS65217C* are used to power the VRTC rail on the processor.

Processor Control Interface

figure-28 above shows two interfaces between the processor and the **TPS65217C** used for control after the power up sequence has completed.

The first is the *I2C0* bus. This allows the processor to turn on and off rails and to set the voltage levels of each regulator to supports such things as voltage scaling.

The second is the interrupt signal. This allows the *TPS65217C* to alert the processor when there is an event, such as when the power button is pressed. The interrupt is an open drain output which makes it easy to interface to 3.3V of the processor.

Low Power Mode Support

This section covers three general power down modes that are available. These modes are only described from a Hardware perspective as it relates to the HW design.

RTC Only

In this mode all rails are turned off except the *VDD_RTC*. The processor will need to turn off all the rails to enter this mode. The **VDD_RTC** staying on will keep the RTC active and provide for the wakeup interfaces to be active to respond to a wake up event.

RTC Plus DDR

In this mode all rails are turned off except the *VDD_RTC* and the **VDDS_DDR**, which powers the DDR3L memory. The processor will need to turn off all the rails to enter this mode. The *VDD_RTC* staying on will keep the RTC active and provide for the wakeup interfaces to be active to respond to a wake up event.

The *VDDS_DDR* rail to the DDR3L is provided by the 1.5V rail of the **TPS65217C** and with *VDDS_DDR* active, the DDR3L can be placed in a self refresh mode by the processor prior to power down which allows the memory data to be saved.

Currently, this feature is not included in the standard software release. The plan is to include it in future releases.

Voltage Scaling

For a mode where the lowest power is possible without going to sleep, this mode allows the voltage on the ARM processor to be lowered along with slowing the processor frequency down. The I2C0 bus is used to control the voltage scaling function in the *TPS65217C*.

5.6.2 Sitara AM3358BZCZ100 Processor

The board is designed to use the Sitara AM3358BZCZ100 processor in the 15 x 15 package. Earlier revisions of the board used the XM3359AZCZ100 processor.

Description

Figure below shows is a high level block diagram of the processor. For more information on the processor, go to http://www.ti.com/product/am3358

High Level Features

Table 5.5: Processor Features

Operating Systems	Linux, Android, Windows Embedded CE,QNX,ThreadX	**MMC/SD**	3
Standby Power	7 mW	**CAN**	2
ARM CPU	1 ARM Cortex-A8	**UART (SCI)**	6
ARM MHz (Max.)	275,500,600,800,1000	**ADC**	8-ch 12-bit
ARM MIPS (Max.)	1000,1200,2000	**PWM (Ch)**	3
Graphics Acceleration	1 3D	**eCAP**	3
Other Hardware Acceleration	2 PRU-ICSS,Crypto Accelerator	**eQEP**	3
On-Chip L1 Cache	64 KB (ARM Cortex-A8)	**RTC**	1
On-Chip L2 Cache	256 KB (ARM Cortex- A8)	**I2C**	3
Other On-Chip Memory	128 KB	**McASP**	2

continues on next page

Table 5.5 – continued from previous page

Operating Systems	Linux, Android, Windows Embedded CE,QNX,ThreadX	**MMC/SD**	3
Display Options	LCD	**SPI**	2
General Purpose Memory	1 16-bit (GPMC, NAND flash, NOR Flash, SRAM)	**DMA (Ch)**	64-Ch EDMA
DRAM	1 16-bit (LPDDR-400,DDR2-532, DDR3-400)	**IO Supply (V)**	1.8V(ADC), 3.3V
USB Ports	2	**Operating Temperature Range (C)**	40 to 90

Documentation

Full documentation for the processor can be found on the TI website at http://www.ti.com/product/am3358 for the current processor used on the board. Make sure that you always use the latest datasheets and Technical Reference Manuals (TRM).

Crystal Circuitry

Reset Circuitry

figure-31 is the board reset circuitry. The initial power on reset is generated by the **TPS65217C** power management IC. It also handles the reset for the Real Time Clock.

The board reset is the SYS_RESETn signal. This is connected to the NRESET_INOUT pin of the processor. This pin can act as an input or an output. When the reset button is pressed, it sends a warm reset to the processor and to the system.

On the revision A5D board, a change was made. On power up, the NRESET_INOUT signal can act as an output. In this instance it can cause the SYS_RESETn line to go high prematurely. In order to prevent this, the PORZn signal from the TPS65217C is connected to the SYS_RESETn line using an open drain buffer. These ensure that the line does not momentarily go high on power up.

This change is also in all revisions after A5D.

DDR3L Memory

The BeagleBone Black uses a single MT41K256M16HA-125 512MB DDR3L device from Micron that interfaces to the processor over 16 data lines, 16 address lines, and 14 control lines. On rev C we added the Kingston *KE4CN2H5A-A58* device as a source for the DDR3L device**.**

The following sections provide more details on the design.

Memory Device

The design supports the standard DDR3 and DDR3L x16 devices and is built using the DDR3L. A single x16 device is used on the board and there is no support for two x8 devices. The DDR3 devices work at 1.5V and the DDR3L devices can work down to

1.35V to achieve lower power. The DDR3L comes in a 96-BALL FBGA package with 0.8 mil pitch. Other standard DDR3 devices can also be supported, but the DDR3L is the lower power device and was chosen for its ability to work at 1.5V or 1.35V. The standard frequency that the DDR3L is run at on the board is 400MHZ.

DDR3L Memory Design

figure-32 is the schematic for the DDR3L memory device. Each of the groups of signals is described in the following lines.

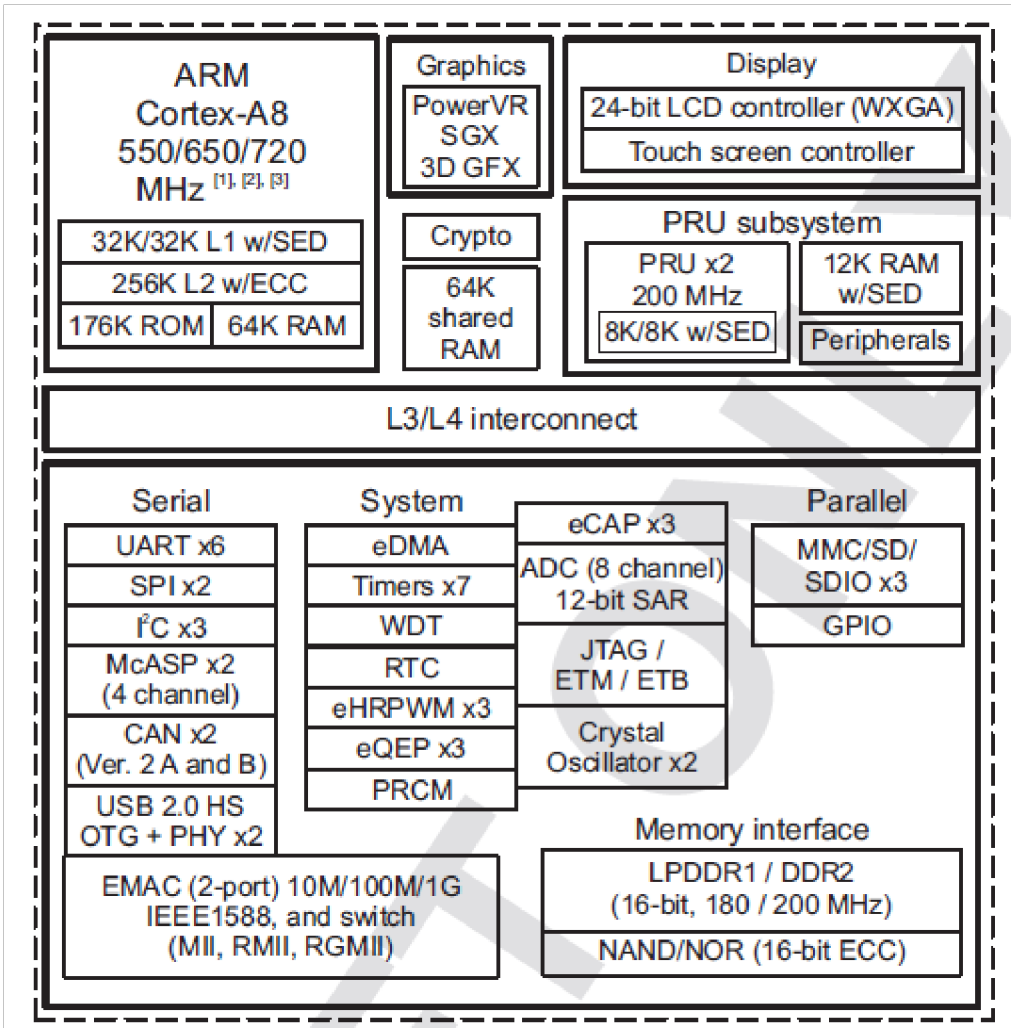


Fig. 5.29: Sitara AM3358BZCZ Block Diagram

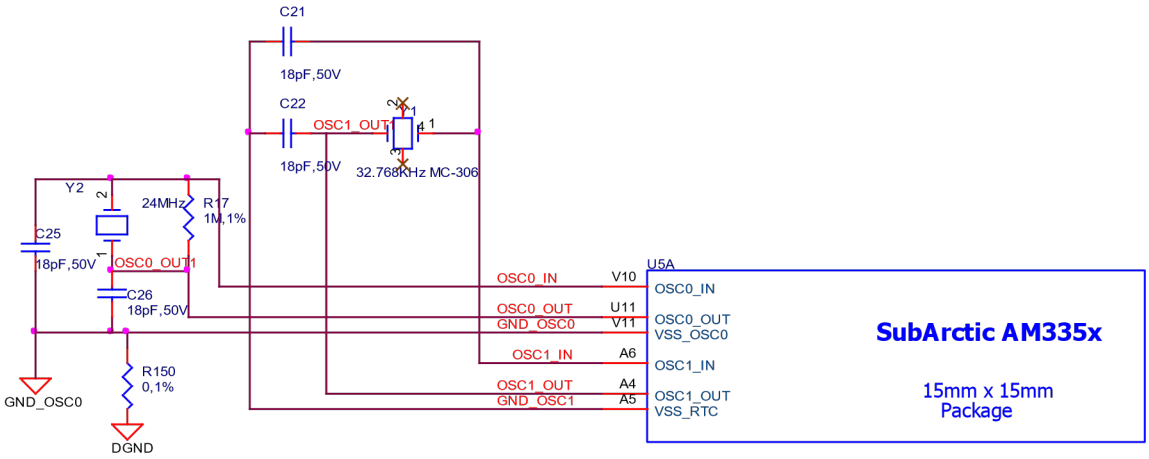


Fig. 5.30: Processor Crystals

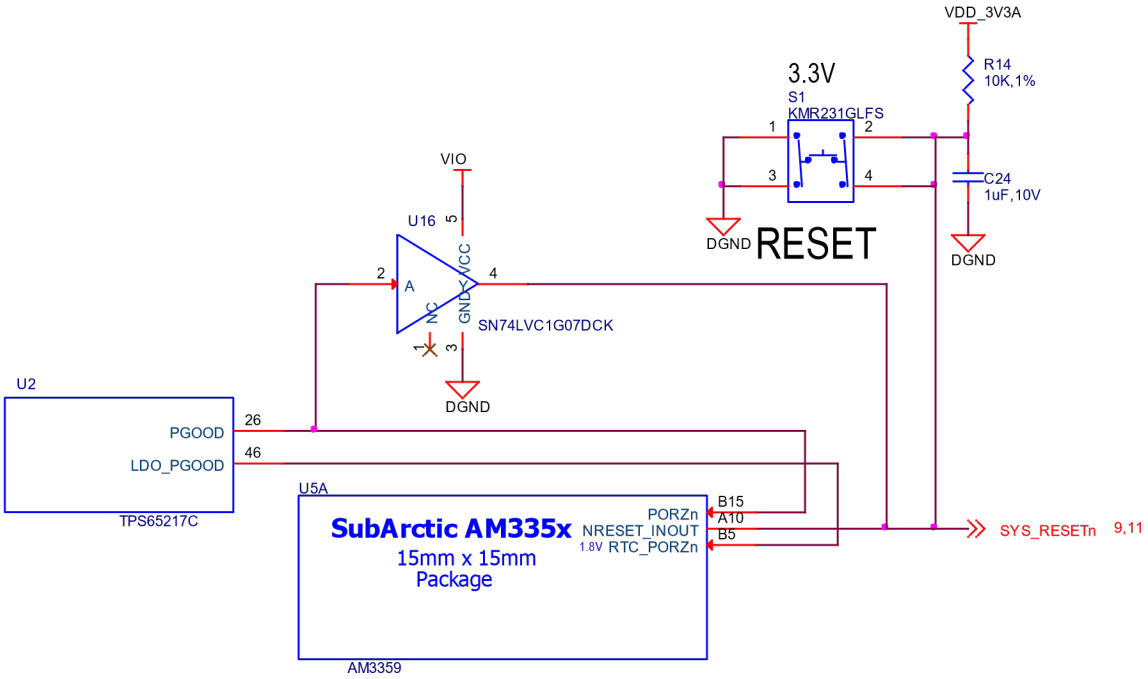


Fig. 5.31: Board Reset Circuitry

Address Lines: Provide the row address for ACTIVATE commands, and the column address and auto precharge bit (A10) for READ/WRITE commands, to select one location out of the memory array in the respective bank. A10 sampled during a PRECHARGE command determines whether the PRECHARGE applies to one bank (A10 LOW, bank selected by BA[2:0]) or all banks (A10 HIGH). The address inputs also provide the op-code during a LOAD MODE command. Address inputs are referenced to VREFCA. A12/BC#: When enabled in the mode register (MR), A12 is sampled during READ and WRITE commands to determine whether burst chop (on-the-fly) will be performed (HIGH = BL8 or no burst chop, LOW = BC4 burst chop).

Bank Address Lines: BA[2:0] define the bank to which an ACTIVATE, READ, WRITE, or PRECHARGE command is being applied. BA[2:0] define which mode register (MR0, MR1, MR2, or MR3) is loaded during the LOAD MODE command. BA[2:0] are referenced to VREFCA.

CK and CK# Lines: are differential clock inputs. All address and control input signals are sampled on the crossing of the positive edge of CK and the negative edge of CK#. Output data strobe (DQS, DQS#) is referenced to the crossings of CK and CK#.

Clock Enable Line: CKE enables (registered HIGH) and disables (registered LOW) internal circuitry and clocks on the DRAM. The specific circuitry that is enabled/disabled is dependent upon the DDR3 SDRAM configuration and operating mode. Taking CKE LOW provides PRECHARGE power-down and SELF REFRESH operations (all banks idle) or active power-down (row active in any bank). CKE is synchronous for powerdown entry and exit and for self refresh entry. CKE is asynchronous for self refresh exit. Input buffers (excluding CK, CK#, CKE, RESET#, and ODT) are disabled during powerdown. Input buffers (excluding CKE and RESET#) are disabled during SELF REFRESH. CKE is referenced to VREFCA.

Chip Select Line: CS# enables (registered LOW) and disables (registered HIGH) the command decoder. All commands are masked when CS# is registered HIGH. CS# provides for external rank selection on systems with multiple ranks. CS# is considered part of the command code. CS# is referenced to VREFCA.

Input Data Mask Line: DM is an input mask signal for write data. Input data is masked when DM is sampled HIGH along with the input data during a write access. Although the DM ball is input-only, the DM loading is designed to match that of the DQ and DQS balls. DM is referenced to VREFDQ.

On-die Termination Line: ODT enables (registered HIGH) and disables (registered LOW) termination resistance internal to the DDR3L SDRAM. When enabled in normal operation, ODT is only applied to each of the following balls: DQ[7:0], DQS, DQS#, and DM for the x8; DQ[3:0], DQS, DQS#, and DM for the x4. The ODT input is ignored if disabled via the LOAD MODE command. ODT is referenced to VREFCA.

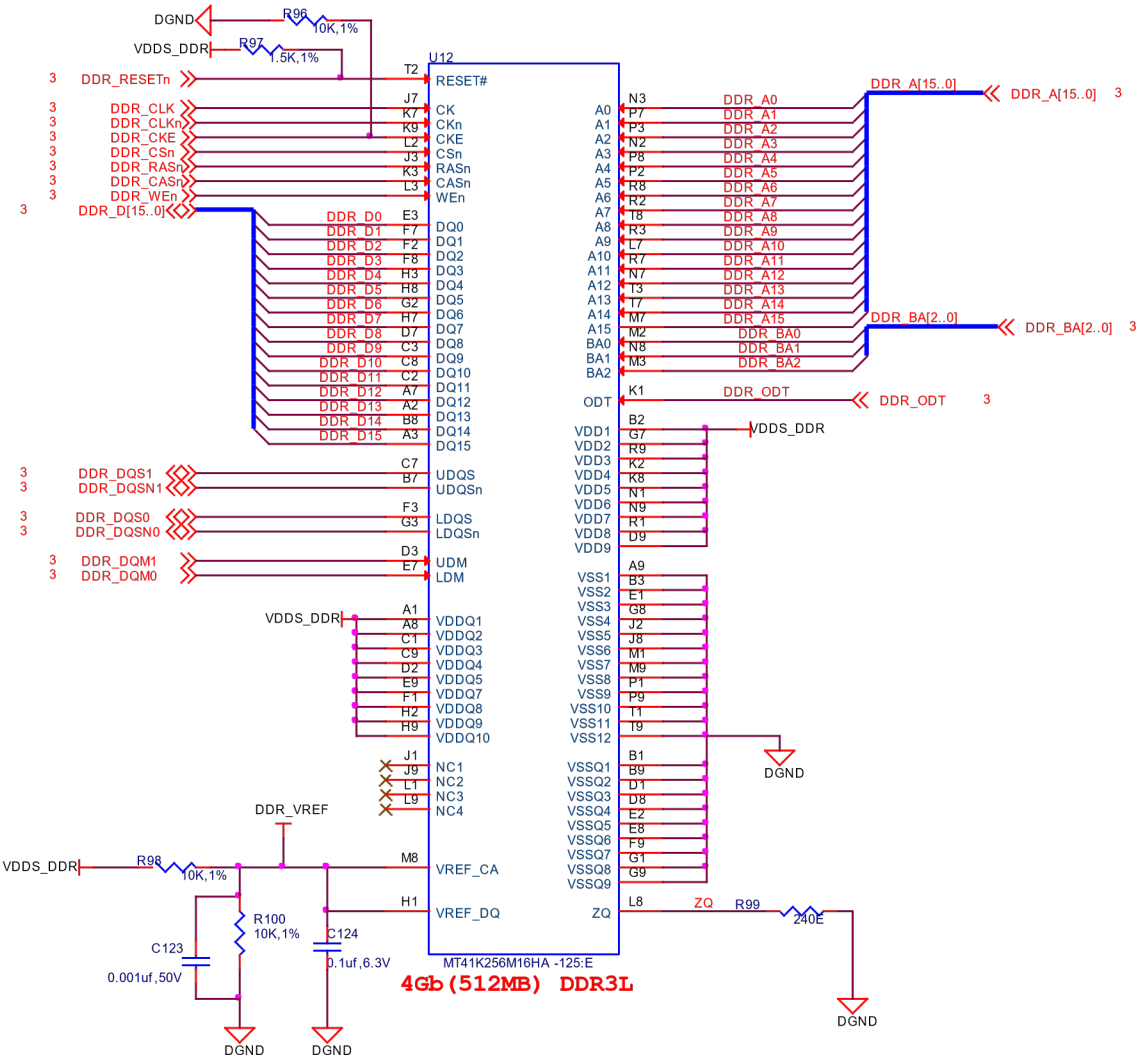


Fig. 5.32: DDR3L Memory Design

Power Rails

The *DDR3L* memory device and the DDR3 rails on the processor are supplied by the**TPS65217C**. Default voltage is 1.5V but can be scaled down to 1.35V if desired.

VREF

The *VREF* signal is generated from a voltage divider on the**VDDS_DDR** rail that powers the processor DDR rail and the DDR3L device itself. *Figure 33* below shows the configuration of this signal and the connection to the DDR3L memory device and the processor.

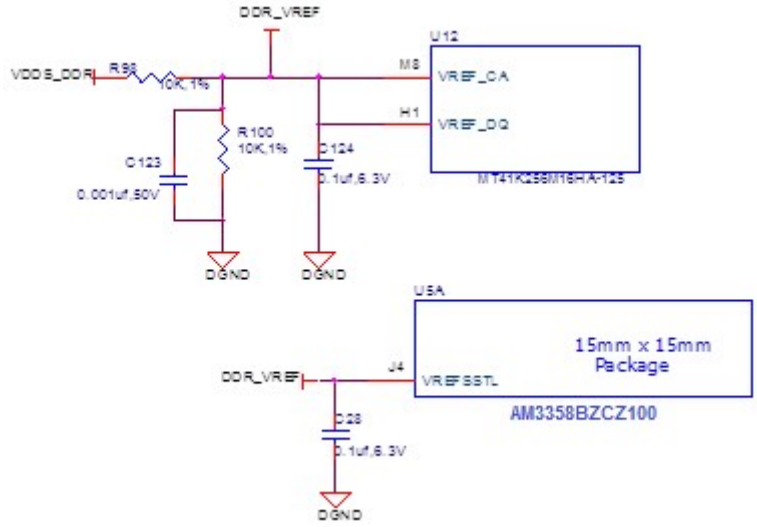


Fig. 5.33: DDR3L VREF Design

5.6.3 4GB eMMC Memory

The eMMC is a communication and mass data storage device that includes a Multi-MediaCard (MMC) interface, a NAND Flash component, and a controller on an advanced 11-signal bus, which is compliant with the MMC system specification. The nonvolatile eMMC draws no power to maintain stored data, delivers high performance across a wide range of operating temperatures, and resists shock and vibration disruption.

One of the issues faced with SD cards is that across the different brands and even within the same brand, performance can vary. Cards use different controllers and different memories, all of which can have bad locations that the controller handles. But the controllers may be optimized for reads or writes. You never know what you will be getting. This can lead to varying rates of performance. The eMMC card is a known controller and when coupled with the 8bit mode, 8 bits of data instead of 4, you get double the performance which should result in quicker boot times.

The following sections describe the design and device that is used on the board to implement this interface.

eMMC Device

The device used is one of two different devices:

- Micron *MTFC4GLDEA 0M WT*
- Kingston *KE4CN2H5A-A58*

The package is a 153 ball WFBGA device on both devices.

eMMC Circuit Design

figure-34 is the design of the eMMC circuitry. The eMMC device is connected to the MMC1 port on the processor. MMC0 is still used for the microSD card as is currently done on the original BeagleBone. The size of the eMMC supplied is now 4GB.

The device runs at 3.3V both internally and the external I/O rails. The VCCI is an internal voltage rail to the device. The manufacturer recommends that a 1uF capacitor be attached to this rail, but a 2.2uF was chosen to provide a little margin.

Pullup resistors are used to increase the rise time on the signals to compensate for any capacitance on the board.

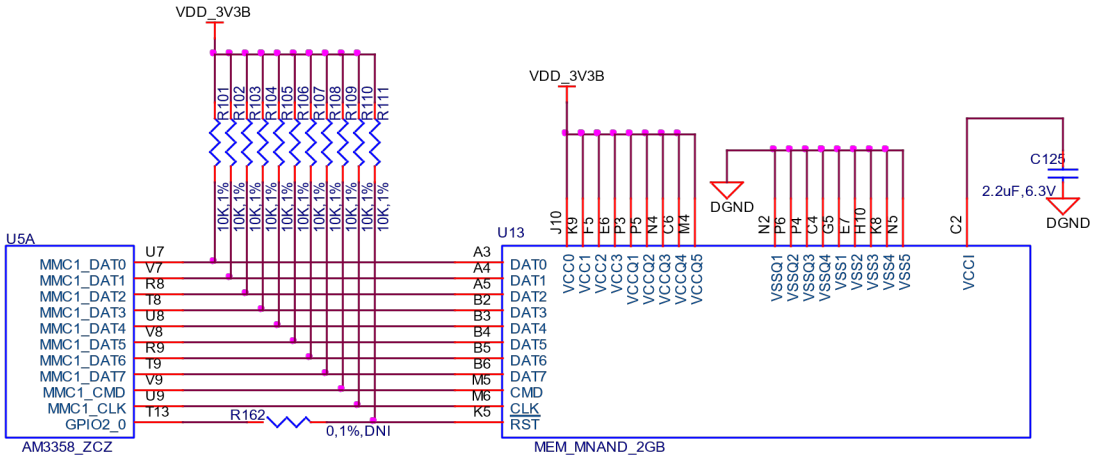


Fig. 5.34: eMMC Memory Design

The pins used by the eMMC1 in the boot mode are listed below in *Table 6*.

Signal name	Pin Used in Device
clk	gpmc_csn1
cmd	gpmc_csn2
dat0	gpmc_ad0
dat1	gpmc_ad1
dat2	gpmc_ad2
dat3	gpmc_ad3

Fig. 5.35: eMMC Boot Pins

For eMMC devices the ROM will only support raw mode. The ROM Code reads out raw sectors from image or the booting file within the file system and boots from it. In raw mode the booting image can be located at one of the four consecutive locations in the main area: offset 0x0 / 0x20000 (128 KB) / 0x40000 (256 KB) / 0x60000 (384 KB). For this reason, a booting image shall not exceed 128KB in size. However it is possible to flash a device with an image greater than 128KB starting at one of the aforementioned locations. Therefore the ROM Code does not check the image size. The only drawback is that the image will cross the subsequent image boundary. The raw mode is detected by reading sectors #0, #256, #512, #768. The content of these sectors is then verified for presence of a TOC structure. In the case of a *GP Device*, a Configuration Header (CH)*must* be located in the first sector followed by a *GP header*. The CH might be void (only containing a CHSETTINGS item for which the Valid field is zero).

The ROM only supports the 4-bit mode. After the initial boot, the switch can be made to 8-bit mode for increasing the overall performance of the eMMC interface.

5.6.4 Board ID EEPROM

The BeagleBone is equipped with a single 32Kbit(4KB) 24LC32AT-I/OT EEPROM to allow the SW to identify the board. *Table 7* below defined the contents of the EEPROM.

Table 5.6: EEPROM Contents

Name	Size (bytes)	Contents
Header	4	0xAA, 0x55, 0x33, EE
Board Name	8	Name for board in ASCII: **A335BNLT**
Version	4	Hardware version code for board in ASCII: **00A3 for Rev A3, 00A4 for Rev A4, 00A5 for Rev A5, 00A6 for Rev A6,00B0 for Rev B, and 00C0 for Rev C.**
Serial Number	12	Serial number of the board. This is a 12 character string which is: **WWYY4P16nnnn** where, WW = 2 digit week of the year of production YY = 2 digit year of production BBBK = BeagleBone Black nnnn = incrementing board number
Configuration Option	32	Codes to show the configuration setup on this board. **All FF**
RSVD	6	FF FF FF FF FF FF
RSVD	6	FF FF FF FF FF FF
RSVD	6	FF FF FF FF FF FF
Available	4018	Available space for other non-volatile codes/data

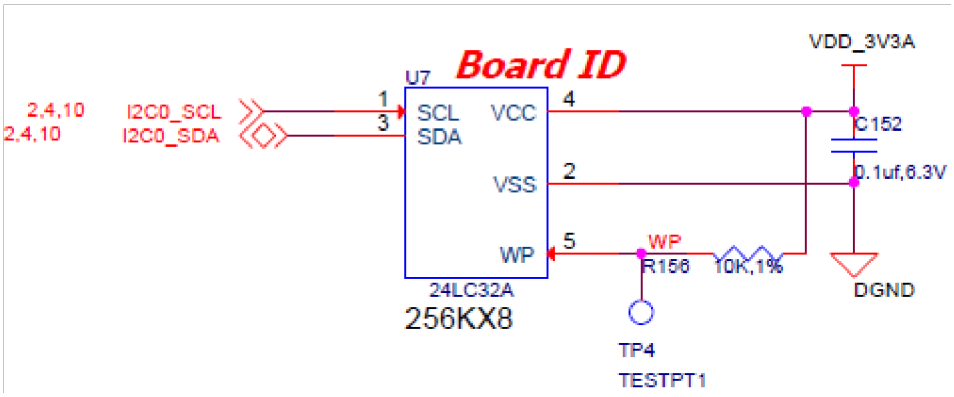


Fig. 5.36: EEPROM Design Rev A5

The EEPROM is accessed by the processor using the I2C 0 bus. The *WP* pin is enabled by default. By grounding the test point, the write protection is removed.

The first 48 locations should not be written to if you choose to use the extras storage space in the EEPROM for other purposes. If you do, it could prevent the board from booting properly as the SW uses this information to determine how to set up the board.

5.6.5 Micro Secure Digital

The microSD connector on the board will support a microSD card that can be used for booting or file storage on the BeagleBone Black.

microSD Design

The signals *MMC0-3* are the data lines for the transfer of data between the processor and the microSD connector.

The *MMC0_CLK* signal clocks the data in and out of the microSD card.

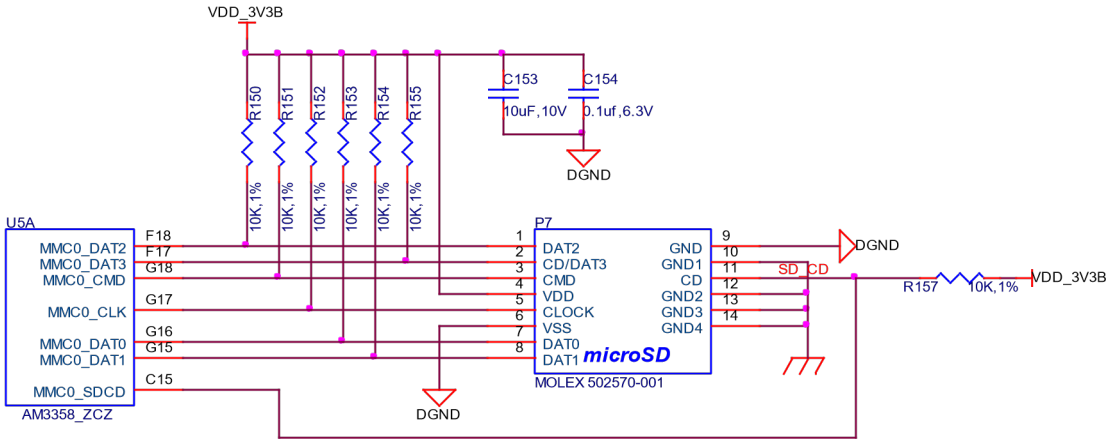


Fig. 5.37: microSD Design

The *MMCO_CMD* signal indicates that a command versus data is being sent.

There is no separate card detect pin in the microSD specification. It uses *MMCO_DAT3* for that function. However, most microSD connectors still supply a CD function on the connectors. In the BeagleBone Black design, this pin is connected to the **MMC0_SDCD** pin for use by the processor. You can also change the pin to *GPIO0_6*, which is able to wake up the processor from a sleep mode when an microSD card is inserted into the connector.

Pullup resistors are provided on the signals to increase the rise times of the signals to overcome PCB capacitance.

Power is provided from the *VDD_3V3B* rail and a 10uF capacitor is provided for filtering.

5.6.6 6.6 User LEDs

There are four user LEDs on the BeagleBone Black. These are connected to GPIO pins on the processor. *Figure 37* shows the interfaces for the user LEDs.

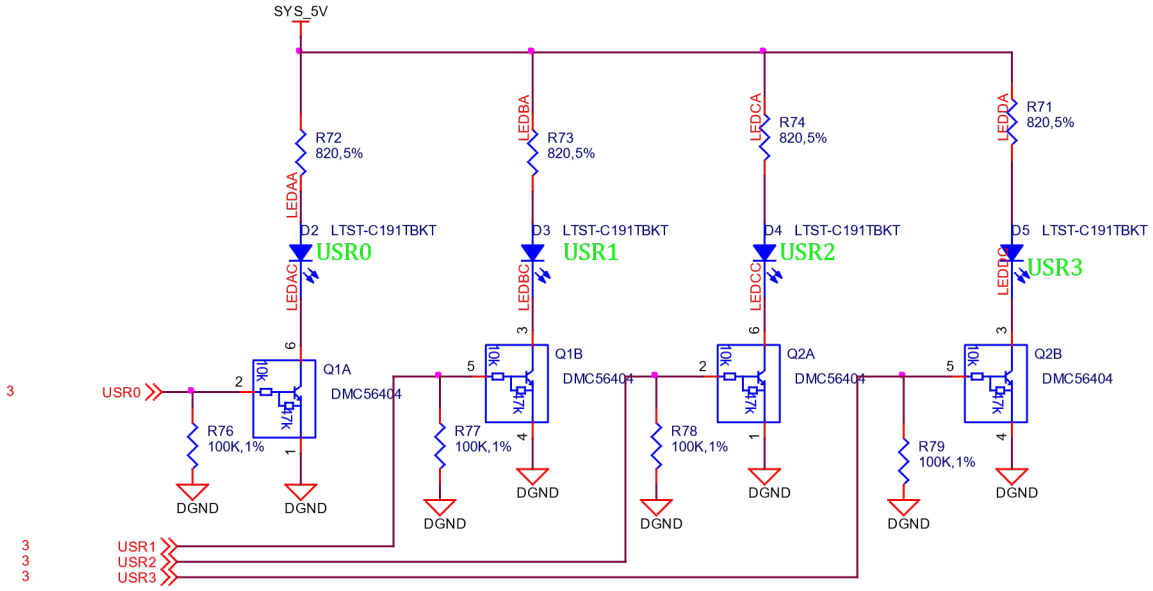


Fig. 5.38: User LEDs

Resistors R71-R74 were changed to 4.75K on the revision A5B and later boards.

Table 5.7: User LED Control Signals/Pins

LED	GPIO SIGNAL	PROC PIN
USR0	GPIO1_21	V15
USR1	GPIO1_22	U15
USR2	GPIO1_23	T15
USR3	GPIO1_24	V16

A logic level of “1” will cause the LEDs to turn on.

5.6.7 Boot Configuration

The design supports two groups of boot options on the board. The user can switch between these modes via the Boot button. The primary boot source is the onboard eMMC device. By holding the Boot button, the user can force the board to boot from the microSD slot. This enables the eMMC to be overwritten when needed or to just boot an alternate image. The following sections describe how the boot configuration works.

In most applications, including those that use the provided demo distributions available from beagleboard.org the processor-external boot code is composed of two stages. After the primary boot code in the processor ROM passes control, a secondary stage (secondary program loader – “SPL” or “MLO”) takes over. The SPL stage initializes only the required devices to continue the boot process, and then control is transferred to the third stage “U-boot”. Based on the settings of the boot pins, the ROM knows where to go and get the SPL and UBoot code. In the case of the BeagleBone Black, that is either eMMC or microSD based on the position of the boot switch.

Boot Configuration Design

figure-38 shows the circuitry that is involved in the boot configuration process. On power up, these pins are read by the processor to determine the boot order. S2 is used to change the level of one bit from HI to LO which changes the boot order.

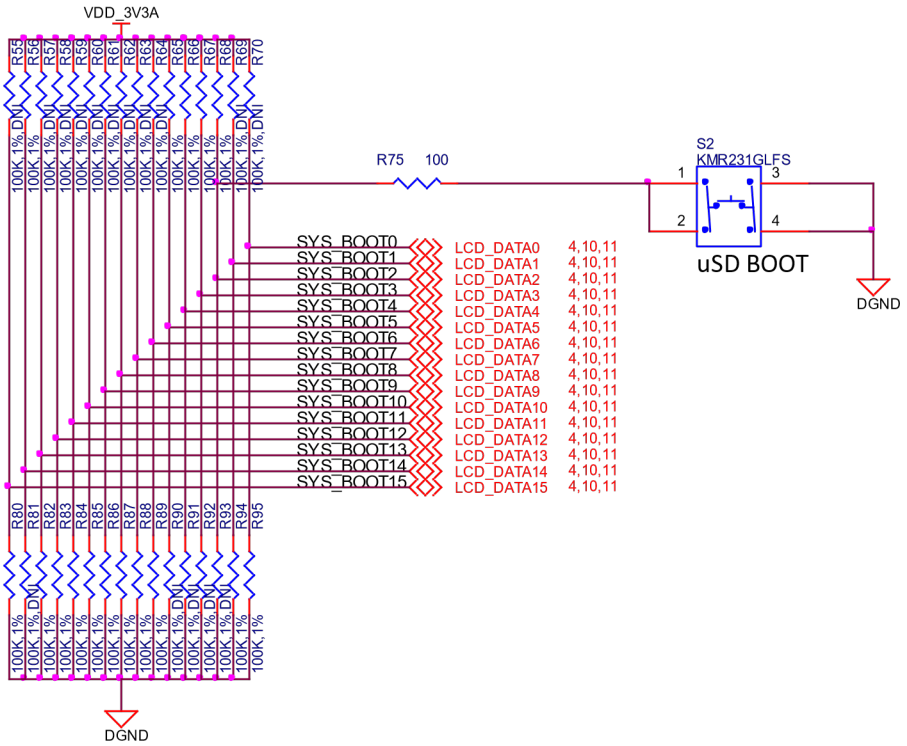


Fig. 5.39: Processor Boot Configuration Design

It is possible to override these setting via the expansion headers. But be careful not to add too much load such that it could interfere with the operation of the HDMI interface or LCD panels. If you choose to override these settings, it is strongly recommended that you gate these signals with the *SYS_RESETn* signal. This ensures that after coming out of reset these signals are removed from the expansion pins.

5.6.8 Default Boot Options

Based on the selected option found in *figure-39* below, each of the boot sequences for each of the two settings is shown.

SYSBOOT[15:14]	SYSBOOT[13:12]	SYSBOOT[11:10]	SYSBOOT[9]	SYSBOOT[8]	SYSBOOT[7:6]	SYSBOOT[5]	SYSBOOT[4:0]	Boot Sequence			
00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz	00b (all other values reserved)	Don't care for ROM code	Don't care for ROM code	Don't care for ROM code	Don't care for ROM code	0 = CLKOUT1 disabled 1 = CLKOUT1 enabled	11100b	MMC1	MMC0	UART0	USB0[5]
00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz	00b (all other values reserved)	Don't care for ROM code	Don't care for ROM code	Don't care for ROM code	Don't care for ROM code	0 = CLKOUT1 disabled 1 = CLKOUT1 enabled	11000b	SPI0	MMC0	USB0[5]	UART0

Fig. 5.40: Processor Boot Configuration

The first row in «figure-39» is the default setting. On boot, the processor will look for the eMMC on the MMC1 port first, followed by the microSD slot on MMC0, USB0 and UART0. In the event there is no microSD card and the eMMC is empty, UART0 or USB0 could be used as the board source.

If you have a microSD card from which you need to boot from, hold the boot button down. On boot, the processor will look for the SPIO0 port first, then microSD on the MMC0 port, followed by USB0 and UART0. In the event there is no microSD card and the eMMC is empty, USB0 or UART0 could be used as the board source.

5.6.9 10/100 Ethernet

The BeagleBone Black is equipped with a 10/100 Ethernet interface. It uses the same PHY as is used on the original BeagleBone. The design is described in the following sections.

6.9.1 Ethernet Processor Interface

This is the same interface as is used on the BeagleBone. No changes were made in this design for the board.

Ethernet Connector Interface

The off board side of the PHY connections are shown in *Figure 41* below.

This is the same interface as is used on the BeagleBone. No changes were made in this design for the board.

Ethernet PHY Power, Reset, and Clocks

VDD_3V3B Rail

The VDD_3V3B rail is the main power rail for the *LAN8710A*. It originates at the VD_3V3B regulator and is the primary rail that supports all of the peripherals on the board. This rail also supplies the VDDIO rails which set the voltage levels for all of the I/O signals between the processor and the**LAN8710A**.

VDD_PHYA Rail

A filtered version of VDD_3V3B rail is connected to the VDD rails of the LAN8710 and the termination resistors on the Ethernet signals. It is labeled as *VDD_PHYA*. The filtering inductor helps block transients that may be seen on the VDD_3V3B rail.

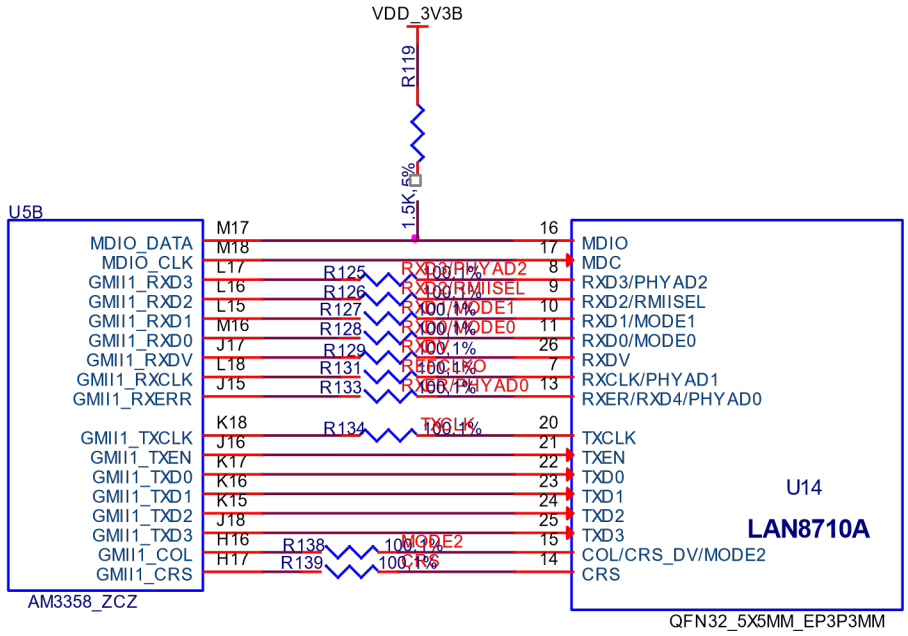


Fig. 5.41: Ethernet Processor Interface

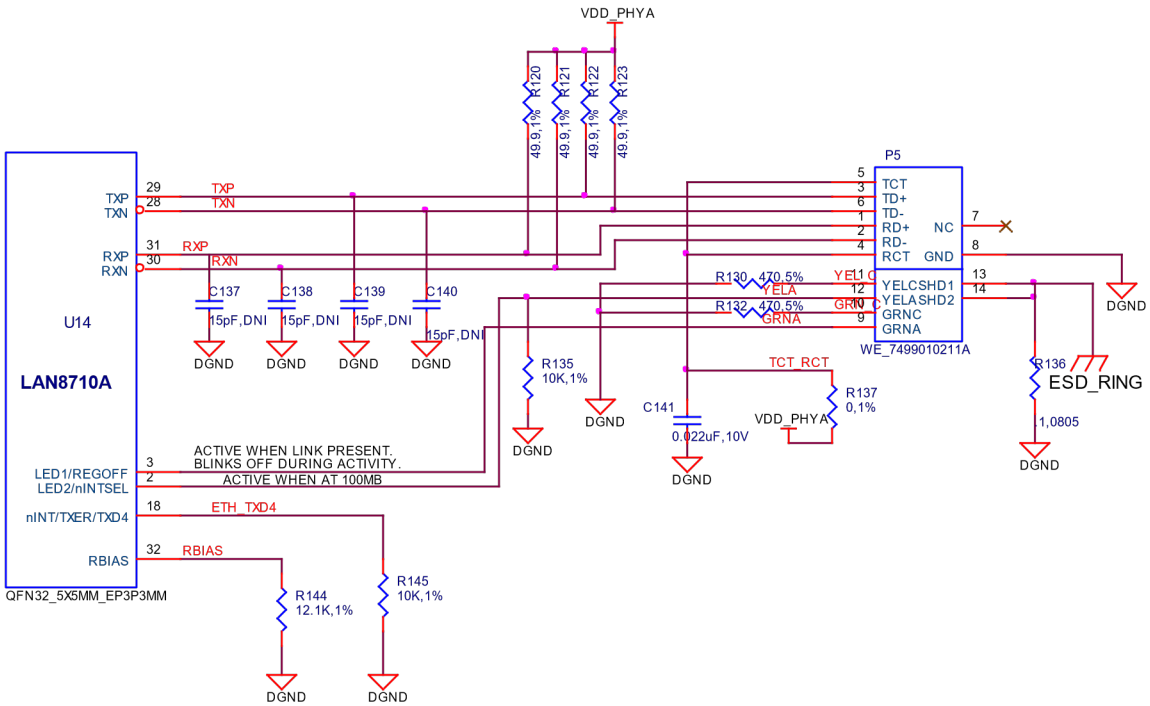


Fig. 5.42: Ethernet Connector Interface

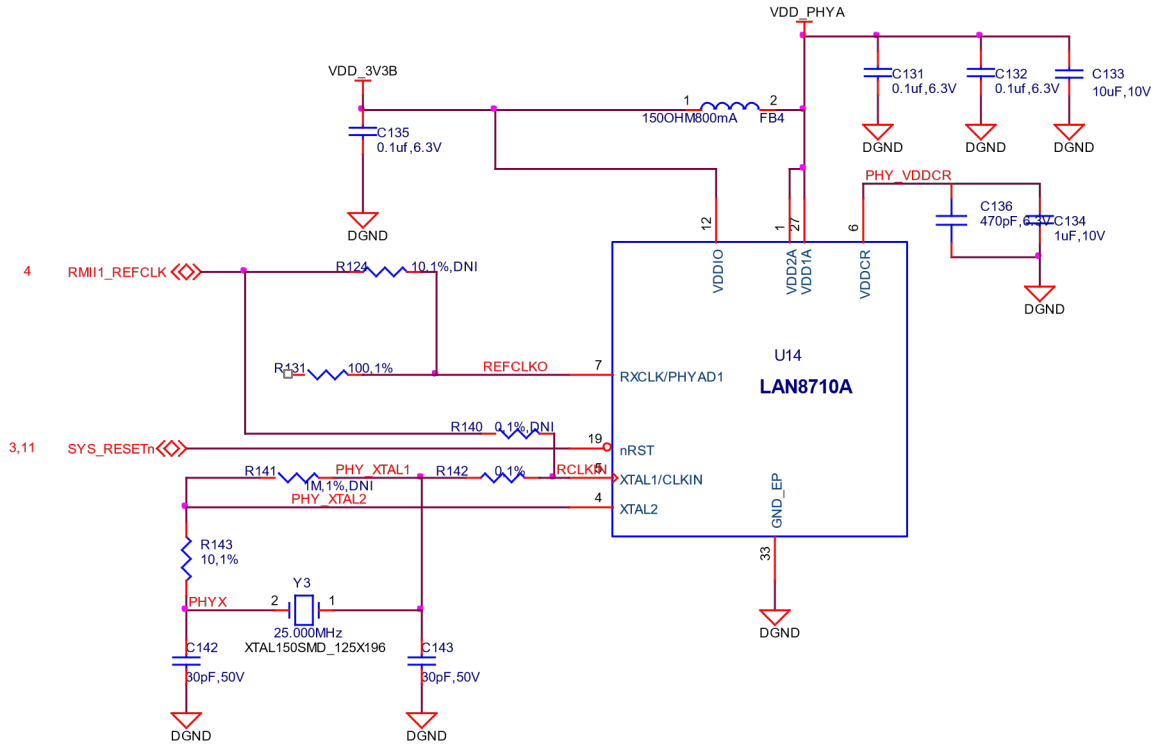


Fig. 5.43: Ethernet PHY, Power, Reset, and Clocks

PHY_VDDCR Rail

The *PHY_VDDCR* rail originates inside the LAN8710A. Filter and bypass capacitors are used to filter the rail. Only circuitry inside the LAN8710A uses this rail.

SYS_RESET

The reset of the LAN8710A is controlled via the *SYS_RESETn* signal, the main board reset line.

Clock Signals

A crystal is used to create the clock for the LAN8710A. The processor uses the *RMII_RXCLK* signal to provide the clocking for the data between the processor and the LAN8710A.

5.6.10 LAN8710A Mode Pins

There are mode pins on the LAN8710A that sets the operational mode for the PHY when coming out of reset. These signals are also used to communicate between the processor and the LAN8710A. As a result, these signals can be driven by the processor which can cause the PHY not to be initialized correctly. To ensure that this does not happen, three low value pull up resistors are used. *Figure 43* below shows the three mode pin resistors.

This will set the mode to be 111, which enables all modes and enables auto-negotiation.

5.6.11 HDMI Interface

The BeagleBone Black has an onboard HDMI framer that converts the LCD signals and audio signals to drive a HDMI monitor. The design uses an NXP *TDA19988* HDMI Framer.

The following sections provide more detail into the design of this interface.

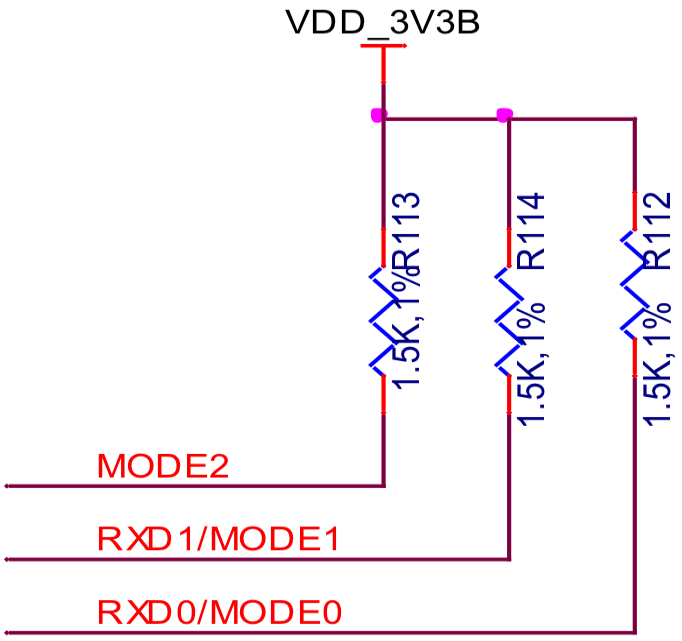


Fig. 5.44: Ethernet PHY Mode Pins

Supported Resolutions

The maximum resolution supported by the BeagleBone Black is 1280x1024 @ 60Hz. *Table 9* below shows the supported resolutions. Not all resolutions may work on all monitors, but these have been tested and shown to work on at least one monitor. EDID is supported on the BeagleBone Black. Based on the EDID reading from the connected monitor, the highest compatible resolution is selected.

Table 5.8: HDMI Supported Monitor Resolutions

RESOLUTION	AUDIO
800 x 600 @60Hz	
800 x 600 @56Hz	
640 x 480 @75Hz	
640 x 480 @60Hz	YES
720 x 400 @70Hz	
1280 x 1024 @75Hz	
1024 x 768 @75Hz	
1024 x 768 @70Hz	
1024 x 768 @60Hz	
800 x 600 @75Hz	
800 x 600 @72Hz	
720 x 480 @60Hz	YES
1280 x 720 @60Hz	YES
1920 x 1080 @24Hz	YES

NOTE: The updated software image used on the Rev A5B and later boards added support for 1920x1080@24HZ.

Audio is limited to CEA supported resolutions. LCD panels only activate the audio in CEA modes. This is a function of the specification and is not something that can be fixed on the board via a hardware change or a software change.

HDMI Framer

The *TDA19988* is a High-Definition Multimedia Interface (HDMI) 1.4a transmitter. It is backward compatible with DVI 1.0 and can be connected to any DVI 1.0 or HDMI sink. The HDCP mode is not used in the design. The non-HDCP version of the device is used in the BeagleBone Black design.

This device provides additional embedded features like CEC (Consumer Electronic Control). CEC is a single bidirectional bus that transmits CEC over the home appliance network connected through this bus. This elimi-

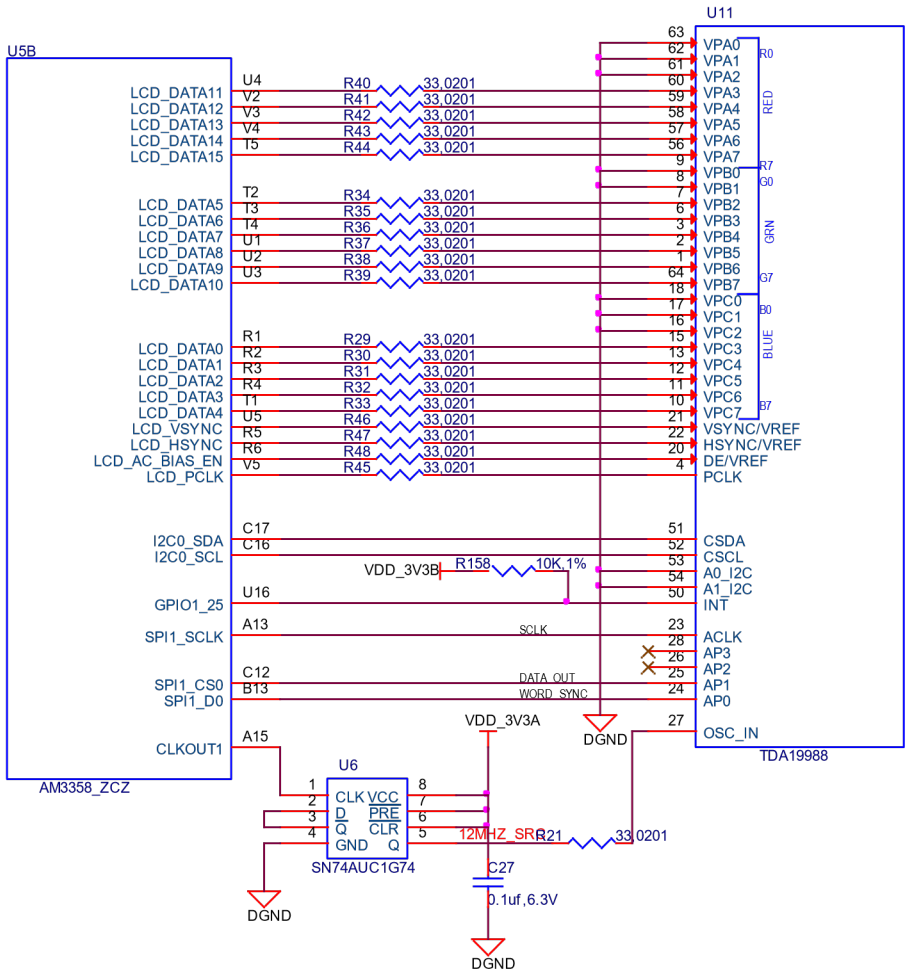


Fig. 5.45: HDMI Framer Processor Interface

the address of the *TDA19988*. In this design they are both tied low. The I2C interface supports both 400kHz and 100KhZ operation. *Table 10* shows the I2C address.

HDMI core address							
A6	A5	A4	A3	A2	A1	A0	R/W
1	1	1	0	0	X[1]	X[1]	0/1

Fig. 5.46: TDA19988 I2C Address

Interrupt Signal

There is a HDMI_INT signal that connects from the TDA19988 to the processor. This signal can be used to alert the processor in a state change on the HDMI interface.

Audio Interface

There is an I2S audio interface between the processor and the *TDA19988*. Stereo audio can be transported over the HDMI interface to an audio equipped display. In order to create the required clock frequencies, an external 24.576MHz oscillator,*Y4*, is used. From this clock, the processor generates the required clock frequencies for the *TDA19988*.

There are three signals used to pass data from the processor to the *TDA19988*. SCLK is the serial clock. SPI1_CS0 is the data pin to the **TDA199888**. SPI1_D0 is the word sync pin. These signals are configured as I2S interfaces.

Audio is limited to CEA supported resolutions. LCD panels only activate the audio in CEA modes. This is a function of the specification and is not something that can be fixed on the board via a hardware change or a software change.

In order to create the correct clock frequencies, we had to add an external *24.576MHz* oscillator. Unfortunately this had to be input into the processor using the pin previously used for **GPIO3_21**. In order to keep GPIO3_21 functionality, we provided a way to disable the oscillator if the need was there to use the pin on the expansion header. *Figure 45* shows the oscillator circuitry.

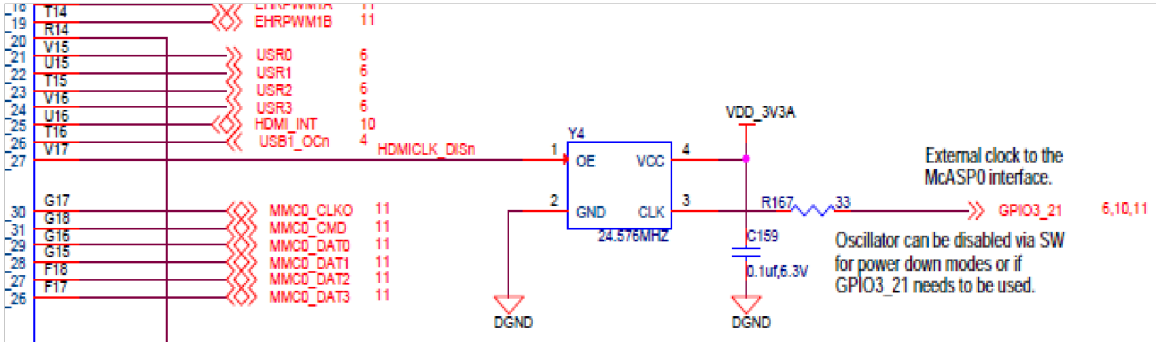


Fig. 5.47: 24.576MHZ Oscillator

Power Connections

figure-46 shows the power connections to the **TDA19988** device. All voltage rails for the device are at 1.8V. A filter is provided to minimize any noise from the 1.8V rail getting back into the device.

All of the interfaces between the processor and the *TDA19988* are 3.3V tolerant allowing for direct connection.

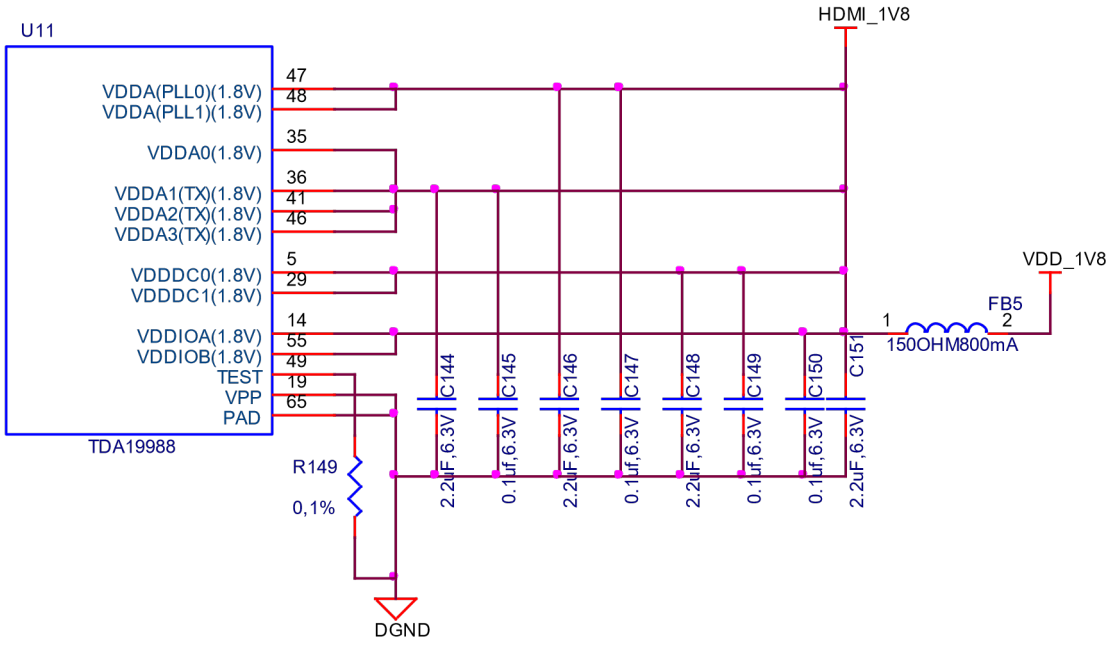


Fig. 5.48: HDMI Power Connections

HDMI Connector Interface

figure-47 shows the design of the interface between the HDMI Framer and the connector.

The connector for the HDMI interface is a microHDMI. It should be noted that this connector has a different pinout than the standard or mini HDMI connectors. D6 and D7 are ESD protection devices.

5.6.12 USB Host

The board is equipped with a single USB host interface accessible from a single USB Type A female connector. «figure-48» is the design of the USB Host circuitry.

Power Switch

U8 is a switch that allows the power to the connector to be turned on or off by the processor. It also has an over current detection that can alert the processor if the current gets too high via the**USB1_OC** signal. The power is controlled by the *USB1_DRVBUS* signal from the processor.

ESD Protection

U9 is the ESD protection for the signals that go to the connector.

Filter Options

FB7 and**FB8** were added to assist in passing the FCC emissions test. The *USB1_VBUS* signal is used by the processor to detect that the 5V is present on the connector. *FB7* is populated and *FB8* is replaced with a .1 ohm resistor.

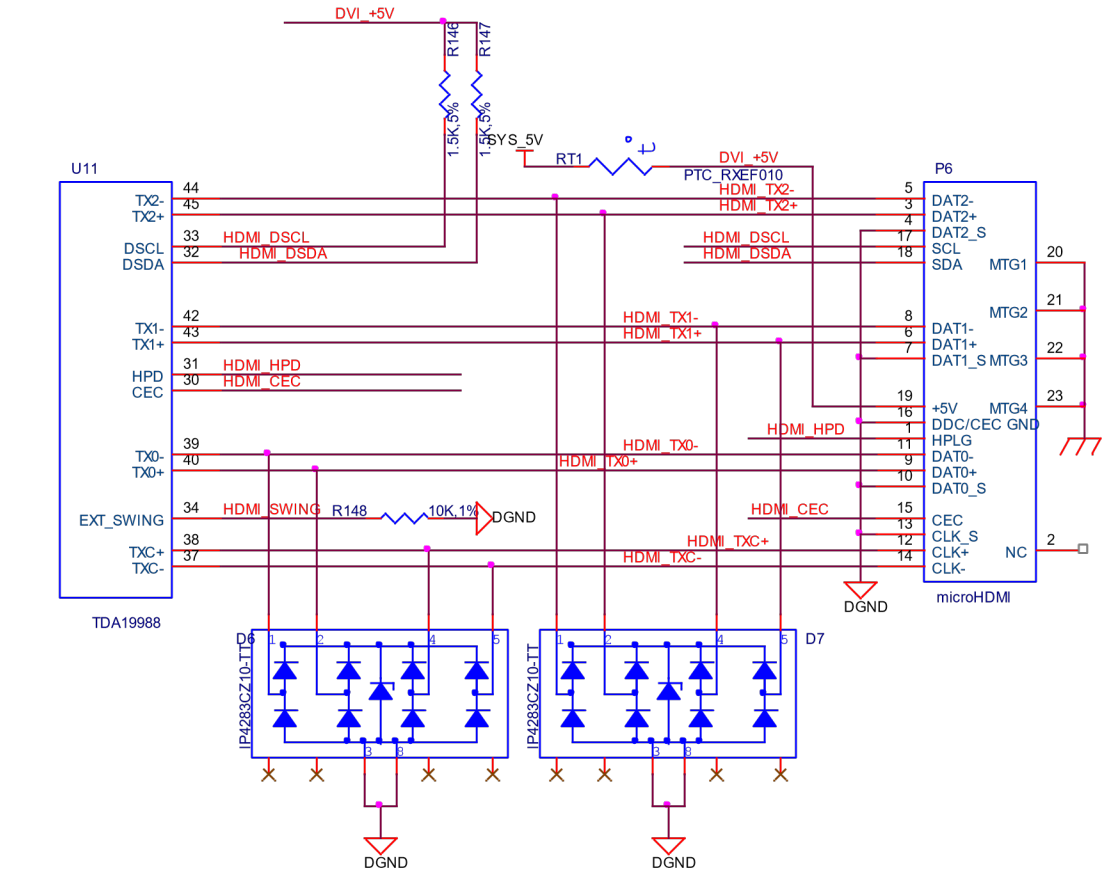


Fig. 5.49: Connector Interface Circuitry

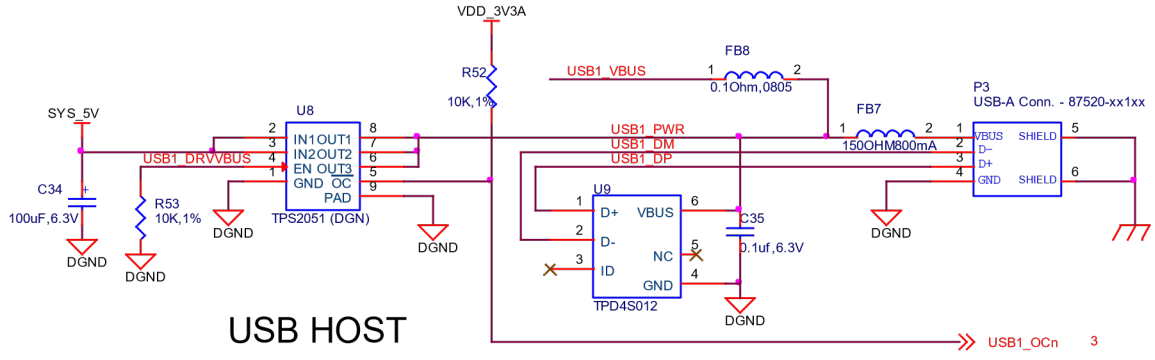


Figure 48. USB Host Circuitry

Fig. 5.50: USB Host circuit

5.6.13 PRU-ICSS

The PRU-ICSS module is located inside the AM3358 processor. Access to these pins is provided by the expansion headers and is multiplexed with other functions on the board. Access is not provided to all of the available pins.

All documentation is located at http://github.com/beagleboard/am335x_pru_package_

This feature is not supported by Texas Instruments.

PRU-ICSS Features

The features of the PRU-ICSS include:

Two independent programmable real-time (PRU) cores:

- 32-Bit Load/Store RISC architecture
- 8K Byte instruction RAM (2K instructions) per core
- 8K Bytes data RAM per core
- 12K Bytes shared RAM
- Operating frequency of 200 MHz
- PRU operation is little endian similar to ARM processor
- All memories within PRU-ICSS support parity
- Includes Interrupt Controller for system event handling
- Fast I/O interface

16 input pins and 16 output pins per PRU core. *(Not all of these are accessible on the BeagleBone Black).*

PRU-ICSS Block Diagram

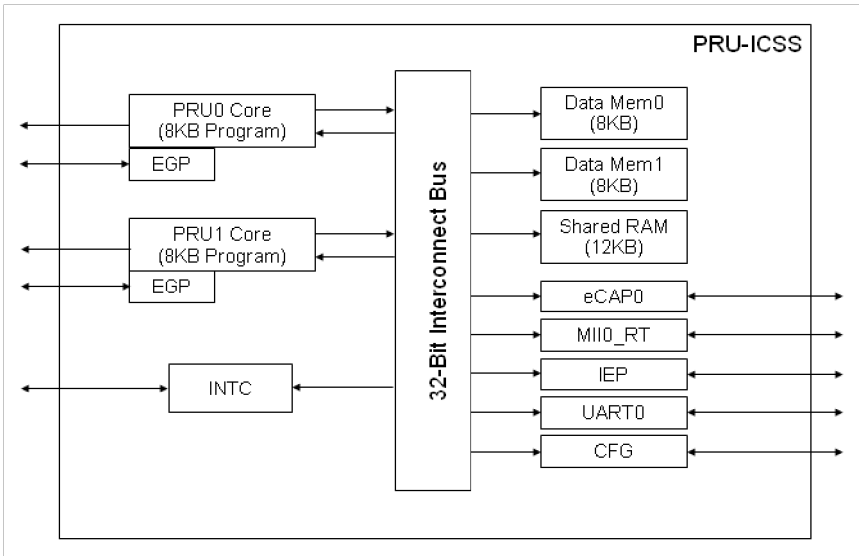


Fig. 5.51: PRU-ICSS Block Diagram

PRU-ICSS Pin Access

Both PRU 0 and PRU1 are accessible from the expansion headers. Some may not be usable without first disabling functions on the board like LCD for example. Listed below is what ports can be accessed on each PRU.

- 8 outputs or 9 inputs
- 13 outputs or 14 inputs
- UART0_TXD, UART0_RXD, UART0_CTS, UART0_RTS

Table 5.9: P8 PRU0 and PRU1 Access

PIN	PROC	NAME			
11	R12	GPIO1_13		pr1_pru0_pru_r30_15 (Output)	
12	T12	GPIO1_12		pr1_pru0_pru_r30_14 (Output)	
15	U13	GPIO1_15		pr1_pru0_pru_r31_15 (Input)	
16	V13	GPIO1_14		pr1_pru0_pru_r31_14 (Input)	
20	V9	GPIO1_31	pr1_pru1_pru_r30_13 (Output)	pr1_pru1_pru_r31_13 (INPUT)	
21	U9	GPIO1_30	pr1_pru1_pru_r30_12 (Output)	pr1_pru1_pru_r31_12 (INPUT)	
27	U5	GPIO2_22	pr1_pru1_pru_r30_8 (Output)	pr1_pru1_pru_r31_8 (INPUT)	
28	V5	GPIO2_24	pr1_pru1_pru_r30_10 (Output)	pr1_pru1_pru_r31_10 (INPUT)	
29	R5	GPIO2_23	pr1_pru1_pru_r30_9 (Output)	pr1_pru1_pru_r31_9 (INPUT)	
39	T3	GPIO2_12	pr1_pru1_pru_r30_6 (Output)	pr1_pru1_pru_r31_6 (INPUT)	
40	T4	GPIO2_13	pr1_pru1_pru_r30_7 (Output)	pr1_pru1_pru_r31_7 (INPUT)	
41	T1	GPIO2_10	pr1_pru1_pru_r30_4 (Output)	pr1_pru1_pru_r31_4 (INPUT)	
42	T2	GPIO2_11	pr1_pru1_pru_r30_5 (Output)	pr1_pru1_pru_r31_5 (INPUT)	
43	R3	GPIO2_8	pr1_pru1_pru_r30_2 (Output)	pr1_pru1_pru_r31_2 (INPUT)	
44	R4	GPIO2_9	pr1_pru1_pru_r30_3 (Output)	pr1_pru1_pru_r31_3 (INPUT)	
45	R1	GPIO2_6	pr1_pru1_pru_r30_0 (Output)	pr1_pru1_pru_r31_0 (INPUT)	
46	R2	GPIO2_7	pr1_pru1_pru_r30_1 (Output)	pr1_pru1_pru_r31_1 (INPUT)	

Table 5.10: P9 PRU0 and PRU1 Access

PIN	PROC	NAME			
17	A16	I2C1_SCL	pr1_uart0_txd		
18	B16	I2C1_SDA	pr1_uart0_rxd		
19	D17	I2C2_SCL	pr1_uart0_rts_n		
20	D18	I2C2_SDA	pr1_uart0_cts_n		
21	B17	UART2_TXD	pr1_uart0_rts_n		

continues on next page

Table 5.10 – continued from previous page

PIN	PROC	NAME			
22	A17	UART2_RXD	pr1_uart0_cts_n		
24	D15	UART1_TXD	pr1_uart0_txd	pr1_pru0_pru_r31_16 (Input)	
25	A14	GPIO3_21	pr1_pru0_pru_r30_5 (Output)	pr1_pru0_pru_r31_5 (Input)	
26	D16	UART1_RXD	pr1_uart0_rxd	pr1_pru1_pru_r31_16	
27	C13	GPIO3_19	pr1_pru0_pru_r30_7 (Output)	pr1_pru0_pru_r31_7 (Input)	
28	C12	SPI1_CS0	eCAP2_in_PWM2_out	pr1_pru0_pru_r30_3 (Output)	pr1_pru0_pru_r31_3 (Input)
29	B13	SPI1_D0	pr1_pru0_pru_r30_1 (Output)	pr1_pru0_pru_r31_1 (Input)	
30	D12	SPI1_D1	pr1_pru0_pru_r30_2 (Output)	pr1_pru0_pru_r31_2 (Input)	
31	A13	SPI1_SCLK	pr1_pru0_pru_r30_0 (Output)	pr1_pru0_pru_r31_0 (Input)	

Note: GPIO3_21 is also the 24.576MHZ clock input to the processor to enable HDMI audio. To use this pin the oscillator must be disabled.

5.7 Connectors

This section describes each of the connectors on the board.

5.7.1 Expansion Connectors

The expansion interface on the board is comprised of two 46 pin connectors. All signals on the expansion headers are _3.3V_ unless otherwise indicated.

NOTE: Do not connect 5V logic level signals to these pins or the board will be damaged.

NOTE: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

The location and spacing of the expansion headers are the same as on the original BeagleBone.

Connector P8

table-12 shows the pinout of the **P8** expansion header. Other signals can be connected to this connector based on setting the pin mux on the processor, but this is the default settings on power up. The SW is responsible for setting the default function of each pin. There are some signals that have not been listed here. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The *PROC* column is the pin number on the processor.

The *PIN* column is the pin number on the expansion header.

The *MODE* columns are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

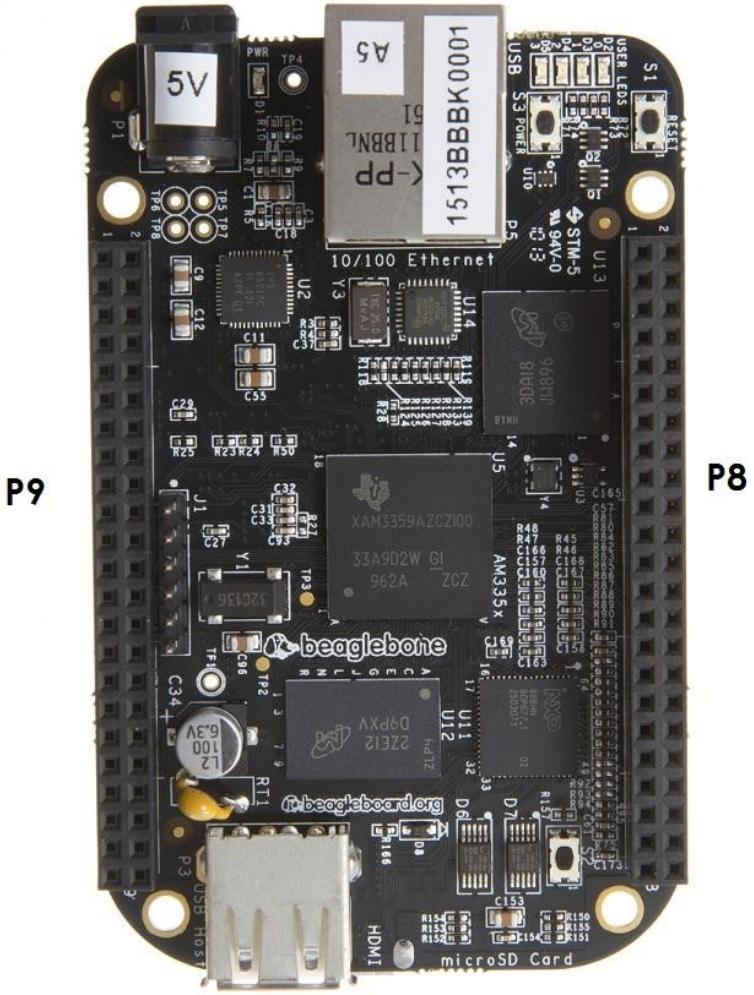


Fig. 5.52: Expansion Connector Location

NOTE: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

Table 5.11: Expansion Header P8 Pinout

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
1,2		GND								
3	R9	GPIO1_6	gpmc_ad6	mmc1_dat6						gpio1[6]
4	T9	GPIO1_7	gpmc_ad7	mmc1_dat7						gpio1[7]
5	R8	GPIO1_2	gpmc_ad2	mmc1_dat2						gpio1[2]
6	T8	GPIO1_3	gpmc_ad3	mmc1_dat3						gpio1[3]
7	R7	TIMER4	gpmc_advn_ale		timer4					gpio2[2]
8	T7	TIMER7	gpmc_oen_ren		timer7					gpio2[3]
9	T6	TIMER5	gpmc_be0n_cle		timer5					gpio2[5]
10	U6	TIMER6	gpmc_wen		timer6					gpio2[4]
11	R12	GPIO1_13	gpmc_ad13	lcd_data18	mmc1_dat5	mmc2_dat1	eQEP2B_in		pr1_pru0_pru_r30_15	gpio1[13]
12	T12	GPIO1_12	gpmc_ad12	lcd_data19	mmc1_dat4	mmc2_dat0	eQEP2a_in		pr1_pru0_pru_r30_14	gpio1[12]
13	T10	EHRPWM2B	gpmc_ad9	lcd_data22	mmc1_dat1	mmc2_dat5	ehrpwm2B			gpio0[23]
14	T11	GPIO0_26	gpmc_ad10	lcd_data21	mmc1_dat2	mmc2_dat6	ehrpwm2_tripzone_in			gpio0[26]
15	U13	GPIO1_15	gpmc_ad15	lcd_data16	mmc1_dat7	mmc2_dat3	eQEP2_strobe		pr1_pru0_pru_r31_15	gpio1[15]
16	V13	GPIO1_14	gpmc_ad14	lcd_data17	mmc1_dat6	mmc2_dat2	eQEP2_index		pr1_pru0_pru_r31_14	gpio1[14]
17	U12	GPIO0_27	gpmc_ad11	lcd_data20	mmc1_dat3	mmc2_dat7	ehrpwm0_synco			gpio0[27]
18	V12	GPIO2_1	gpmc_clk_mux0	lcd_memory_clk	gpmc_wait1	mmc2_clk			mcasp0_fsr	gpio2[1]
19	U10	EHRPWM2A	gpmc_ad8	lcd_data23	mmc1_dat0	mmc2_dat4	ehrpwm2A			gpio0[22]
20	V9	GPIO1_31	gpmc_csn2	gpmc_be1n	mmc1_cmd			pr1_pru1_pru_r30_13	pr1_pru1_pru_r31_13	gpio1[31]
21	U9	GPIO1_30	gpmc_csn1	gpmc_clk	mmc1_clk			pr1_pru1_pru_r30_12	pr1_pru1_pru_r31_12	gpio1[30]
22	V8	GPIO1_5	gpmc_ad5	mmc1_dat5						gpio1[5]
23	U8	GPIO1_4	gpmc_ad4	mmc1_dat4						gpio1[4]
24	V7	GPIO1_1	gpmc_ad1	mmc1_dat1						gpio1[1]
25	U7	GPIO1_0	gpmc_ad0	mmc1_dat0						gpio1[0]
26	V6	GPIO1_29	gpmc_csn0							gpio1[29]
27	U5	GPIO2_22	lcd_vsync	gpmc_a8				pr1_pru1_pru_r30_8	pr1_pru1_pru_r31_8	gpio2[22]
28	V5	GPIO2_24	lcd_pclk	gpmc_a10				pr1_pru1_pru_r30_10	pr1_pru1_pru_r31_10	gpio2[24]
29	R5	GPIO2_23	lcd_hsync	gpmc_a9				pr1_pru1_pru_r30_9	pr1_pru1_pru_r31_9	gpio2[23]
30	R6	GPIO2_25	lcd_ac_bias_en	gpmc_a11						gpio2[25]
31	V4	UART5_CTSN	lcd_data14	gpmc_a18	eQEP1_index	mcasp0_axr1	uart5_rxd		uart5_ctsn	gpio0[10]
32	T5	UART5_RTSN	lcd_data15	gpmc_a19	eQEP1_strobe	mcasp0_ahclkx	mcasp0_axr3		uart5_rtsn	gpio0[11]
33	V3	UART4_RTSN	lcd_data13	gpmc_a17	eQEP1B_in	mcasp0_fsr	mcasp0_axr3		uart4_rtsn	gpio0[9]
34	U4	UART3_RTSN	lcd_data11	gpmc_a15	ehrpwm1B	mcasp0_ahclkr	mcasp0_axr2		uart3_rtsn	gpio2[17]
35	V2	UART4_CTSN	lcd_data12	gpmc_a16	eQEP1A_in	mcasp0_aclkr	mcasp0_axr2		uart4_ctsn	gpio0[8]
36	U3	UART3_CTSN	lcd_data10	gpmc_a14	ehrpwm1A	mcasp0_axr0			uart3_ctsn	gpio2[16]
37	U1	UART5_TXD	lcd_data8	gpmc_a12	ehrpwm1_tripzone_in	mcasp0_aclkx	uart5_txd		uart2_ctsn	gpio2[14]
38	U2	UART5_RXD	lcd_data9	gpmc_a13	ehrpwm0_synco	mcasp0_fsx	uart5_rxd		uart2_rtsn	gpio2[15]
39	T3	GPIO2_12	lcd_data6	gpmc_a6		eQEP2_index		pr1_pru1_pru_r30_6	pr1_pru1_pru_r31_6	gpio2[12]
40	T4	GPIO2_13	lcd_data7	gpmc_a7		eQEP2_strobe	pr1_edio_data_out7	pr1_pru1_pru_r30_7	pr1_pru1_pru_r31_7	gpio2[13]
41	T1	GPIO2_10	lcd_data4	gpmc_a4		eQEP2A_in		pr1_pru1_pru_r30_4	pr1_pru1_pru_r31_4	gpio2[10]
42	T2	GPIO2_11	lcd_data5	gpmc_a5		eQEP2B_in		pr1_pru1_pru_r30_5	pr1_pru1_pru_r31_5	gpio2[11]
43	R3	GPIO2_8	lcd_data2	gpmc_a2		ehrpwm2_tripzone_in		pr1_pru1_pru_r30_2	pr1_pru1_pru_r31_2	gpio2[8]
44	R4	GPIO2_9	lcd_data3	gpmc_a3		ehrpwm0_synco		pr1_pru1_pru_r30_3	pr1_pru1_pru_r31_3	gpio2[9]
45	R1	GPIO2_6	lcd_data0	gpmc_a0		ehrpwm2A		pr1_pru1_pru_r30_0	pr1_pru1_pru_r31_0	gpio2[6]
46	R2	GPIO2_7	lcd_data1	gpmc_a1		ehrpwm2B		pr1_pru1_pru_r30_1	pr1_pru1_pru_r31_1	gpio2[7]

Connector P9

Table-13 lists the signals on connector **P9**. Other signals can be connected to this connector based on setting the pin mux on the processor, but this is the default settings on power up.

There are some signals that have not been listed here. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The *PROC* column is the pin number on the processor.

The *PIN* column is the pin number on the expansion header.

The *MODE* columns are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

NOTES:

In the table are the following notations:

PWR_BUT is a 5V level as pulled up internally by the TPS65217C. It is activated by pulling the signal to GND.

NOTE: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

- Both of these signals connect to pin 41 of P11. Resistors are installed that allow for the GPIO3_20 connection to be removed by removing R221. The intent is to allow the SW to use either of these signals, one or the other, on pin 41. SW should set the unused pin in input mode when using the other pin. This allowed us to get an extra signal out to the expansion header.
- Both of these signals connect to pin 42 of P11. Resistors are installed that allow for the GPIO3_18 connection to be removed by removing R202. The intent is to allow the SW to use either of these signals, on pin 42. SW should set the unused pin in input mode when using the other pin. This allowed us to get an extra signal out to the expansion header.

Table 5.12: Expansion Header P9 Pinout

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
1,2	**GND**									
3,4	**DC_3.3V**									
5,6	**VDD_5V**									
7,8	**SYS_5V**									
9	**PWR_BUT**									
10	**A10**	**SYS_RESETn**								
11	**T17**	**UART4_RXD**	gpmc_wait0	mii2_crs	gpmc_csn4	rmii2_crs_dv	mmc1_sdcd		uart4_rxd_mux2	gpio0[30]
12	**U18**	**GPIO1_28**	gpmc_be1n	mii2_col	gpmc_csn6	mmc2_dat3	gpmc_dir		mcasp0_aclkr_mux3	gpio1[28]
13	**U17**	**UART4_TXD**	gpmc_wpn	mii_rxerr	gpmc_csn5	rmii2_rxerr	mmc2_sdcd		uart4_txd_mux2	gpio0[31]
14	**U14**	**EHRPWM1A**	gpmc_a2	mii2_txd3	rgmii2_td3	mmc2_dat1	gpmc_a18		ehrpwm1A_mux1	gpio1[18]
15	**R13**	**GPIO1_16**	gpmc_a0	gmii2_txen	rmii2_tctl	mii2_txen	gpmc_a16		ehrpwm1_tripzone_input	gpio1[16]
16	**T14**	**EHRPWM1B**	gpmc_a3	mii2_txd2	rgmii2_td2	mmc2_dat2	gpmc_a19		ehrpwm1B_mux1	gpio1[19]
17	**A16**	**I2C1_SCL**	spi0_cs0	mmc2_sdwp	I2C1_SCL	ehrpwm0_synci	pr1_uart0_txd			gpio0[5]
18	**B16**	**I2C1_SDA**	spi0_d1	mmc1_sdwp	I2C1_SDA	ehrpwm0_tripzone	pr1_uart0_rxd			gpio0[4]
19	**D17**	**I2C2_SCL**	uart1_rtsn	timer5	dcan0_rx	I2C2_SCL	spi1_cs1	pr1_uart0_rts_n		gpio0[13]
20	**D18**	**I2C2_SDA**	uart1_ctsn	timer6	dcan0_tx	I2C2_SDA	spi1_cs0	pr1_uart0_cts_n		gpio0[12]
21	**B17**	**UART2_TXD**	spi0_d0	uart2_txd	I2C2_SCL	ehrpwm0B	pr1_uart0_rts_n		EMU3_mux1	gpio0[3]
22	**A17**	**UART2_RXD**	spi0_sclk	uart2_rxd	I2C2_SDA	ehrpwm0A	pr1_uart0_cts_n		EMU2_mux1	gpio0[2]
23	**V14**	**GPIO1_17**	gpmc_a1	gmii2_rxdv	rgmii2_rxdv	mmc2_dat0	gpmc_a17		ehrpwm0_synco	gpio1[17]
24	**D15**	**UART1_TXD**	uart1_txd	mmc2_sdwp	dcan1_rx	I2C1_SCL		pr1_uart0_txd	pr1_pru0_pru_r31_16	gpio0[15]
25	**A14**	**GPIO3_21**	mcasp0_ahclkx	eQEP0_strobe	mcasp0_axr3	mcasp1_axr1	EMU4_mux2	pr1_pru0_pru_r30_7	pr1_pru0_pru_r31_7	gpio3[21]
26	**D16**	**UART1_RXD**	uart1_rxd	mmc1_sdwp	dcan1_tx	I2C1_SDA		pr1_uart0_rxd	pr1_pru1_pru_r31_16	gpio0[14]
27	**C13**	**GPIO3_19**	mcasp0_fsr	eQEP0B_in	mcasp0_axr3	mcasp1_fsx	EMU2_mux2	pr1_pru0_pru_r30_5	pr1_pru0_pru_r31_5	gpio3[19]
28	**C12**	**SPI1_CS0**	mcasp0_ahclkr	ehrpwm0_synci	mcasp0_axr2	spi1_cs0	eCAP2_in_PWM2_out	pr1_pru0_pru_r30_3	pr1_pru0_pru_r31_3	gpio3[17]
29	**B13**	**SPI1_D0**	mcasp0_fsx	ehrpwm0B		spi1_d0	mmc1_sdcd_mux1	pr1_pru0_pru_r30_1	pr1_pru0_pru_r31_1	gpio3[15]
30	**D12**	**SPI1_D1**	mcasp0_axr0	ehrpwm0_tripzone		spi1_d1	mmc2_sdcd_mux1	pr1_pru0_pru_r30_2	pr1_pru0_pru_r31_2	gpio3[16]
31	**A13**	**SPI1_SCLK**	mcasp0_aclkx	ehrpwm0A		spi1_sclk	mmc0_sdcd_mux1	pr1_pru0_pru_r30_0	pr1_pru0_pru_r31_0	gpio3[14]
32	**VADC**									
33	**C8**	**AIN4**								
34	**AGND**									
35	**A8**	**AIN6**								
36	**B8**	**AIN5**								
37	**B7**	**AIN2**								
38	**A7**	**AIN3**								
39	**B6**	**AIN0**								
40	**C7**	**AIN1**								
41	**D14**	**CLKOUT2**	xdma_event_intr1		tclkin	clkout2	timer7_mux1	pr1_pru0_pru_r31_16	EMU3_mux0	gpio0[20]
	D13	**GPIO3_20**	mcasp0_axr1	eQEP0_index		mcasp1_axr0	emu3	pr1_pru0_pru_r30_6	pr1_pru0_pru_r31_6	gpio3[20]
42	**C18**	**GPIO0_7**	eCAP0_in_PWM0_out	uart3_txd	spi1_cs1	pr1_ecap0_ecap_capin_apwm_o	spi1_sclk	mmc0_sdwp	xdma_event_intr2	gpio0[7]
	B12	**GPIO3_18**	mcasp0_aclkr	eQEP0A_in	mcasp0_axr2	mcasp1_aclkx		pr1_pru0_pru_r30_4	pr1_pru0_pru_r31_4	gpio3[18]
43-46	**GND**									

5.7.2 Power Jack

The DC power jack is located next to the RJ45 Ethernet connector as shown in «figure-51». This uses the same power connector as is used on the original BeagleBone. The connector has a 2.1mm diameter center post (5VDC) and a 5.5mm diameter outer dimension on the barrel (GND).

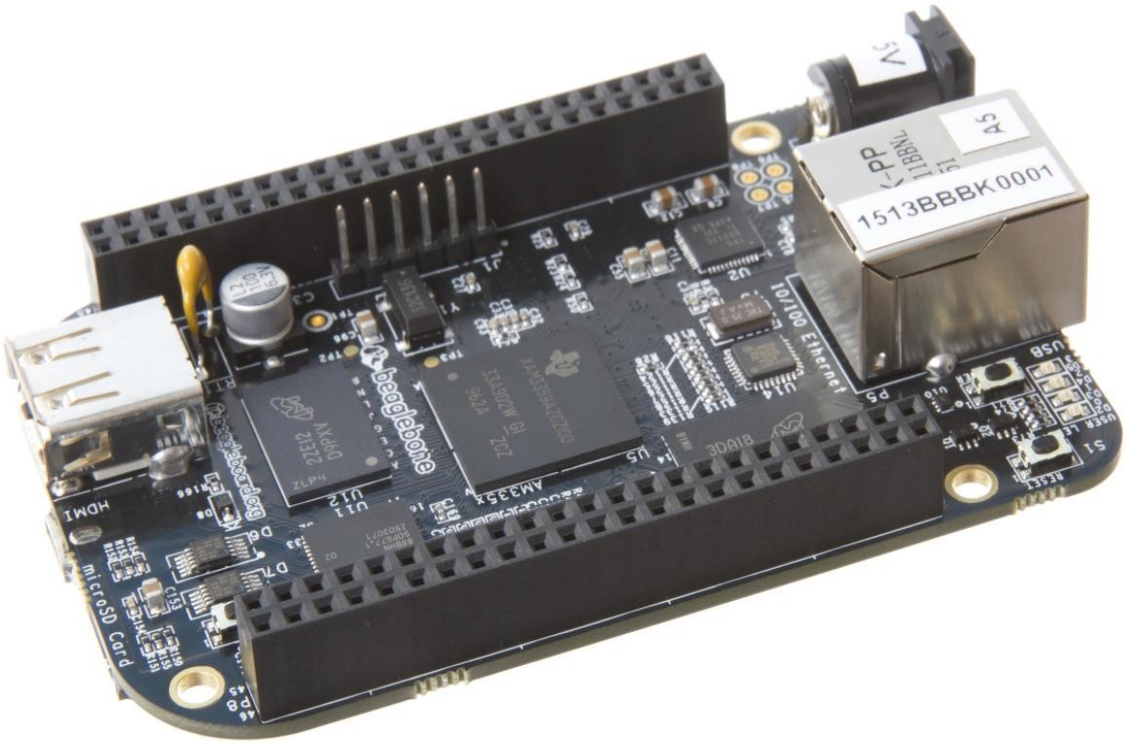


Fig. 5.53: 5VDC Power Jack

The board requires a regulated 5VDC +/-.25V supply at 1A. A higher current rating may be needed if capes are plugged into the expansion headers. Using a higher current power supply will not damage the board.

5.7.3 USB Client

The USB Client connector is accessible on the bottom side of the board under the row of four LEDs as shown in «figure-52». It uses a 5 pin miniUSB cable, the same as is used on the original BeagleBone. The cable is provided with the board. The cable can also be used to power the board.

This port is a USB Client only interface and is intended for connection to a PC.

5.7.4 USB Host

There is a single USB Host connector on the board and is shown in *Figure 53* below.

The port is USB 2.0 HS compatible and can supply up to 500mA of current. If more current or ports is needed, then a HUB can be used.

5.7.5 Serial Header

Each board has a debug serial interface that can be accessed by using a special serial cable that is plugged into the serial header as shown in *Figure 54* below.

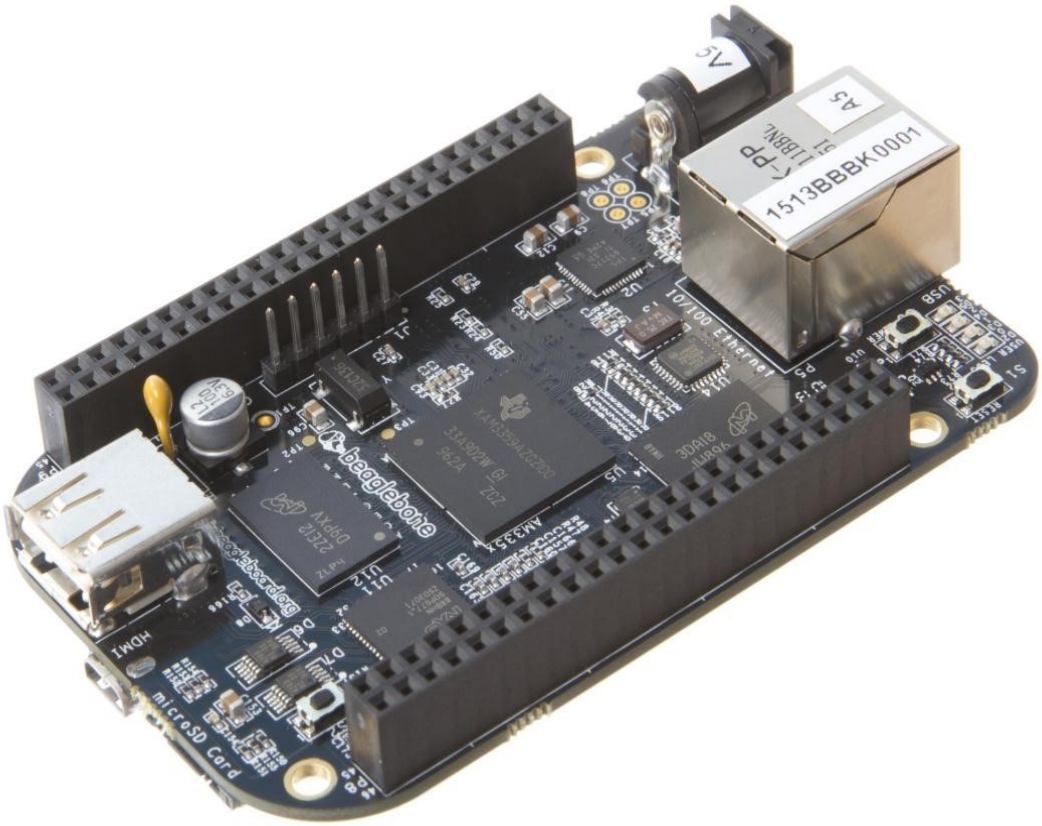


Fig. 5.54: USB Client

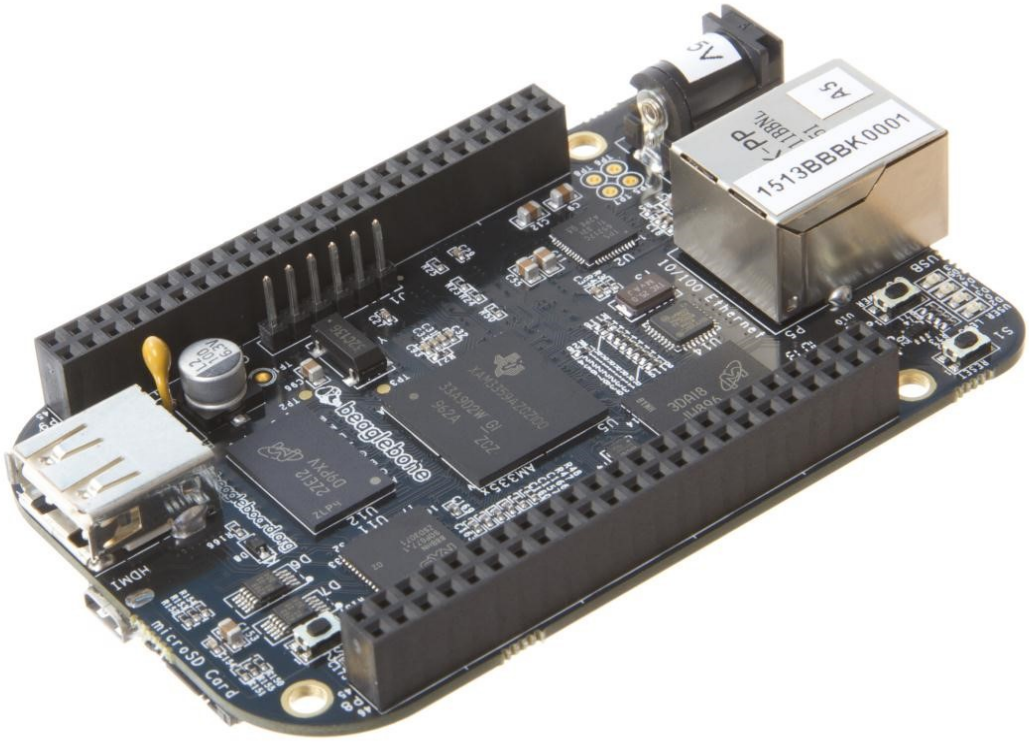


Fig. 5.55: USB Host Connector

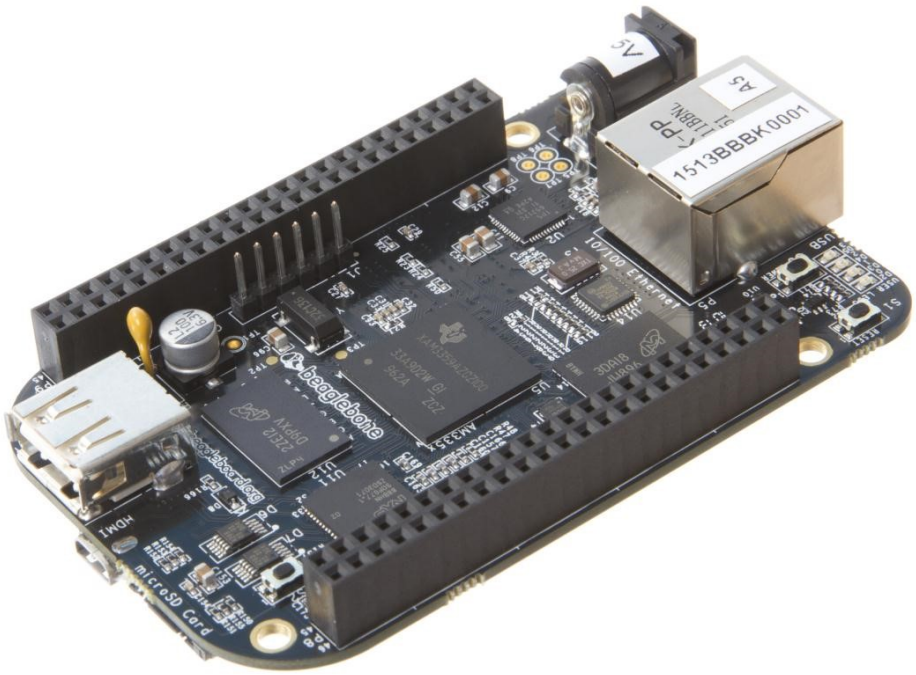


Fig. 5.56: Serial Debug Header

Todo: Make all figure references actual references

Two signals are provided, TX and RX on this connector. The levels on these signals are 3.3V. In order to access these signals, a FTDI USB to Serial cable is recommended as shown in *Figure 55* below.



Fig. 5.57: PRU-ICSS Block Diagram

The cable can be purchased from several different places and must be the 3.3V version TTL-232R-3V3. Information on the cable itself can be found direct from FTDI at: pdf

Todo: move accessory links to a single common document for all boards.

Pin 1 of the cable is the black wire. That must align with the pin 1 on the board which is designated by the white dot next to the connector on the board.

Refer to the support WIKI http://elinux.org/BeagleBoneBlack for more sources of this cable and other options that will work.

Todo: We should include all support information in docs.beagleboard.org now and leave eLinux to others,

freeing it as much as possible

Table is the pinout of the connector as reflected in the schematic. It is the same as the FTDI cable which can be found at https://ftdichip.com/wp-content/uploads/2020/07/DS_USB_RS232_CABLES.pdf with the exception that only three pins are used on the board. The pin numbers are defined in *Table 14*. The signals are from the perspective of the board.

Table 5.13: J1 Serial Header Pins

PIN NUMBER	SIGNAL
1	Ground
4	Receive
5	Transmit

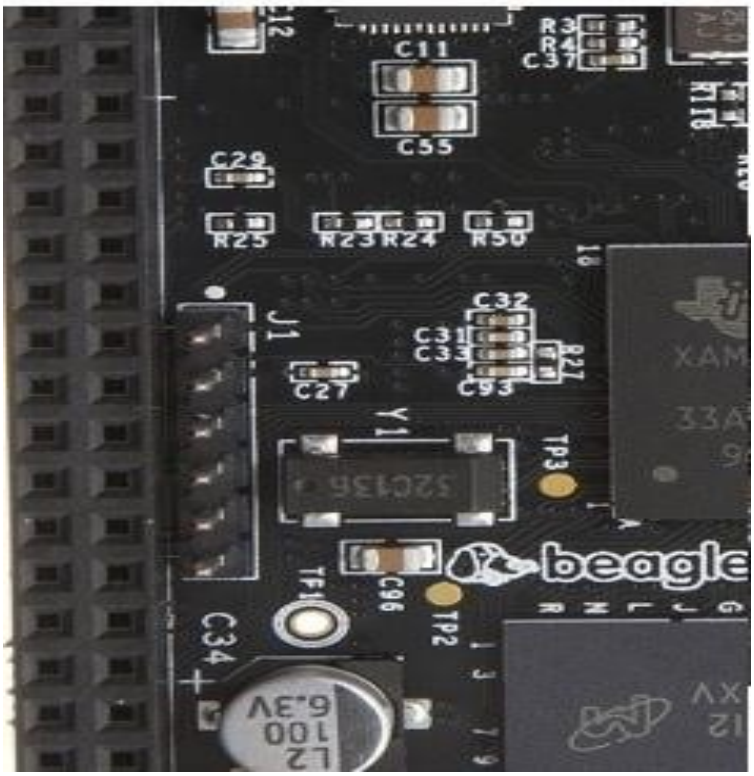


Fig. 5.58: Serial Header

5.7.6 HDMI

Access to the HDMI interface is through the HDMI connector that is located on the bottom side of the board as shown in *Figure 57* below.

The connector is microHDMI connector. This was done due to the space limitations we had in finding a place to fit the connector. It requires a microHDMI to HDMI cable as shown in *Figure 58* below. The cable can be purchased from several different sources.

5.7.7 microSD

A microSD connector is located on the back or bottom side of the board as shown in *Figure 59* below. The microSD card is not supplied with the board.

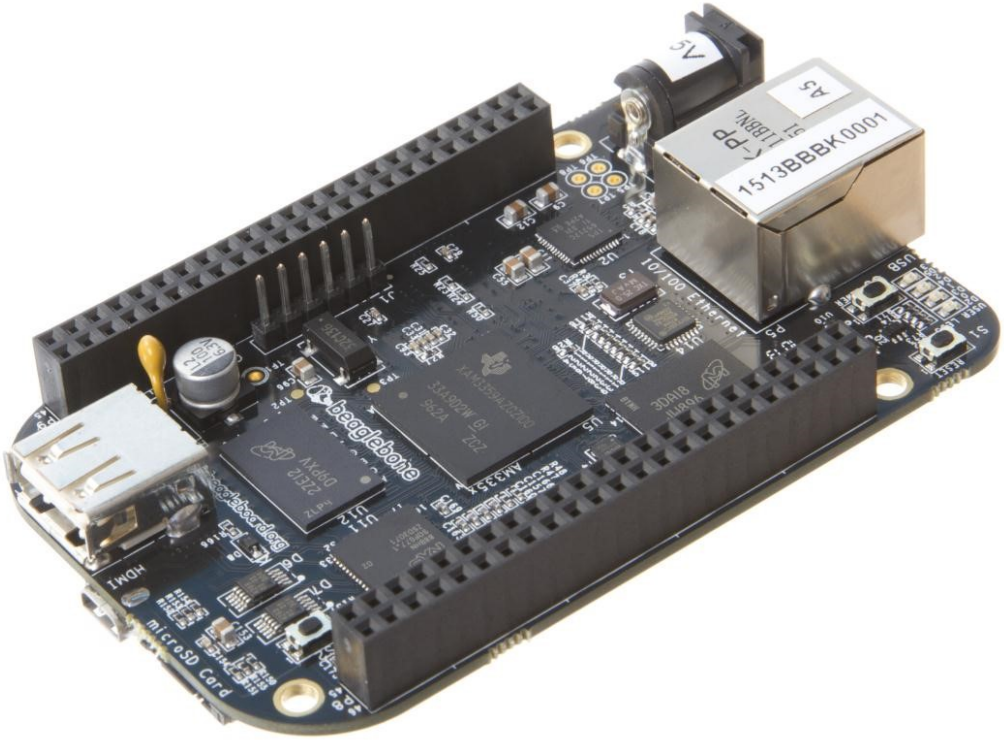


Fig. 5.59: HDMI Connector



Fig. 5.60: HDMI Cable

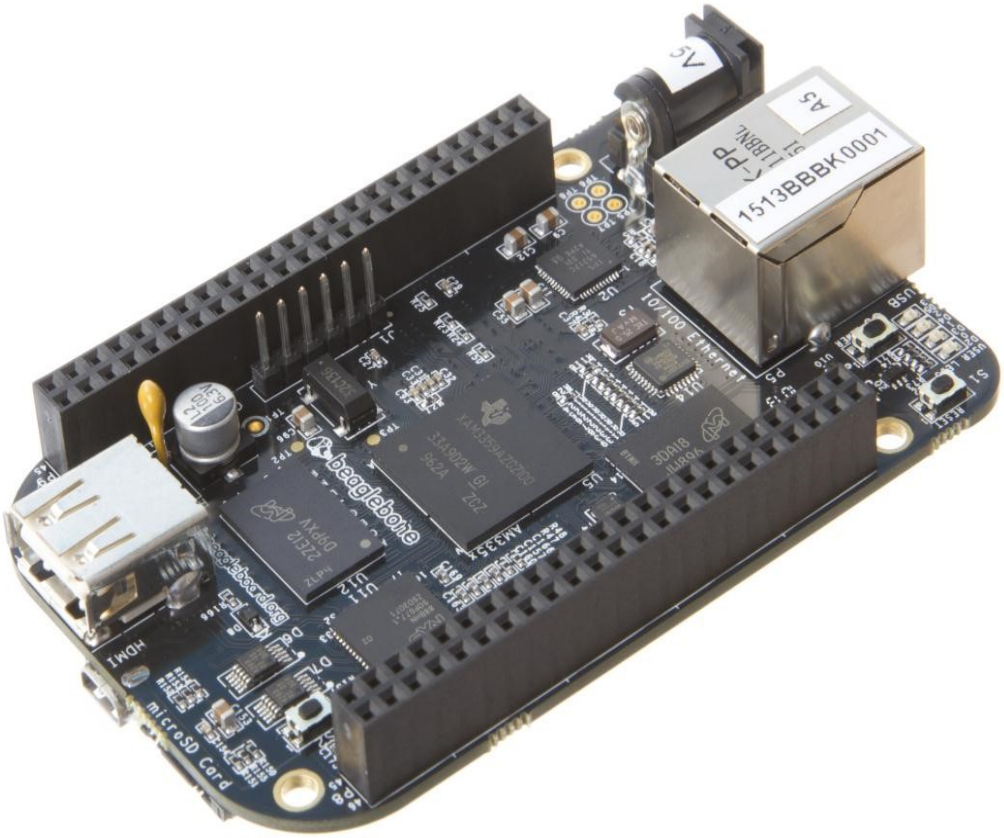


Fig. 5.61: microSD Connector

When plugging in the SD card, the writing on the card should be up. Align the card with the connector and push to insert. Then release. There should be a click and the card will start to eject slightly, but it then should latch into the connector. To eject the card, push the SD card in and then remove your finger. The SD card will be ejected from the connector.

Do not pull the SD card out or you could damage the connector.

5.7.8 Ethernet

The board comes with a single 10/100 Ethernet interface located next to the power jack as shown in Figure below.

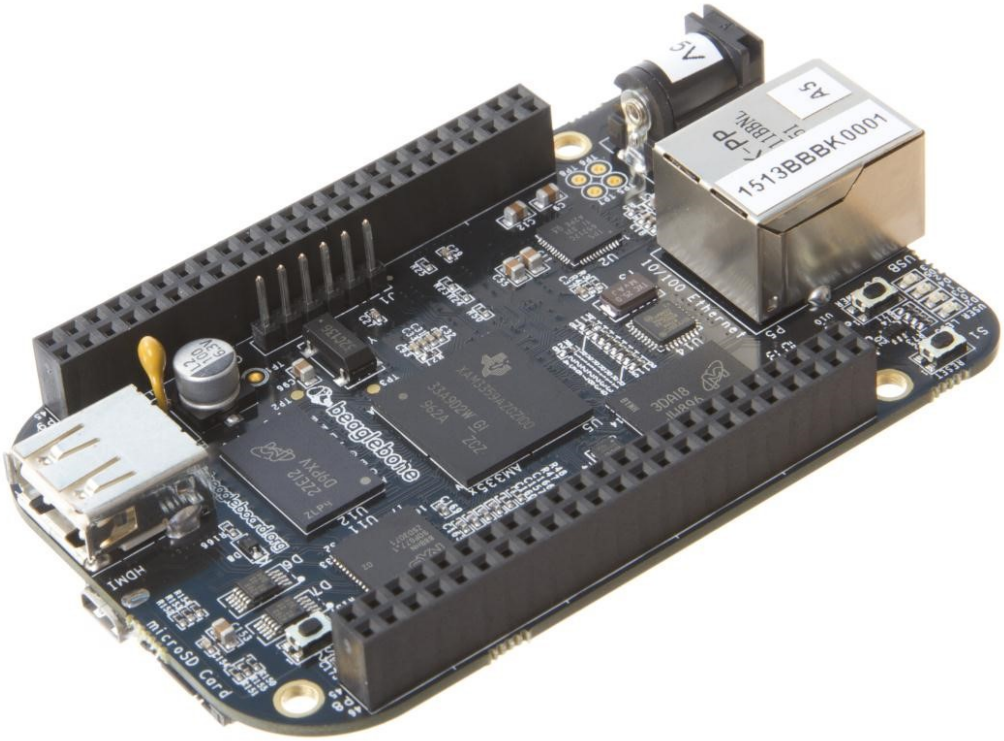


Fig. 5.62: Ethernet Connector

The PHY supports AutoMDX which means either a straight or a swap cable can be used.

5.7.9 JTAG Connector

A place for an optional 20 pin CTI JTAG header is provided on the board to facilitate the SW development and debugging of the board by using various JTAG emulators. This header is not supplied standard on the board. To use this, a connector will need to be soldered onto the board.

If you need the JTAG connector you can solder it on yourself. No other components are needed. The connector is made by Samtec and the part number is FTR-110-03-G-D-06. You can purchase it from http://www.digikey.com/

5.8 Cape Board Support

The BeagleBone Black has the ability to accept up to four expansion boards or capes that can be stacked onto the expansion headers. The word cape comes from the shape of the board as it is fitted around the Ethernet connector on the main board. This notch acts as a key to ensure proper orientation of the cape.

This section describes the rules for creating capes to ensure proper operation with the BeagleBone Black and proper interoperability with other capes that are intended to coexist with each other. Co-existence is not a requirement and is in itself, something that is impossible to control or administer. But, people will be able to create capes that operate with other capes that are already available based on public information as it pertains to what pins and features each cape uses. This information will be able to be read from the EEPROM on each cape.

This section is intended as a guideline for those wanting to create their own capes. Its intent is not to put limits on the creation of capes and what they can do, but to set a few basic rules that will allow the SW to administer their operation with the BeagleBone Black. For this reason there is a lot of flexibility in the specification that we hope most people will find liberating and in the spirit of Open Source Hardware. I am sure there are others that would like to see tighter control, more details, more rules and much more order to the way capes are handled.

Over time, this specification will change and be updated, so please refer to the latest version of this manual prior to designing your own capes to get the latest information.

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN

POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

5.8.1 BeagleBone Black Cape Compatibility

The main expansion headers are the same between the BeagleBone and BeagleBone Black. While the pins are the same, some of these pins are now used on the BeagleBone Black. The following sections discuss these pins.

The Power Expansion header was removed from the BeagleBone Black and is not available.

PAY VERY CLOSE ATTENTION TO THIS SECTION AND READ CAREFULLY!!

LCD Pins

The LCD pins are used on the BeagleBone Black to drive the HDMI framer. These signals are listed in *Table 15* below.

Table 5.14: P8 LCD Conflict Pins

PIN	PROC	NAME	MODE0
27	U5	GPIO2_22	lcd_vsync
28	V5	GPIO2_24	lcd_pclk
29	R5	GPIO2_23	lcd_hsync
30	R6	GPIO2_25	lcd_ac_bias_en
31	V4	UART5_CTSN	lcd_data14
32	T5	UART5_RTSN	lcd_data15
33	V3	UART4_RTSN	lcd_data13
34	U4	UART3_RTSN	lcd_data11
35	V2	UART4_CTSN	lcd_data12
36	U3	UART3_CTSN	lcd_data10
37	U1	UART5_TXD	lcd_data8
38	U2	UART5_RXD	lcd_data9
39	T3	GPIO2_12	lcd_data6
40	T4	GPIO2_13	lcd_data7
41	T1	GPIO2_10	lcd_data4
42	T2	GPIO2_11	lcd_data5
43	R3	GPIO2_8	lcd_data2
44	R4	GPIO2_9	lcd_data3
45	R1	GPIO2_6	lcd_data0

continues on next page

Table 5.14 – continued from previous page

PIN	PROC	NAME	MODE0
46	R2	GPIO2_7	lcd_data1

If you are using these pins for other functions, there are a few things to keep in mind:

- On the HDMI Framer, these signals are all inputs so the framer will not be driving these pins.
- The HDMI framer will add a load onto these pins.
- There are small filter caps on these signals which could also change the operation of these pins if used for other functions.
- When used for other functions, the HDMI framer cannot be used.
- There is no way to power off the framer as this would result in the framer being powered through these input pins which would not a be a good idea.
- These pins are also the *SYSBOOT* pins. DO NOT drive them before the **SYS_RESETN** signal goes high. If you do, the board may not boot because you would be changing the boot order of the processor.

In order to use these pins, the SW will need to reconfigure them to whatever function you need the pins to do. To keep power low, the HDMI framer should be put in a low power mode via the SW using the *I2C0* interface.

eMMC Pins

The BeagleBone Black uses 10 pins to connect to the processor that also connect to the P8 expansion connector. These signals are listed below in *Table 16*. The proper mode is MODE2.

Table 5.15: P8 eMMC Conflict Pins

PIN	PROC	SIGNAL	MODE
22	V8	MMC1_DAT5	1
23	U8	MMC1_DAT4	1
24	V7	MMC1_DAT1	1
5	R8	MMC1_DAT2	1
4	T9	MMC1_DAT7	1
3	R9	MMC1_DAT6	1
6	T8	MMC1_DAT3	1
25	U7	MMC1_DAT0	1
20	V9	MMC1_CMD	2
21	U9	MMC1_CLK	2

If using these pins, several things need to be kept in mind when doing so:

- On the eMMC device, these signals are inputs and outputs.
- The eMMC device will add a load onto these pins.
- When used for other functions, the eMMC cannot be used. This means you must boot from the microSD slot.
- If using these pins, you need to put the eMMC into reset. This requires that the eMMC be accessible from the processor in order to set the eMMC to accept the eMMC pins.
- DO NOT drive the eMMC pins until the eMMC has been put into reset. This means that if you choose to use these pins, they must not drive any signal until enabled via Software. This requires a buffer or some other form of hold off function enabled by a GPIO pin on the expansion header.

On power up, the eMMC is NOT reset. If you hold the Boot button down, this will force a boot from the microSD. This is not convenient when a cape is plugged into the board. There are two solutions to this issue:

1. Wipe the eMMC clean. This will cause the board to default to microSD boot. If you want to use the eMMC later, it can be reprogrammed. 2. You can also tie LCD_DATA2 low on the cape during boot. This will be the

same as if you were holding the boot button. However, in order to prevent unforeseen issues, you need to gate this signal with RESET, when the data is sampled. After set goes high, the signal should be removed from the pin.

BEFORE the SW reinitializes the pins, it **MUST** put the eMMC in reset. This is done by taking eMMC_RSTn (GPIO1_20) LOW **after** the eMMC has been put into a mode to enable the reset line. This pin does not connect to the expansion header and is accessible only on the board.

DO NOT automatically drive any conflicting pins until the SW enables it. This puts the SW in control to ensure that the eMMC is in reset before the signals are used from the cape. You can use a GPIO pin for this. No, we will not designate a pin for this function. It will be determined on a cape by cape basis by the designer of the respective cape.

5.8.2 EEPROM

Each cape must have its own EEPROM containing information that will allow the SW to identify the board and to configure the expansion headers pins as needed. The one exception is proto boards intended for prototyping. They may or may not have an EEPROM on them. An EEPROM is required for all capes sold in order for them operate correctly when plugged into the BeagleBone Black.

The address of the EEPROM will be set via either jumpers or a dipswitch on each expansion board. *Figure 61* below is the design of the EEPROM circuit.

The EEPROM used is the same one as is used on the BeagleBone and the BeagleBone Black, a CAT24C256. The CAT24C256 is a 256 kb Serial CMOS EEPROM, internally organized as 32,768 words of 8 bits each. It features a 64-byte page write buffer and supports the Standard (100 kHz), Fast (400 kHz) and Fast-Plus (1 MHz) I2C protocol.

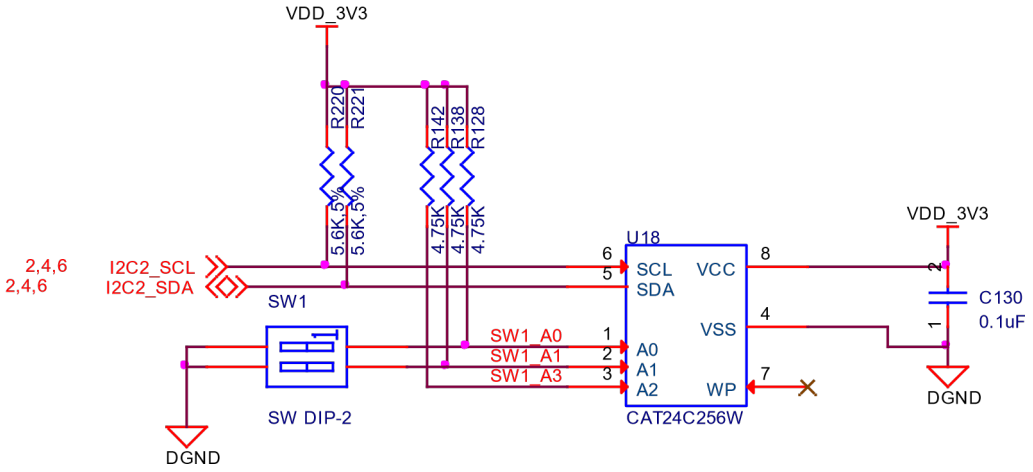


Fig. 5.63: Expansion Board EEPROM Without Write Protect

The addressing of this device requires two bytes for the address which is not used on smaller size EEPROMs, which only require only one byte. Other compatible devices may be used as well. Make sure the device you select supports 16 bit addressing. The part package used is at the discretion of the cape designer.

EEPROM Address

In order for each cape to have a unique address, a board ID scheme is used that sets the address to be different depending on the setting of the dipswitch or jumpers on the capes. A two position dipswitch or jumpers is used to set the address pins of the EEPROM.

It is the responsibility of the user to set the proper address for each board and the position in the stack that the board occupies has nothing to do with which board gets first choice on the usage of the expansion bus signals. The process for making that determination and resolving conflicts is left up to the SW and, as of this moment

in time, this method is a something of a mystery due to the new Device Tree methodology introduced in the 3.8 kernel.

Address line A2 is always tied high. This sets the allowable address range for the expansion cards to *0x54* to**0x57**. All other I2C addresses can be used by the user in the design of their capes. But, these addresses must not be used other than for the board EEPROM information. This also allows for the inclusion of EEPROM devices on the cape if needed without interfering with this EEPROM. It requires that A2 be grounded on the EEPROM not used for cape identification.

I2C Bus

The EEPROMs on each expansion board are connected to I2C2 on connector P9 pins 19 and 20. For this reason I2C2 must always be left connected and should not be changed by SW to remove it from the expansion header pin mux settings. If this is done, the system will be unable to detect the capes.

The I2C signals require pullup resistors. Each board must have a 5.6K resistor on these signals. With four capes installed this will result in an effective resistance of 1.4K if all capes were installed and all the resistors used were exactly 5.6K. As more capes are added the resistance is reduced to overcome capacitance added to the signals. When no capes are installed the internal pullup resistors must be activated inside the processor to prevent I2C timeouts on the I2C bus.

The I2C2 bus may also be used by capes for other functions such as I/O expansion or other I2C compatible devices that do not share the same address as the cape EEPROM.

EEPROM **********

The design in *Figure 62* has the write protect disabled. If the write protect is not enabled, this does expose the EEPROM to being corrupted if the I2C2 bus is used on the cape and the wrong address written to. It is recommended that a write protection function be implemented and a Test Point be added that when grounded, will allow the EEPROM to be written to. To enable write operation, Pin 7 of the EEPROM must be tied to ground.

When not grounded, the pin is HI via pullup resistor R210 and therefore write protected. Whether or not Write Protect is provided is at the discretion of the cape designer.

Variable & MAC Memory VDD_3V3B

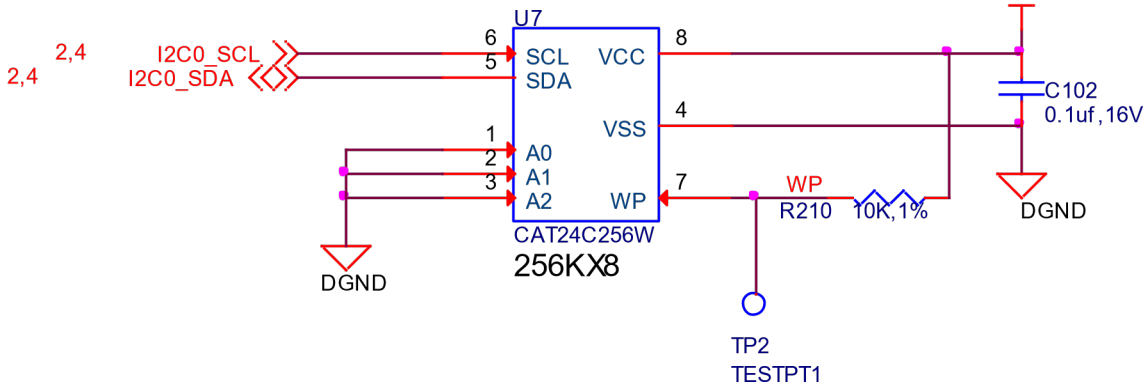


Fig. 5.64: Expansion Board EEPROM Write Protect

EEPROM Data Format

Table below shows the format of the contents of the expansion board EEPROM. Data is stored in Big Endian with the least significant value on the right. All addresses read as a single byte data from the EEPROM, but two byte addressing is used. ASCII values are intended to be easily read by the user when the EEPROM contents are dumped.

Name	Offset	Size (bytes)	Contents
Header	0	4	0xAA, 0x55, 0x33, 0xEE
EEPROM Revision	4	2	Revision number of the overall format of this EEPROM in ASCII =A1
Board Name	6	32	Name of board in ASCII so user can read it when the EEPROM is dumped. Up to
Version	38	4	Hardware version code for board in ASCII.Version format is up to the developer.
Manufacturer	42	16	ASCII name of the manufacturer. Company or individual's name.
Part Number	58	16	ASCII Characters for the part number. Up to maker of the board.
Number of Pins	74	2	Number of pins used by the daughter board including the power pins used. Dec
Serial Number	76	12	Serial number of the board. This is a 12 character string which is: **WWYY&&&**
Pin Usage	88	148	Two bytes for each configurable pins of the 74 pins on the expansion connector
VDD_3V3B Current	236	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500
VDD_5V Current	238	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500
SYS_5V Current	240	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500
DC Supplied	242	2	Indicates whether or not the board is supplying voltage on the VDD_5V rail and
Available	244	32543	Available space for other non-volatile codes/data to be used as needed by the r

Pin Usage

Table 18 is the locations in the EEPROM to set the I/O pin usage for the cape. It contains the value to be written to the Pad Control Registers. Details on this can be found in section **9.2.2** of the *AM3358 Technical Reference Manual*, The table is left blank as a convenience and can be printed out and used as a template for creating a custom setting for each cape. The 16 bit integers and all 16 bit fields are to be stored in Big Endian format.

Bit 15 PIN USAGE is an indicator and should be a 1 if the pin is used or 0 if it is unused.

Bits 14-7 RESERVED is not to be used and left as 0.

Bit 6 SLEW CONTROL 0=Fast 1=Slow

Bit 5 RX Enabled 0=Disabled 1=Enabled

Bit 4 PU/PD 0=Pulldown 1=Pullup.

Bit 3 PULLUP/DN 0=Pullup/pulldown enabled 1= Pullup/pulldown disabled

Bit 2-0 MUX MODE SELECT Mode 0-7. (refer to TRM)

Refer to the TRM for proper settings of the pin MUX mode based on the signal selection to be used.

The *AIN0-6* pins do not have a pin mux setting, but they need to be set to indicate if each of the pins is used on the cape. Only bit 15 is used for the AIN signals.

Table 5.17: EEPROM Pin Usage

+	+	+	15	14	13	12	11	10	9	8	7	6	5
Off set	Conn	Name	Pin Usage	Type	+	Reserved	+	+	S L E W	R X	P U - P D	P U / D E N	Mux Mode
88	P9-22	UART2_RXD	+	+	+	+	+	+	+	+	+	+	+
90	P9-21	UART2_TXD	+	+	+	+	+	+	+	+	+	+	+
92	P9-18	I2C1_SDA	+	+	+	+	+	+	+	+	+	+	+
94	P9-17	I2C1_SCL	+	+	+	+	+	+	+	+	+	+	+
96	P9-42	GPIO0_7	+	+	+	+	+	+	+	+	+	+	+
98	P8-35	UART4_CTSN	+	+	+	+	+	+	+	+	+	+	+
100	P8-33	UART4_RTSN	+	+	+	+	+	+	+	+	+	+	+
102	P8-31	UART5_CTSN	+	+	+	+	+	+	+	+	+	+	+
104	P8-32	UART5_RTSN	+	+	+	+	+	+	+	+	+	+	+
106	P9-19	I2C2_SCL	+	+	+	+	+	+	+	+	+	+	+
108	P9-20	I2C2_SDA	+	+	+	+	+	+	+	+	+	+	+
110	P9-26	UART*T1_RXD	+	+	+	+	+	+	+	+	+	+	+
112	P9-24	UART1_TXD	+	+	+	+	+	+	+	+	+	+	+
114	P9-41	CLKOUT2	+	+	+	+	+	+	+	+	+	+	+
116	P8-19	EHRPWM2A	+	+	+	+	+	+	+	+	+	+	+
118	P8-13	EHRPWM2B	+	+	+	+	+	+	+	+	+	+	+
120	P8-14	GPIO0_26	+	+	+	+	+	+	+	+	+	+	+
122	P8-17	GPIO0_27	+	+	+	+	+	+	+	+	+	+	+
124	P9-11	UART4_RXD	+	+	+	+	+	+	+	+	+	+	+
126	P9-13	UART4_TXD	+	+	+	+	+	+	+	+	+	+	+
128	P8-25	GPIO1_0	+	+	+	+	+	+	+	+	+	+	+
130	P8-24	GPIO1_1	+	+	+	+	+	+	+	+	+	+	+
132	P8-5	GPIO1_2	+	+	+	+	+	+	+	+	+	+	+
134	P8-6	GPIO1_3	+	+	+	+	+	+	+	+	+	+	+
136	P8-23	GPIO1_4	+	+	+	+	+	+	+	+	+	+	+
138	P8-22	GPIO1_5	+	+	+	+	+	+	+	+	+	+	+
140	P8-3	GPIO1_6	+	+	+	+	+	+	+	+	+	+	+
142	P8-4	GPIO1_7	+	+	+	+	+	+	+	+	+	+	+
144	P8-12	GPIO1_12	+	+	+	+	+	+	+	+	+	+	+
146	P8-11	GPIO1_13	+	+	+	+	+	+	+	+	+	+	+
148	P8-16	GPIO1_14	+	+	+	+	+	+	+	+	+	+	+
150	P8-15	GPIO1_15	+	+	+	+	+	+	+	+	+	+	+
152	P9-15	GPIO1_16	+	+	+	+	+	+	+	+	+	+	+

			15	14	13	12	11	10	9	8	7	6	5
Off set	Conn	Name	Pin Usage	Type	+	Reserve	+	+	S L E W	R X	P U - P D	P U / DE N	Mux Mode
154	P9-23	GPIO1_17											
156	P9-14	EHRPWM1A											
158	P9-16	EHRPWM1B											
160	P9-12	GPIO1_28											
162	P8-26	GPIO1_29											
164	P8-21	GPIO1_30											
166	P8-20	GPIO1_31											
168	P8-18	GPIO2_1											
170	P8-7	TIMER4											
172	P8-9	TIMER5	+	+	+	+	+	+	+	+	+	+	+
174	P8-10	TIMER6	+	+	+	+	+	+	+	+	+	+	+
176	P8-8	TIMER7	+	+	+	+	+	+	+	+	+	+	+
178	P8-45	GPIO2_6	+	+	+	+	+	+	+	+	+	+	+
180	P8-46	GPIO2_7	+	+	+	+	+	+	+	+	+	+	+
182	P8-43	GPIO2_8	+	+	+	+	+	+	+	+	+	+	+
184	P8-44	GPIO2_9	+	+	+	+	+	+	+	+	+	+	+
186	P8-41	GPIO2_10	+	+	+	+	+	+	+	+	+	+	+
188	P8-42	GPIO2_11	+	+	+	+	+	+	+	+	+	+	+
190	P8-39	GPIO2_12	+	+	+	+	+	+	+	+	+	+	+
192	P8-40	GPIO2_13	+	+	+	+	+	+	+	+	+	+	+
194	P8-37	UART5_TX'+'	+	+	+	+	+	+	+	+	+	+	+
196	P8-38	UART5_RX'+'	+	+	+	+	+	+	+	+	+	+	+
198	P8-36	UART3_CTSN	+	+	+	+	+	+	+	+	+	+	+
200	P8-34	UART3_RTSN	+	+	+	+	+	+	+	+	+	+	+
202	P8-27	GPIO2_22	+	+	+	+	+	+	+	+	+	+	+
204	P8-29	GPIO2_23	+	+	+	+	+	+	+	+	+	+	+
206	P8-28	GPIO2_24	+	+	+	+	+	+	+	+	+	+	+
208	P8-30	GPIO2_25	+	+	+	+	+	+	+	+	+	+	+
210	P9-29	SPI1_D0	+	+	+	+	+	+	+	+	+	+	+
212	P9-30	SPI1_D1	+	+	+	+	+	+	+	+	+	+	+
214	P9-28	SPI1_CS0	+	+	+	+	+	+	+	+	+	+	+
216	P9-27	GPIO3_19	+	+	+	+	+	+	+	+	+	+	+
218	P9-31	SPI1_SCLK	+	+	+	+	+	+	+	+	+	+	+
220	P9-25	GPIO3_21	+	+	+	+	+	+	+	+	+	+	+
+	+	+	15	14	13	12	11	10	9	8	7	6	5
Off set	Conn	Name	Pin Usage	Type		Reserve			S L E W	R X	P U - P D	P U / DE N	Mux Mode
+	+	+	+	0	0	0	0	0	0	0	0	0	0
222	P9-39	AIN0	+	+	+	+	+	+	+	+	+	+	+
224	P9-40	AIN1	+	+	+	+	+	+	+	+	+	+	+
226	P9-37	AIN2	+	+	+	+	+	+	+	+	+	+	+
228	P9-38	AIN3	+	+	+	+	+	+	+	+	+	+	+
230	P9-33	AIN4	+	+	+	+	+	+	+	+	+	+	+
232	P9-36	AIN5	+	+	+	+	+	+	+	+	+	+	+
234	P9-35	AIN6	+	+	+	+	+	+	+	+	+	+	+

5.8.3 Pin Usage Consideration

This section covers things to watch for when hooking up to certain pins on the expansion headers.

Boot PIN

There are 16 pins that control the boot mode of the processor that are exposed on the expansion headers. *Figure 63* below shows those signals as they appear on the BeagleBone Black.:

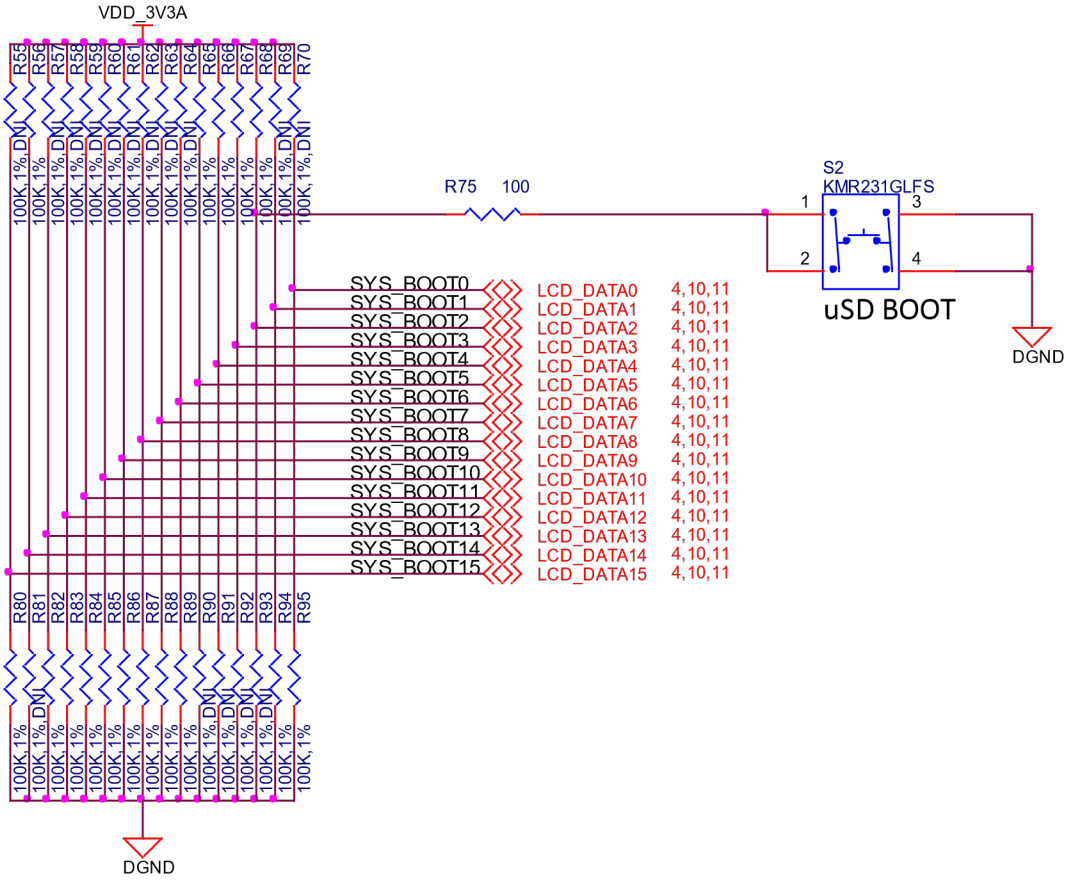


Figure 63. Expansion Boot Pins

Fig. 5.65: Boot signals

If you plan to use any of these signals, then on power up, these pins should not be driven. If you do, it can affect the boot mode of the processor and could keep the processor from booting or working correctly.

If you are designing a cape that is intended to be used as a boot source, such as a NAND board, then you should drive the pins to reconfigure the boot mode, but only at reset. After the reset phase, the signals should not be driven to allow them to be used for the other functions found on those pins. You will need to override the resistor values in order to change the settings. The DC pull-up requirement should be based on the AM3358 Vih min voltage of 2 volts and AM3358 maximum input leakage current of 18uA. Also take into account any other current leakage paths on these signals which could be caused by your specific cape design.

The DC pull-down requirement should be based on the AM3358 Vil max voltage of 0.8 volts and AM3358 maximum input leakage current of 18uA plus any other current leakage paths on these signals.

5.8.4 Expansion Connectors

A combination of male and female headers is used for access to the expansion headers on the main board. There are three possible mounting configurations for the expansion headers:

- *Single* no board stacking but can be used on the top of the stack.
- *Stacking-up* to four boards can be stacked on top of each other.
- *Stacking* with signal *stealing-up* to three boards can be stacked on top of each other, but certain boards will not pass on the signals they are using to prevent signal loading or use by other cards in the stack.

The following sections describe how the connectors are to be implemented and used for each of the different configurations.

Non-Stacking Headers-Single Cape

For non-stacking capes single configurations or where the cape can be the last board on the stack, the two 46 pin expansion headers use the same connectors. *Figure 64* is a picture of the connector. These are dual row 23 position 2.54mm x 2.54mm connectors.



Fig. 5.66: Single Expansion Connector

The connector is typically mounted on the bottom side of the board as shown in *Figure 65*. These are very common connectors and should be easily located. You can also use two single row 23 pin headers for each of the dual row headers.

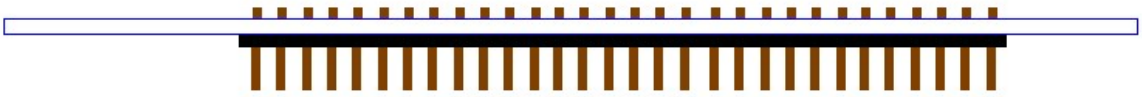


Fig. 5.67: Single Cape Expansion Connector

It is allowed to only populate the pins you need. As this is a non-stacking configuration, there is no need for all headers to be populated. This can also reduce the overall cost of the cape. This decision is up to the cape designer.

For convenience listed in *Table 19* are some possible choices for part numbers on this connector. They have varying pin lengths and some may be more suitable than others for your use. It should be noted, that the longer the pin and the further it is inserted into the BeagleBone Black connector, the harder it will be to remove due to the tension on 92 pins. This can be minimized by using shorter pins or removing those pins that are not used by your particular design. The first item in**Table 18** is on the edge and may not be the best solution. Overhang is the amount of the pin that goes past the contact point of the connector on the BeagleBone Black

Table 5.19: Single Cape Connectors

SUPPLIER	PARTNUMBER	LENGTH(in)	OVERHANG(in)
Major League	TSHC-123-D-03-145-G-LF	.145	.004
Major League	TSHC-123-D-03-240-G-LF	.240	.099
Major League	TSHC-123-D-03-255-G-LF	.255	.114

The G in the part number is a plating option. Other options may be used as well as long as the contact area is gold. Other possible sources are Sullins and Samtec for these connectors. You will need to ensure the depth

into the connector is sufficient

Main Expansion Headers-Stacking

For stacking configuration, the two 46 pin expansion headers use the same connectors. *Figure 66* is a picture of the connector. These are dual row 23 position 2.54mm x 2.54mm connectors.



Fig. 5.68: Expansion Connector

The connector is mounted on the top side of the board with longer tails to allow insertion into the BeagleBone Black. *Figure 67* is the connector configuration for the connector.



Fig. 5.69: Stacked Cape Expansion Connector

For convenience listed in *Table 18* are some possible choices for part numbers on this connector. They have varying pin lengths and some may be more suitable than others for your use. It should be noted, that the longer the pin and the further it is inserted into the BeagleBone Black connector, the harder it will be to remove due to the tension on 92 pins. This can be minimized by using shorter pins. There are most likely other suppliers out there that will work for this connector as well. If anyone finds other suppliers of compatible connectors that work, let us know and they will be added to this document. The first item in**Table 19** is on the edge and may not be the best solution. Overhang is the amount of the pin that goes past the contact point of the connector on the BeagleBone Black.

The third part listed in *Table 20* will have insertion force issues.

Table 5.20: Stacked Cape Connectors

SUPPLIER	PARTNUMBER	TAIL LENGTH(in)	OVERHANG(in)
Major League	SSHQ-123-D-06-G-LF	.190	0.049
Major League	SSHQ-123-D-08-G-LF	.390	0.249
Major League	SSHQ-123-D-10-G-LF	.560	0.419

There are also different plating options on each of the connectors above. Gold plating on the contacts is the minimum requirement. If you choose to use a different part number for plating or availability purposes, make sure you do not select the “LT” option.

Other possible sources are Sullins and Samtec but make sure you select one that has the correct mating depth.

StackedStealing

Figure 68 is the connector configuration for stackable capes that does not provide all of the signals upwards for use by other boards. This is useful if there is an expectation that other boards could interfere with the operation of your board by exposing those signals for expansion. This configuration consists of a combination of the stacking and nonstacking style connectors.

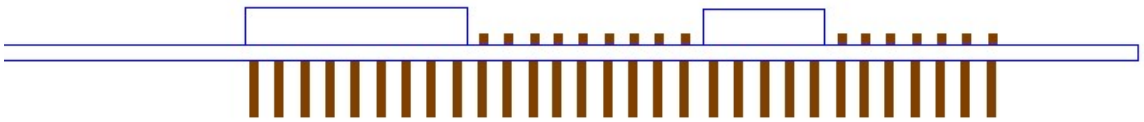


Fig. 5.70: Stacked w/Signal Stealing Expansion Connector

Retention Force

The length of the pins on the expansion header has a direct relationship to the amount of force that is used to remove a cape from the BeagleBone Black. The longer the pins extend into the connector the harder it is to remove. There is no rule that says that if longer pins are used, that the connector pins have to extend all the way into the mating connector on the BeagleBone Black, but this is controlled by the user and therefore is hard to control. We have also found that if you use gold pins, while more expensive, it makes for a smoother finish which reduces the friction.

This section will attempt to describe the tradeoffs and things to consider when selecting a connector and its pin length.

Figure 69 shows the key measurements used in calculating how much the pin extends past the contact point on the connector, what we call overhang.

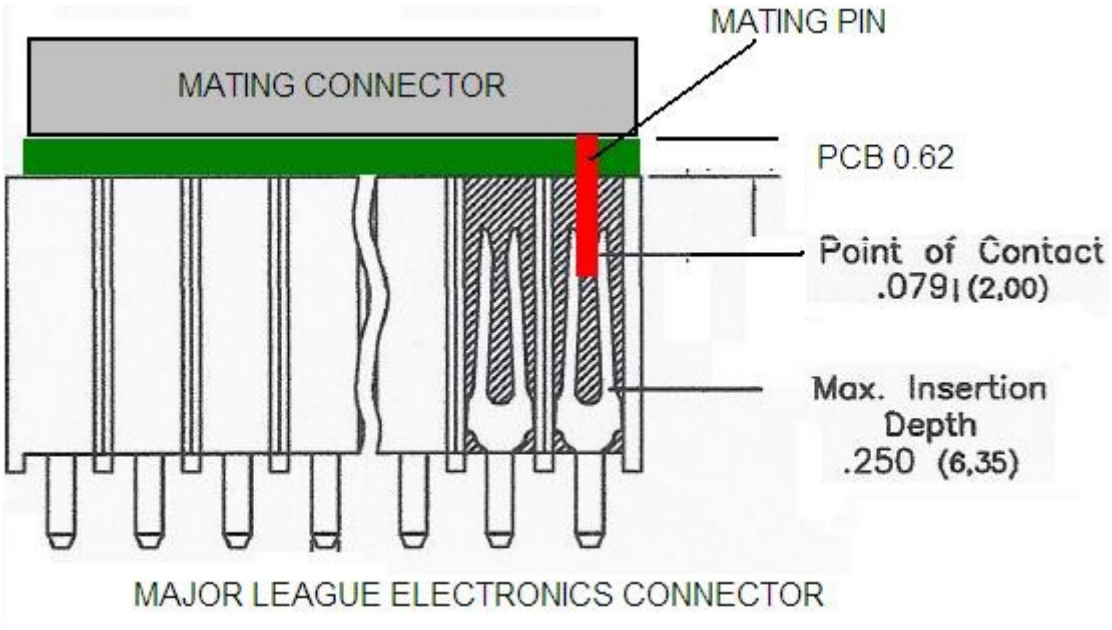


Fig. 5.71: Connector Pin Insertion Depth

To calculate the amount of the pin that extends past the Point of Contact, use the following formula:

Overhang=Total Pin Length- PCB thickness (.062) - contact point (.079)

The longer the pin extends past the contact point, the more force it will take to insert and remove the board. Removal is a greater issue than the insertion.

5.8.5 8.5 Signal Usage

Based on the pin muxing capabilities of the processor, each expansion pin can be configured for different functions. When in the stacking mode, it will be up to the user to ensure that any conflicts are resolved between multiple stacked cards. When stacked, the first card detected will be used to set the pin muxing of

each pin. This will prevent other modes from being supported on stacked cards and may result in them being inoperative.

In «section-7-1» of this document, the functions of the pins are defined as well as the pin muxing options. Refer to this section for more information on what each pin is. To simplify things, if you use the default name as the function for each pin and use those functions, it will simplify board design and reduce conflicts with other boards.

Interoperability is up to the board suppliers and the user. This specification does not specify a fixed function on any pin and any pin can be used to the full extent of the functionality of that pin as enabled by the processor.

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

5.8.6 8.6 Cape Power

This section describes the power rails for the capes and their usage.

Main Board Power

The *Table 1* describes the voltages from the main board that are available on the expansion connectors and their ratings. All voltages are supplied by connector**P9**. The current ratings listed are per pin.

Table 5.21: Expansion Voltages

Current	Name	P9	P9	Name	Current
250mA	VDD_3V3B	3	4	VDD_3V3B	250mA
1000mA	VDD_5V	5	6	VDD_5V	1000mA
250mA	SYS_5V	7	8	SYS_5V	250mA

The *VDD_3V3B* rail is supplied by the LDO on the BeagleBone Black and is the primary power rail for expansion boards. If the power requirement for the capes exceeds the current rating, then locally generated voltage rail can be used. It is recommended that this rail be used to power any buffers or level translators that may be used.

VDD_5V is the main power supply from the DC input jack. This voltage is not present when the board is powered via USB. The amount of current supplied by this rail is dependent upon the amount of current available. Based on the board design, this rail is limited to 1A per pin from the main board.

The *SYS_5V* rail is the main rail for the regulators on the main board. When powered from a DC supply or USB, this rail will be 5V. The available current from this rail depends on the current available from the USB and DC external supplies.

Power

A cape can have a jack or terminals to bring in whatever voltages may be needed by that board. Care should be taken not to let this voltage be fed back into any of the expansion header pins.

It is possible to provide 5V to the main board from an expansion board. By supplying a 5V signal into the *VDD_5V* rail, the main board can be supplied. This voltage must not exceed 5V. You should not supply any voltage into any other pin of the expansion connectors. Based on the board design, this rail is limited to 1A per pin to the BeagleBone Black.

There are several precautions that need to be taken when working with the expansion headers to prevent damage to the board.

1. *Do not apply any voltages to any I/O pins when the board is not powered on.* 2. *Do not drive any external signals into the I/O pins until after the VDD_3V3B rail is up.* 3. *Do not apply any voltages that are generated*

from external sources. 4. *If voltages are generated from the VDD_5V signal, those supplies must not become active until after the VDD_3V3B rail is up.* 5. *If you are applying signals from other boards into the expansion headers, make sure you power the board up after you power up the BeagleBone Black or make the connections after power is applied on both boards.*

Powering the processor via its I/O pins can cause damage to the processor.

5.8.7 8.7 Mechanical

This section provides the guidelines for the creation of expansion boards from a mechanical standpoint. Defined is a standard board size that is the same profile as the BeagleBone Black. It is expected that the majority of expansion boards created will be of standard size. It is possible to create boards of other sizes and in some cases this is required, as in the case of an LCD larger than the BeagleBone Black board.

Standard Cape Size

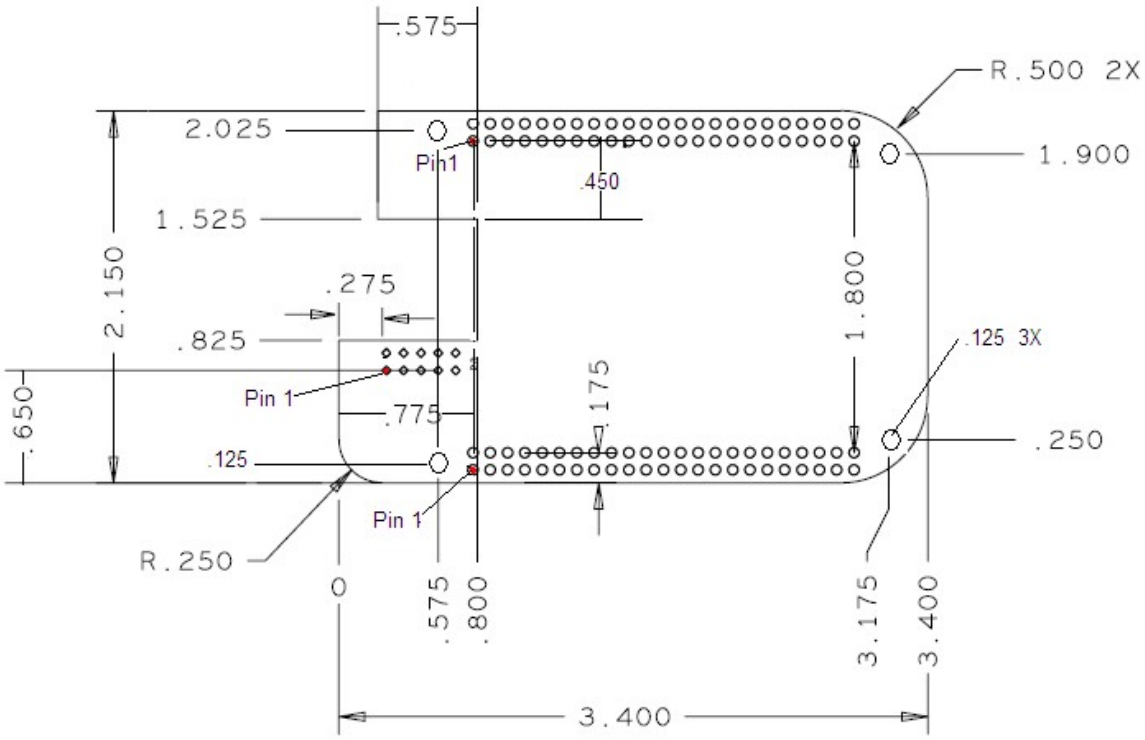


Fig. 5.72: Cape Board Dimensions

A slot is provided for the Ethernet connector to stick up higher than the cape when mounted. This also acts as a key function to ensure that the cape is oriented correctly. Space is also provided to allow access to the user LEDs and reset button on the main board.

Some people have inquired as to the difference in the radius of the corners of the BeagleBone Black and why they are different. This is a result of having the BeagleBone fit into the Altoids style tin.

It is not required that the cape be exactly like the BeagleBone Black board in this respect.

Extended Cape Size

Capes larger than the standard board size are also allowed. A good example would be an LCD panel. There is no practical limit to the sizes of these types of boards. The notch for the key is also not required, but it is up to the supplier of these boards to ensure that the BeagleBone Black is not plugged in incorrectly in such a

manner that damage would be caused to the BeagleBone Black or any other capes that may be installed. Any such damage will be the responsibility of the supplier of such a cape to repair.

As with all capes, the EEPROM is required and compliance with the power requirements must be adhered to.

Enclosures

There are numerous enclosures being created in all different sizes and styles. The mechanical design of these enclosures is not being defined by this specification.

The ability of these designs to handle all shapes and sizes of capes, especially when you consider up to four can be mounted with all sorts of interface connectors, it is difficult to define a standard enclosure that will handle all capes already made and those yet to be defined.

If cape designers want to work together and align with one enclosure and work around it that is certainly acceptable. But we will not pick winners and we will not do anything that impedes the openness of the platform and the ability of enclosure designers and cape designers to innovate and create new concepts.

5.9 BeagleBone Black Mechanical

5.9.1 Dimensions and Weight

Size: 3.5” x 2.15” (86.36mm x 53.34mm)

Max height: .187” (4.76mm)

PCB Layers: 6

PCB thickness: .062”

RoHS Compliant: Yes

Weight: 1.4 oz

5.9.2 Silkscreen and Component Locations

5.10 Pictures

5.11 Support Information

All support for this design is through the BeagleBoard.org community at: beagleboard@googlegroups.com or http://beagleboard.org/discuss

5.11.1 Hardware Design

Design documentation can be found on the eMMC of the board under the documents/hardware directory when connected using the USB cable. Provided there is:

- Schematic in PDF
- Schematic in OrCAD (Cadence Design Entry CIS 16.3)
- PCB Gerber
- PCB Layout (Allegro)
- Bill of Material
- System Reference Manual (This document).

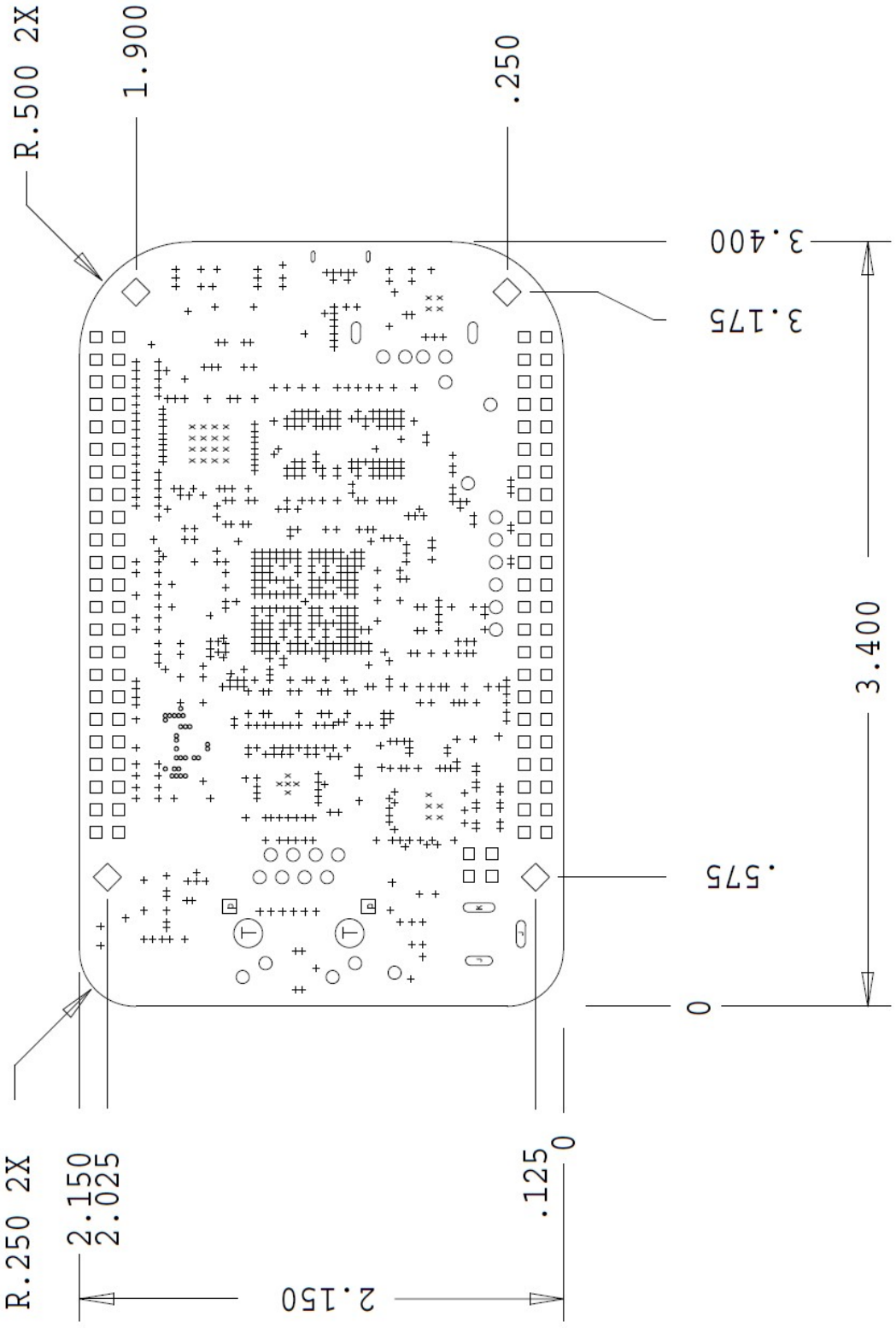


Fig. 5.73: Board Dimensions

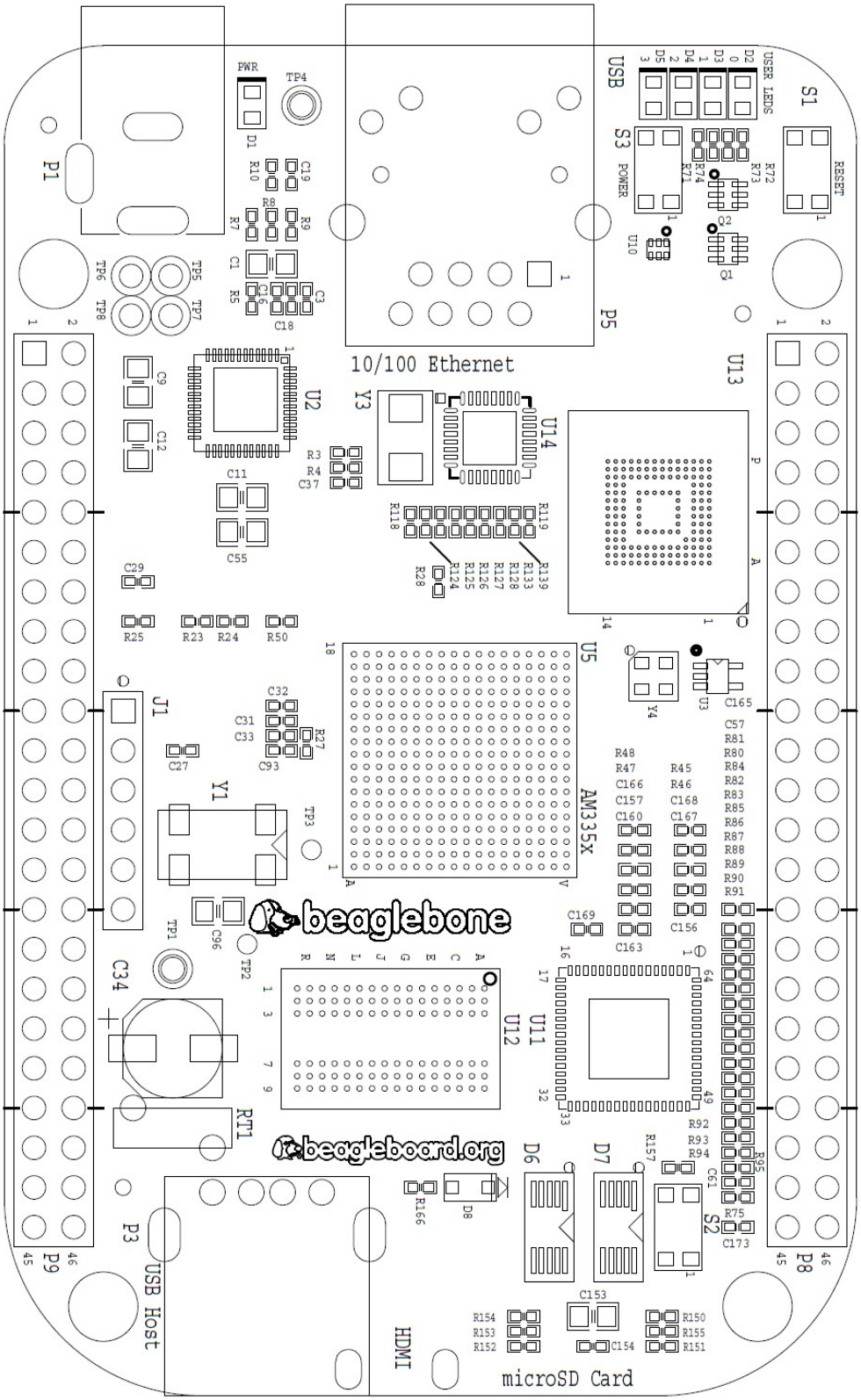


Fig. 5.74: Component Side Silkscreen

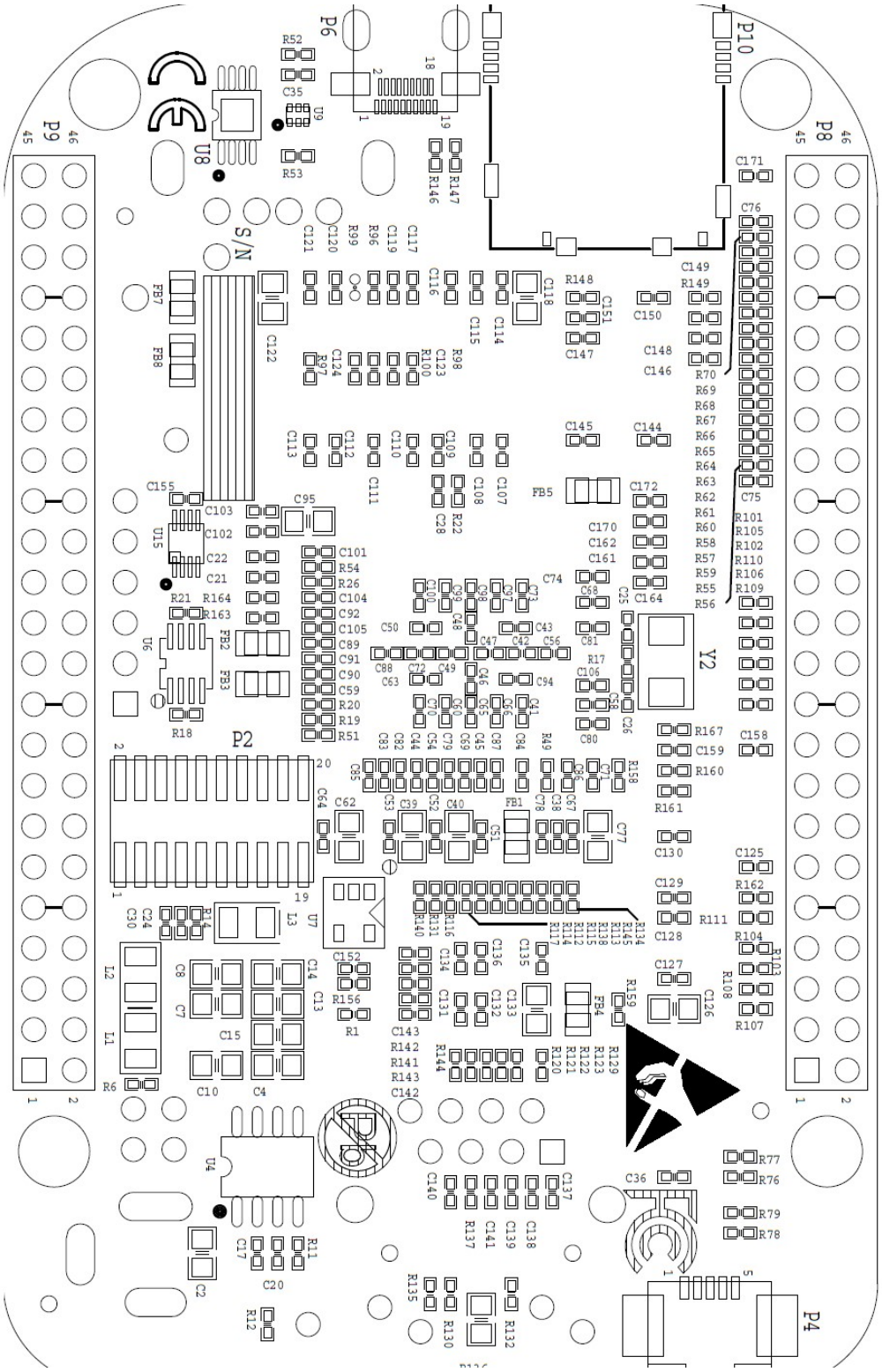


Fig. 5.75: Circuit Side Silkscreen

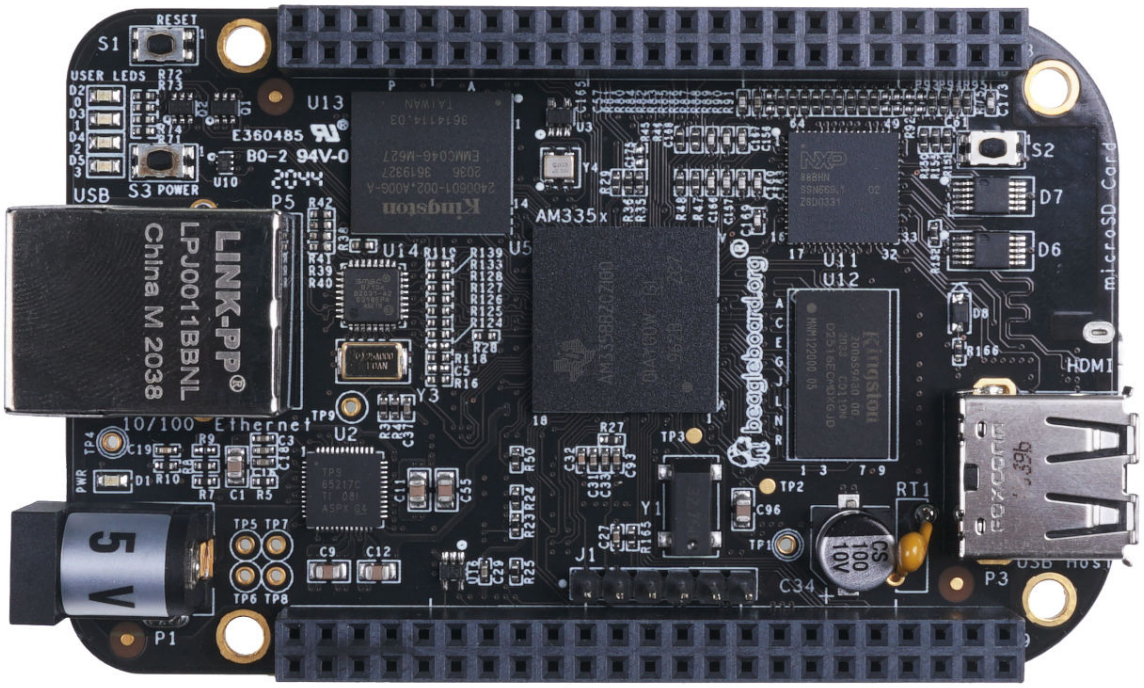


Fig. 5.76: Top Side

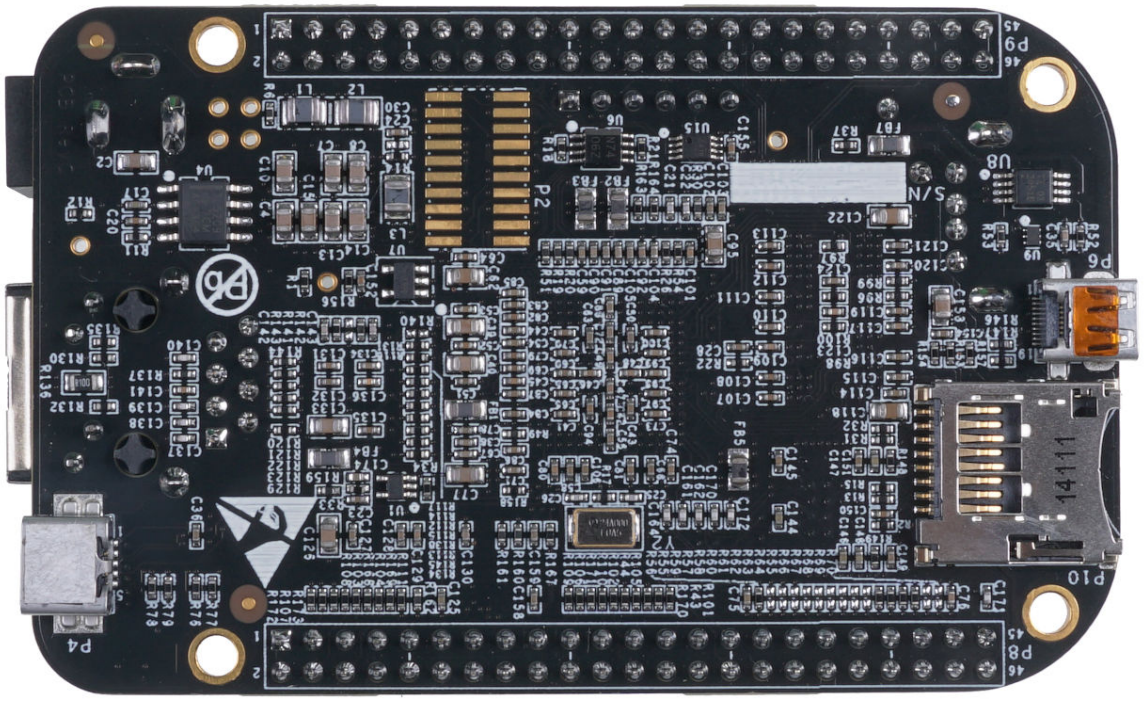


Fig. 5.77: Bottom Side

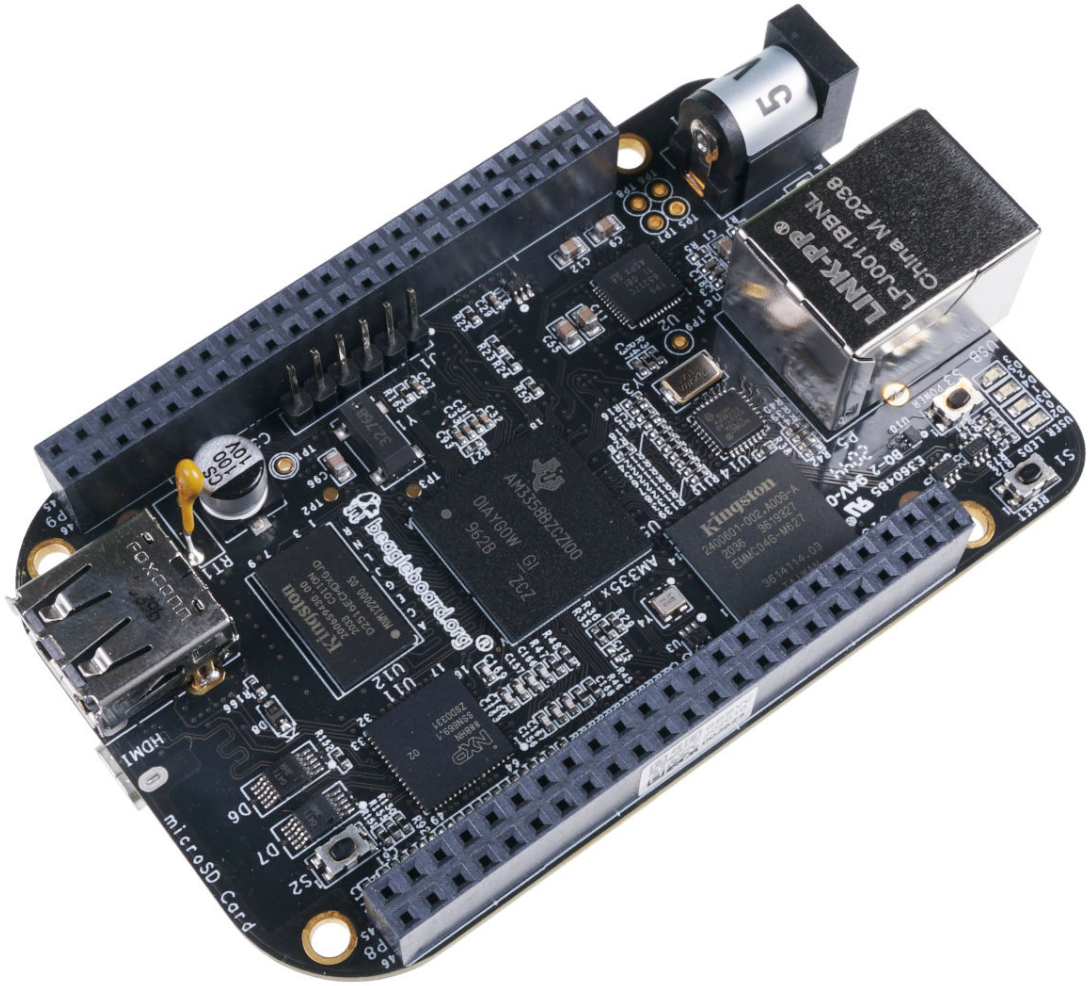


Fig. 5.78: 45 Degree Top

This directory is not always kept up to date in every SW release due to the frequency of changes of the SW. The best solution is to download the files from http://www.beagleboard.org/distros

We do not track SW revision of what is in the eMMC. SW is tracked separately from the HW due to the frequency of changes which would require massive relabeling of boards due to the frequent SW changes. You should always use the latest SW revision.

To see what SW revision is loaded into the eMMC follow the instructions at https://elinux.org/Beagleboard:Updating_The_Software#Checking_The_Angstrom_Image_Version

5.11.2 Software Updates

It is a good idea to always use the latest software. Instructions for how to update your software to the latest version can be found at:

http://elinux.org/BeagleBoneBlack#Updating_the_eMMC_Software

5.11.3 RMA Support

If you feel your board is defective or has issues, request an RMA by filling out the form at http://beagleboard.org/support/rma . You will need the serial number and revision of the board. The serial numbers and revisions keep moving. Different boards can have different locations depending on when they were made. The following figures show the three locations of the serial and revision number.

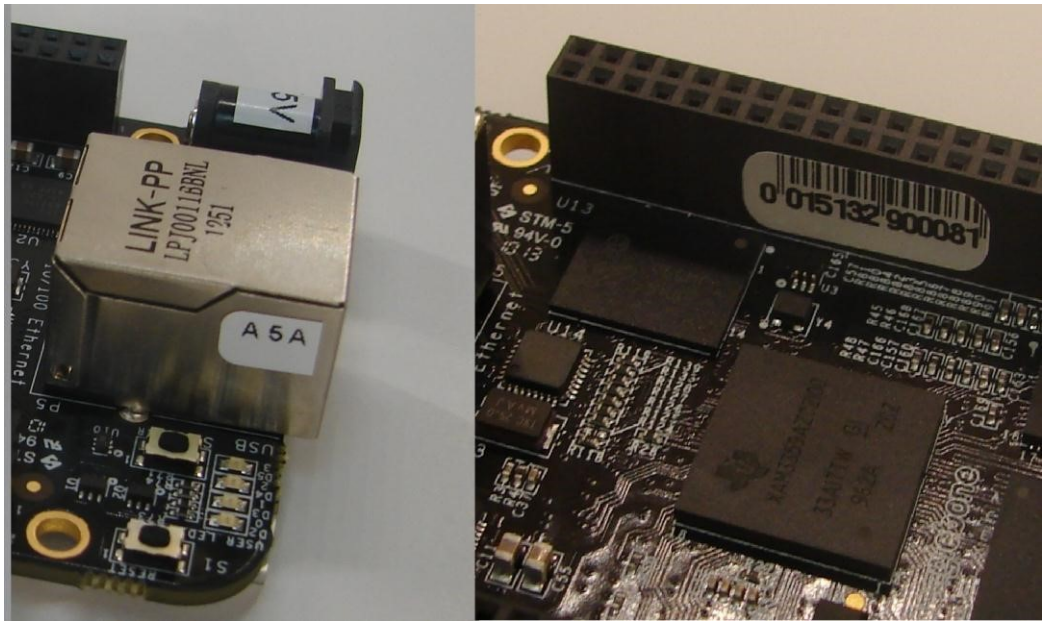


Fig. 5.79: Initial Serial Number and Revision Locations

5.11.4 Trouble Shooting HDMI Issues

Many people are having issues with getting HDMI to work on their TV/Display. Unfortunately, we do not have the resources to buy all the TVs and Monitors on the market today nor go to eBay and buy all of the TVs and monitors made over the last five years to thoroughly test each and every one. We are depending on community members to help us get these tested and information provided on how to get them to work.

One would think that if it worked on a lot of different TVs and monitors it would work on most if not all of them, assuming they meet the specification. However, there are other issues that could also result in these various TVs and monitors not working. The intent is that this page will be useful in navigating some of these issues. As others also find solutions, as long as we know about them, they will be added here as well. For access to

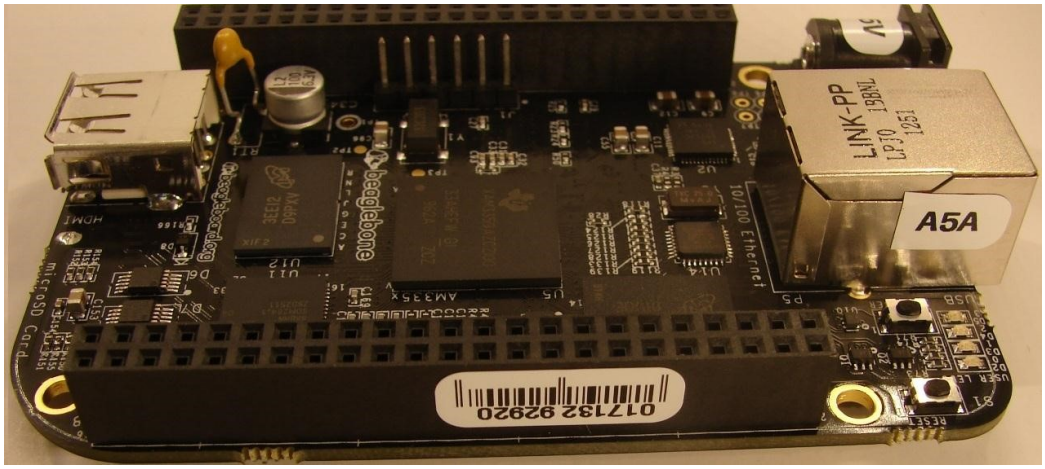


Fig. 5.80: Second Phase Serial Number and Revision Location

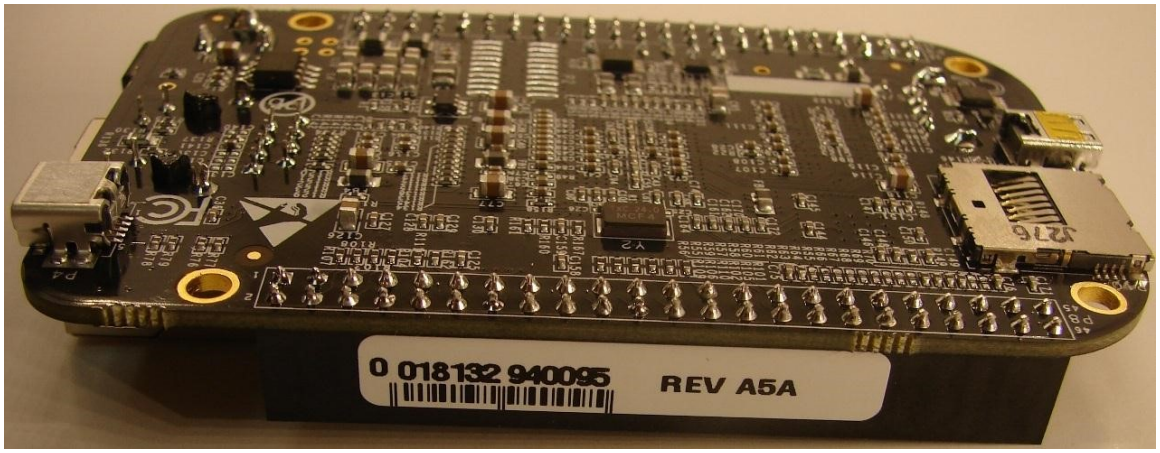


Fig. 5.81: Third Phase Serial Number and Revision Location

the most up to date troubleshooting capabilities, go to the support wiki at http://www.elinux.org/Beagleboard:BeagleBoneBlack_HDMI

The early release of the Software had some issues in the HDMI driver. Be sure and use the latest SW to take advantage of the improvements.

http://www.elinux.org/Beagleboard:BeagleBoneBlack#Software_Resources

EDID

EDID is the way the board requests information from the display and determines all the resolutions that it can support. The driver on the board will then look at these timings and find the highest resolution that is compatible with the board and uses that resolution for the display. For more information on EDID, you can take a look at http://en.wikipedia.org/wiki/Extended_display_identification_data

If the board is not able to read the EDID, for whatever reason, it does not have this information. A few possible reasons for this are:

- Bad cable
- Cable not plugged in all the way on both ends
- Display not powered on. (It should still work powered off, but some displays do not).

DISPLAY SOURCE SELECTION

One easy thing to overlook is that you need to select the display source that matches the port you are using on the TV. Some displays may auto select, so you may need to disconnect the other inputs until you are sure the display works with the board.

OUT OF SEQUENCE

Sometimes the display and the board can get confused. One way to prevent this is after everything is cabled up and running, you can power cycle the display, with the board still running. You can also try resetting the board and let it reboot to resync with the TV.

OVERSCAN

Some displays use what is called overscan. This can be seen in TVs and not so much on Monitors. It causes the image to be missing on the edges, such that you cannot see them displayed. Some higher end displays allow you to disable overscan.

Most TVs have a mode that allows you to adjust the image. These are options like Normal, Wide, Zoom, or Fit. Normal seems to be the best option as it does not chop of the edges. The other ones will crop of the edges.

Taking a Nap

In some cases the board can come up in a power down/screen save mode. No display will be present. This is due to the board believing that it is asleep. To come out of this, you will need to hit the keyboard or move the mouse.

Once working, the board will time out and go back to sleep again. This can cause the display to go into a power down mode as well. You may need to turn the display back on again. Sometimes, it may take a minute or so for the display to catch up and show the image.

AUDIO

Audio will only work on TV resolutions. This is due to the way the specification was written. Some displays have built in speakers and others require external. Make sure you have a TV resolution and speakers are connected if they are not built in. The SW should default to a TV resolution giving audio support. The HDMI driver should default to the highest audio supported resolution.

Getting Help

If you need some up to date troubleshooting techniques, we have a Wiki set up at http://elinux.org/Beagleboard:BeagleBoneBlack_HDMI

Chapter 6

BeagleBone Blue

To optimize BeagleBone for education, BeagleBone Blue was created that integrates many components for robotics and machine control, including connectors for off-the-shelf robotic components. For education, this means you can quickly start talking about topics such as programming and control theory, without needing to spend so much time on electronics. The goal is to still be very hackable for learning electronics as well, including being fully open hardware.

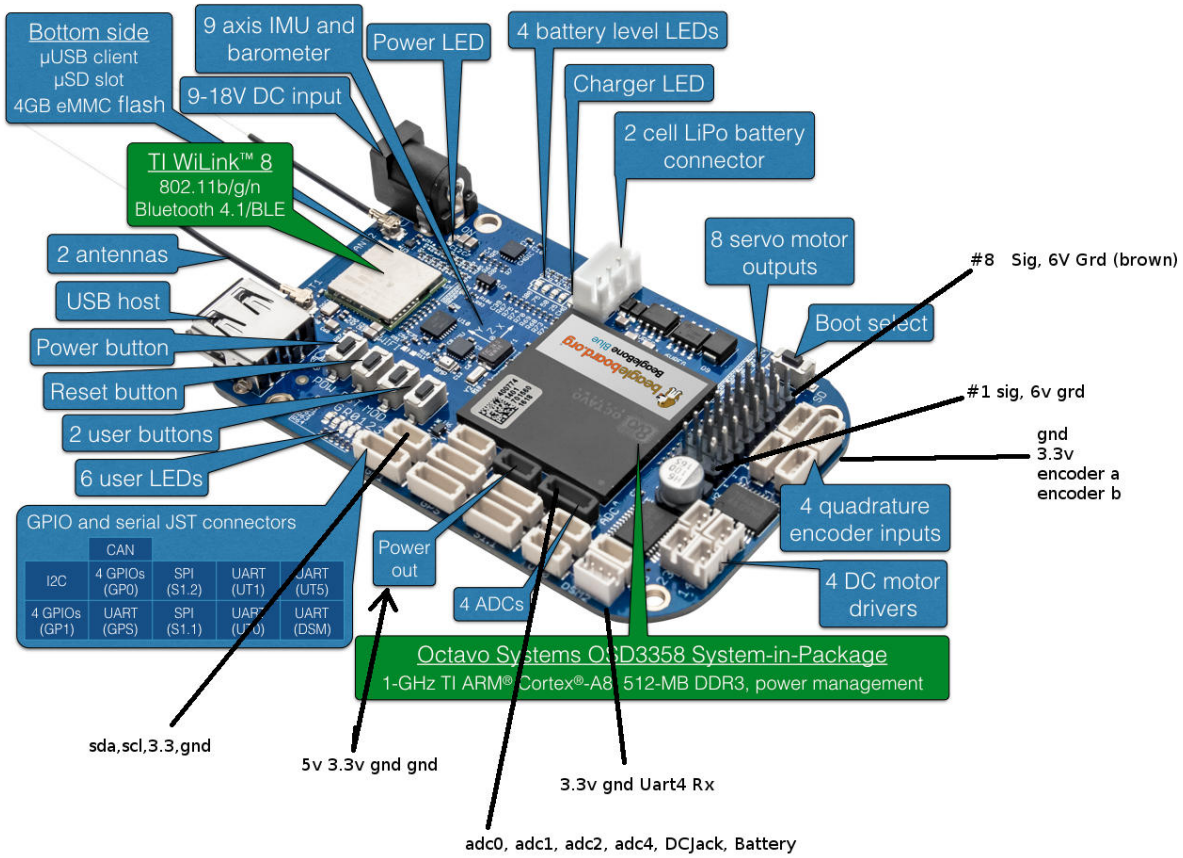
BeagleBone Blue's legacy is primarily from contributions to BeagleBone Black robotics by UCSD Flow Control and Coordinated Robotics Lab, Strawson Design, Octavo Systems, WowWee, National Instruments LabVIEW and of course the BeagleBoard.org Foundation.

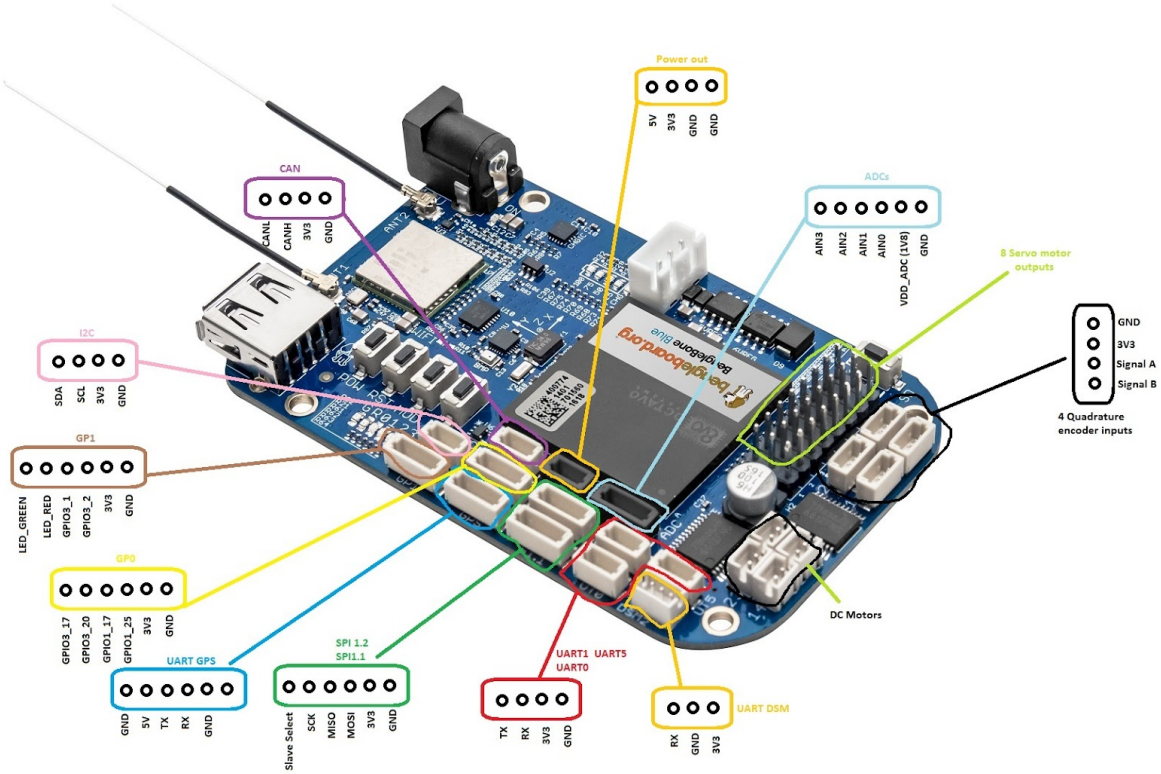


License Terms

- This documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page

6.1 BeagleBone Blue Pinouts





- Connector pinout details from schematic(s)
- Pin Table with some Blue : Black corelation.

6.1.1 UT1

UART (/dev/ttyS1)

```
config-pin P9.24 uart
config-pin P9.26 uart
```

6.1.2 GPS

UART (/dev/ttyS2)

```
config-pin P9.21 uart
config-pin P9.22 uart
```

6.2 SSH

If you don't have ssh installed, install it. (google is your friend) Then *ssh debian@192.168.7.2* The board will tell you what the password is, on my it was *temppwd*.

To change your password use the command password it will ask you what your current password is, then ask for the replacement. Then it will say it was too simple and you have to do it again. Normal stuff.

If you want to insist on using your simple password, try this.

```
sudo -s
(become superuser/root)
enter your password
password debian
 (put your simple password in)
exit
(exit from superuser/root)
```

When you are running as root, password is more compliant and will accept simple password

6.3 WiFi Setup

On my network, I'm set up as ip 192.168.1.*. To turn your wifi on, do the following.

```
sudo -s
(become superuser/root)
cd /etc/network/
ifconfig
(Note the wifi inet address, if it is already set, you are done!)
connmanctl
tether wifi off
enable wifi
scan wifi
services
(at this point you should see your network appear along with other stuff, in␣
↪my case it was "AR Crystal wifi_f45eab2f1ee1_6372797774616c_managed_psk")
nano interfaces
(or whatever editor you like)
remove the comment # from the wifi lines so it now appears like
##connman: WiFi
#
connmanctl
connmanctl> tether wifi off
connmanctl> enable wifi
connmanctl> scan wifi
connmanctl> services
connmanctl> agent on
connmanctl> connect wifi_f45eab2f1ee1_6372797774616c_managed_psk
connmanctl> quit
exit
note that you will need to fill in your own network data
```

6.4 IP settings

You will usually want to have a fixed ip if you are doing robotics, so you have a standard ip to connect to. If you are already connected in dhcp you can borrow some of the settings from that to use in your new configurations.

```
route
```

make a note of the default one, (in the example below 192.168.1.1)

```
cat /etc/resolv.conf
```

make a note of the nameserver, (in the example below 8.8.8.8)

In my case I wanted 192.168.1.7 to do this,

```
sudo -s
connmanctl config wifi_f45eab2f1ee1_6372797774616c_managed_psk --ipv4 manual␣
↪192.168.1.7 255.255.255.0 192.168.1.1 --nameservers 8.8.8.8
exit
```

the –ipv4 says to use ipv4 settings (as opposed to ipv6), the manual means we are setting the values. 192.168.1.7 is the ip address we want. (use your own of course). 255.255.255.0 is the network mask 192.168.1.1 is the route to the internet. (You're might be different, but this is common). –nameservers 8.8.8.8 says where to find the ip address for a given domain name. the 8.8.8.8 says use's googles

6.5 Flashing Firmware

6.5.1 Overview

Most Beaglebones have a built in 4 GB SD card known as a eMMC (embedded MMC). When the boards are made the eMMC is "flashed" with some version of the BeagleBone OS that is usually outdated. Therefore, whenever receiving the BeagleBone it is recommend that you update the eMMC with the last version of the BeagleBone OS or a specific version of it if someone tells you otherwise.

6.5.2 Required Items

1. Micro sd card. 4 GB minimum
2. Micro sd card reader or a built in sd card reader for your PC
3. BeagleBone image you want to flash.
4. Etcher utility for your PC's OS.

6.5.3 Steps Overview

1. Burn the image you want to flash onto a micro sd card using the Etcher utility.
2. Boot the BeagleBone like normal and place the micro sd card into the board once booted.
3. Update the micro sd card image so its in "flashing" mode.
4. Insert micro sd card, remove power from the BeagleBone, hold sd card select button, power up board
5. Let the board flash

6.5.4 Windows PCs

1. Download the BeagleBone OS image you want to use.
2. Use the Etcher utility to burn the BeagleBone image you want to use on the micro sd card you plan on using.
3. Make sure you don't have the micro sd card plugged into your board.
4. Boot the board
5. Connect to the board via serial or ssh so that your on the command prompt.
6. Plug the micro sd card into the board.
7. Type dmesg in the terminal window
8. The last line from the output should say something like (the numbering may differ slightly):
    - "[2805.442940] mmcblk0: p1"

9. You want to take the above and combine it together by removing the : and space. For the above example it will change to “mmcblk0p1”
10. In the terminal window enter the following commands:

```
mkdir sd_tmp
sudo mount /dev/mmcblk0p1 sd_tmp
sudo su
echo ”cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh” >> sd_
↪tmp/boot/uEnv.txt
exit
sudo umount sd_tmp
```

11. Now power off your board
12. Hold the update button labeled SD (the one by itself) to boot off the sdcard.
13. Restart (RST button) or power up (while still pushing SD button).

Flashing can take some minutes. ## Linux/Mac PCs 1. Download the BeagleBone OS image you want to use. 1. Use the Etcher utility to burn the BeagleBone image you want to use on the micro sd card you plan on using. 1. On the SD card edit the file `/boot/uEnv.txt` in order for the SD card contents to be flashed onto the firmware eMMC. (Otherwise the BBBL will do no more than boot the SD image.) Uncomment the line containing `init-eMMC-flasher-v<number>.sh` either manually or using these commands substituting `X` with what your SD card shows in `/dev/:* sudo mount /dev/emmcblkXp1 /mnt * cd /mnt * sed -i 's_#[ ]*\(cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v[0-9]\+.*\.sh\)_\1_' boot/uEnv.txt`

1. Eject the sdcard from your computer.
2. Put it into your BeagleBoneBlue.
3. If your board was already powered on then power it off
4. Hold the update button labeled SD (the one by itself) to boot off the sdcard.
5. Restart (RST button) or power up (while still pushing SD button).

Flashing can take some minutes.

How to tell if it is flashing?

At first a blue heartbeat is shown indicating the image is booted. On flash procedure start, the blue user LEDs light up in a “larson scanner” or “cylon” pattern (back and forth).

When finished, either all blue LEDs are on or the board is already switched off.

If the LEDs are on for a long time then it may indicate failure e.g. wrong image. Can be verified if boot fails, i.e. board turns off again shortly after power up.

6.6 Play with the code

The board has some code built in to the system that can allow you to try out the various options. They all start with rc

```
rc_balance              rc_dsm_passthrough    rc_test_encoders
rc_battery_monitor      rc_kill               rc_test_filters
rc_benchmark_algebra    rc_spi_loopback       rc_test_imu
rc_bind_dsm             rc_startup_routine    rc_test_motors
rc_blink                rc_test_adc           rc_test_polynomial
rc_calibrate_dsm        rc_test_algebra       rc_test_servos
rc_calibrate_escs       rc_test_barometer     rc_test_time
rc_calibrate_gyro       rc_test_buttons       rc_test_vector
```

(continues on next page)

(continued from previous page)

```
rc_calibrate_mag      rc_test_cape          rc_uart_loopback
rc_check_battery      rc_test_dmp           rc_version
rc_check_model        rc_test_drivers
rc_cpu_freq           rc_test_dsm
```

Try them out to try out the various functions of the board. The source code for these tests and demos is at Robotics cape installer at github

6.7 BeagleBone Blue tests

6.7.1 ADC

- Grove Rotary Angle Sensor See output on adc_1 source

```
rc_test_adc
```

6.7.2 GP0

- Grove single GPIO output modules like LED Socket Kit

```
cd /sys/class/gpio;echo 49 >export;cd gpio49;echo out >direction;while sleep␣
↪1;do echo 0 >value;sleep 1;echo 1 >value;done
```

- Grove single GPIO input modules like IR Distance Interrupter or Touch Sensor

```
cd /sys/class/gpio;echo 49 >export;cd gpio49;echo in >direction;watch -n0␣
↪cat value
```

6.7.3 GP1

- Grove single GPIO output modules like LED Socket Kit

```
cd /sys/class/gpio;echo 97 >export;cd gpio97;echo out >direction;while sleep␣
↪1;do echo 0 >value;sleep 1;echo 1 >value;done
```

- Grove single GPIO input modules like IR Distance Interrupter or Touch Sensor

```
cd /sys/class/gpio;echo 97 >export;cd gpio97;echo in >direction;watch -n0␣
↪cat value
```

6.7.4 UT1

- Grove GPS

```
tio /dev/ttyO1 -b 9600
```

6.7.5 GPS

- GPS Receiver - EM-506

```
tio /dev/ttyO2 -b 4800
```

6.7.6 I2C

Grove I2C modules

The Linux kernel source has some basic IIO SYSFS interface documentation which might provide a little help for understanding reading these entries. The ELC2017 conference also had an IIO presentation.

- Digital Light Sensor

```
cd /sys/bus/i2c/devices/i2c-1;echo tsl2561 0x29 >new_device;watch -n0 cat 1-
↪0029/iio\:device0/in_illuminance0_input
```

- Temperature & Humidity Sensor

```
cd /sys/bus/i2c/devices/i2c-1;echo th02 0x40 >new_device;watch -n0 cat 1-
↪0040/iio\:device0/in_temp_raw
```

6.7.7 Motors

```
rc_test_motors
```

6.8 Accessories

Todo: We are going to work on a unified accessories page for all the boards and it should replace this.

6.8.1 Chassis and kits

- EduMIP
- Pololu Romi Chassis with geared motors
  - Wheel encoders
  - Chassis - Black
- Sprout Runt Rover

6.8.2 Cases

6.8.3 Cable assemblies and sub-assemblies

Beware; purchased pre-made connector assembly wire colors may not reflect true pin designations. These assemblies are readily available from Digi-Key, SparkFun, Hobby King, Pololu and Cables and Connectors.

JST Connector Bundle

Renaissance Robotics JST Jumper Bundle

Four of the 2-pin JST ZH (1.5mm pitch) connectors, with 150mm 28AWG wires, for motors,

Eight of the 4-pin JST SH (1mm pitch) connectors, with 150mm 28AWG wires, for encoders, UART, I2C, CAN, PWR, and

Four of the 6-pin JST SH (1mm pitch) connectors, with 150mm 28AWG wires, for SPI, GPS, GPIO, ADC.

Renaissance Robotics JST Jumper Bundle

Conrad BeagleBoard Kabel BB-Blue-Kabelset

10x 4-Pin JST-SH
6x 6-Pin JST-SH
4x 2-Pin JST-ZH
1x 3-Pin JST-ZH
BeagleBoard Kabel BB-Blue-Kabelset (Conrad.de)

6.8.4 UART, I2C, CAN, Quadrature encoders, PWR

4-wire JST-SH (1mm pitch)

- 4-wire Grove cable (Digi-Key)
- Hobby King SKU 258000190-0
- SparkFun PN 10359
- Cables and Connectors 4” ribbon PN #4904
- Digi-Key wires
- Digi-Key housings

6.8.5 SPI, GPIO, ADC

6-wire JST-SH (1mm pitch)

- Hobby King SKU 258000192-0
- SparkFun PN 10361
- Cables and Connectors 50cm length PN #49406
- Digi-Key wires
- Digi-Key housings
- 6-wire Grove cable (4 populated) (Digi-Key)

6.8.6 Motors

2-wire JST-ZH (1.5mm pitch)

- Digi-Key wires
- Digi-Key receptacle

6.8.7 DSM

3-wire JST-ZH (1.5mm pitch)

- Pololu PN# 2411

microUSB

standard

Batteries

2S1P LiPo with 3-wire JST-XH (2.5mm pitch) charge connection

- Hobby King 1000mAh 2S 20C LiPo
- Hobby King 1600mAh 2S 20C LiPo

6.8.8 Power supplies

12V with 5.5mm/2.1mm center positive

- Jameco: supply and power cord
- Hobby King 12V 3A supply

6.8.9 Motors

Servo motors

6V DC

- Parallax Inc. 900-00005 Standard Servo
- Hobby King SKU HD-1900A
- TowerPro SG92R-7

DC motors

6V, typically geared

- SparkFun Hobby Gearmotor - 200 RPM (Pair)
- SparkFun Hobby Motor - Gear

6.8.10 Radio remotes

- Hobby King OrangeRX satellite receiver
- Spektrum DSM2 Remote Receiver

6.8.11 GPS

- Sparkfun GPS Receiver - EM-506 (48 Channel)
- Adafruit Ultimate GPS breakout
- Ublox Neo-M8N GPS with Compass
- SeeedStudio Grove - GPS

6.8.12 Replacement antennas

- LSR PIFA
- LSR Dipole: antenna and cable
- Anaren U.FL 2.4GHz 6MM Antenna
- TI approved antennas

6.8.13 USB devices

USB cameras

- Logitech C270
- Logitech C920

6.8.14 SPI devices

SPI TFT displays

- Adafruit 2.4” LCD breakout

6.8.15 I2C devices

- See *One Liner Module Tests*
- See *Using I2C with Linux drivers*

6.8.16 UART devices

Computer serial adapters

- Sparkfun FTDI Cable 5V VCC-3.3V I/O
- Adafruit FTDI Serial TTL-232 USB Cable

6.8.17 Bluetooth devices

- WowWee Groove Cube Speaker

6.9 Frequently Asked Questions (FAQs)

6.9.1 Are there any books to help me get started?

The book BeagleBone Robotic Projects, Second Edition specifically covers how to get started building robots with BeagleBone Blue.

For more general books on BeagleBone, Linux and other related topics, see https://beagleboard.org/books.

6.9.2 What system firmware should I use for starting to explore my BeagleBone Blue?

Download the latest ‘IoT’ image from https://www.beagleboard.org/distros. As of this writing, that image is https://debian.beagleboard.org/images/bone-debian-9.5-iot-armhf-2018-10-07-4gb.img.xz.

Use http://etcher.io for writing that image to a 4GB or larger microSD card.

Power-up your BeagleBone Blue with the newly created microSD card to run this firmware image.

6.9.3 What is the name of the access point SSID and password default on BeagleBone Blue?

SSID: BeagleBone-XXXX where XXXX is based upon the board's assigned unique hardware address
Password: BeagleBone

6.9.4 I've connected to BeagleBone Blue's access point. How do I get logged into the board?

Browse to http://192.168.8.1:3000 to open the Visual Studio Code IDE and get access to the Linux command prompt.

If you've connected via USB instead, the address will be either http://192.168.6.2:3000 or http://192.168.7.2:3000, depending on the USB networking drivers provided by your operating system.

6.9.5 How do I connect BeagleBone Blue to my own WiFi network?

From the bash command prompt in Linux:

```
sudo -s (become superuser/root)

connmanctl
   connmanctl> tether wifi off (not really necessary on latest images)
   connmanctl> enable wifi (not really necessary)
   connmanctl> scan wifi
   connmanctl> services (at this point you should see your network
   appear along with other stuff, in my case it was "AR Crystal wifi_
↪f45eab2f1ee1_6372797774616c_managed_psk")
   connmanctl> agent on
   connmanctl> connect wifi_f45eab2f1ee1_6372797774616c_managed_psk
   connmanctl> quit
```

6.9.6 Where can I find examples and APIs for programming BeagleBone Blue?

Programming in C: http://www.strawsondesign.com/#!manual-install

Programming in Python: https://github.com/mcdeoliveira/rcpy

Programming in Simulink: https://www.mathworks.com/hardware-support/beaglebone-blue.html

6.9.7 My BeagleBone Blue fails to run successful tests

You've tried to run rc_test_drivers to ensure your board is working for DOA warranty tests, but it errors. You should first look to fixing your bootloader as described http://strawsondesign.com/docs/librobotcontrol/installation.html#installation_s5

6.9.8 I'm running an image off of a microSD card. How do I write it to the on-board eMMC flash?

Refer to the "Flashing Firmware" page: https://git.beagleboard.org/beagleboard/beaglebone-blue/-/wikis/Flashing-firmware

Meanwhile, as root, run the /opt/scripts/tools/eMMC/bbb-eMMC-flasher-eewiki-ext4.sh script which will create a copy of the system in your microSD to a new single ext4 partition on the on-board eMMC.

6.9.9 I've written the latest image to a uSD card, but some features aren't working. How do I make it run properly?

It is possible you are running an old bootloader off of the eMMC. While power is completely off, hold the SD button (near the servo headers) while applying power. You can release the button as soon the power LED comes on. This will make sure the bootloader is loaded from microSD and not eMMC.

Verify the running image using version.sh via:

```
sudo /opt/scripts/tools/version.sh
```

The version.sh output will tell you which version of bootloader is on the eMMC or microSD. Future versions of version.sh might further inform you if the SD button was properly asserted on power-up.

One you've booted the latest image, you can update the bootloader on the eMMC using /opt/scripts/tools/developers/update_bootloader.sh. Better yet, read the above FAQ on flashing firmware.

6.9.10 I've got my on-board eMMC flash configured in a nice way. How do I copy that to other BeagleBone Blue boards?

As root, run the /opt/scripts/tools/eMMC/beaglebone-black-make-microSD-flasher-from-eMMC.sh script with a blank 4GB or larger microSD card installed and wait for the script to complete execution.

Remove the microSD card.

Boot your other BeagleBone Blue boards off of this newly updated microSD card and wait for the flashing process to complete. You'll know it successfully started when you see the "larson scanner" running on the LEDs. You'll know it successfully completed when it shuts off the board.

Remove the microSD card.

Reboot your newly flashed board.

6.9.11 I have some low-latency I/O tasks. How do I get started programming the BeagleBone PRUs?

There is a "Hello, World" app at https://gist.github.com/jadonk/2ecf864e1b3f250bad82c0eae12b7b64 that will get you blinking the USRx LEDS.

The libroboticscape software provides examples that are pre-built and included in the BeagleBone Blue software images for running the servo/ESC outputs and fourth quadrature encoder input. You can use those firmware images as a basis for building your own: https://github.com/StrawsonDesign/Robotics_Cape_Installer/tree/master/pru_firmware

You can find some more at https://beagleboard.org/pru

6.9.12 Are there available mechanical models?

A community contributed model is available at https://grabcad.com/library/beaglebone-blue-1

6.9.13 What is the operating temperature range?

'0..70' due to processor, else '-20..70'

6.9.14 What is the DC motor drive strength?

This is dictated by the 2 cell LiPo battery input, the TB6612FNG motor drivers and the JST-ZH connectors

- Voltage: 6V-8.4V (typical)
- Current: 1A (maximum for connectors) / 1.2A (maximum average from drivers) / 3.2A (peak from drivers) per channel

Chapter 7

BeagleBone (all)

BeagleBone boards are intended to be bare-bones, with a balance of features to enable rapid prototyping and provide a solid reference for building end products.

The most popular design is *BeagleBone Black*, a staple reference for an open hardware embedded Linux single board computer.

BeagleBone AI-64 is our most powerful design with tremendous machine learning inference performance, 64-bit processing and a mixture of microcontrollers for various types of highly-reliable and low-latency control.

For simplicity of developing small, mobile robotics, check out *BeagleBone Blue*, a highly integrated board with motor drivers, battery support, altimeter, gyroscope, accelerometer, and much more to get started developing quickly.

The System Reference Manual for each BeagleBone board is below. Older boards are supported with links to their latest PDF-formatted System Reference Manual and the latest boards are included both here and in the downloadable beagleboard-docs.pdf linked on the bottom-left of your screen.

All boards received without RMA approval will not be worked on.

- BeagleBone (original)
- *BeagleBone Black*
- *BeagleBone Blue*
- *BeagleBone AI*
- *BeagleBone AI-64*

Chapter 8

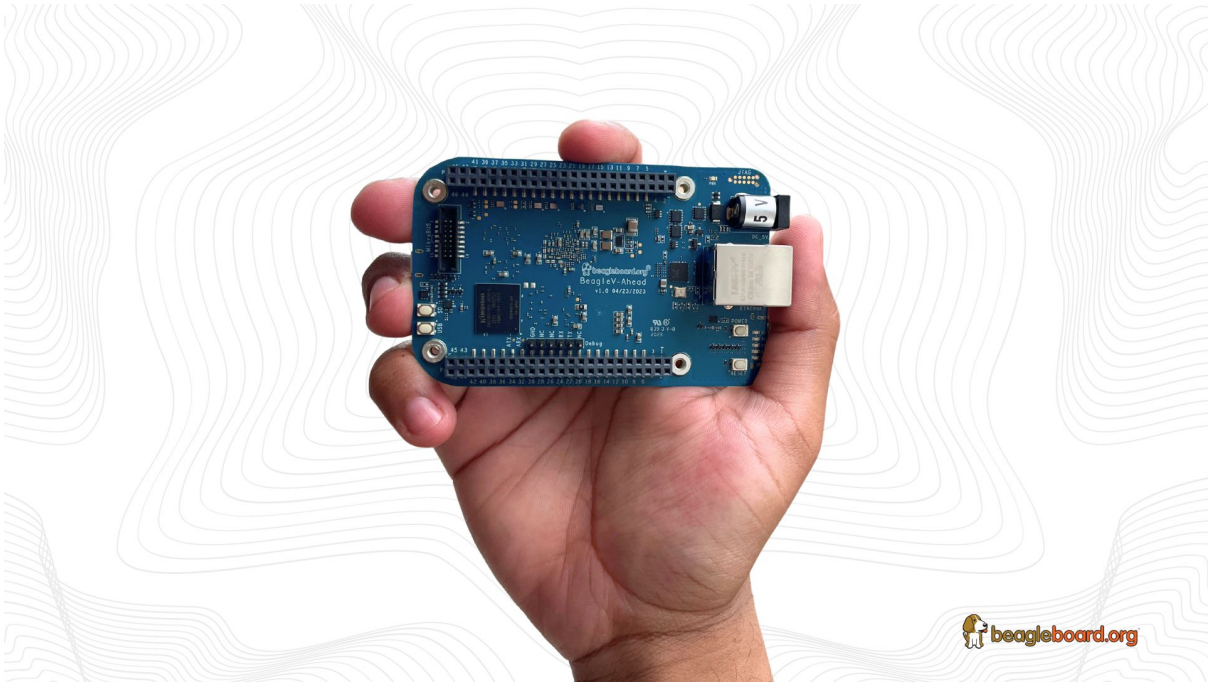
BeagleV-Ahead

BeagleV-Ahead is a high-performance open-source RISC-V single board computer (SBC) built around the Alibaba TH1520 SoC. It has the same P8 & P9 cape header pins as BeagleBone Black allowing you to stack your favourite BeagleBone cape on top to expand it's capability. Featuring a powerful quad-core RISC-V processor BeagleV Ahead is designed as an affordable RISC-V enabled pocket-size computer for anybody who want's to dive deep into the new RISC-V ISA.



License Terms

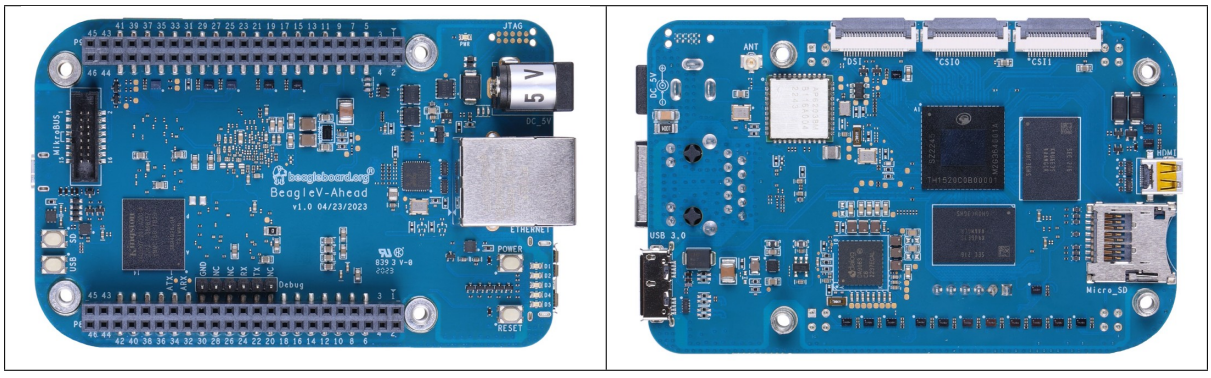
- This documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page
- For export, emissions and other compliance, see *Support*



Important: This is a work in progress, for latest documentation please visit https://docs.beagleboard.org/latest/

8.1 Introduction

BeagleV-Ahead is a high-performance open-source RISC-V single board computer (SBC) built around the Alibaba TH1520 SoC. It has the same P8 & P9 cape header pins as BeagleBone Black allowing you to stack your favourite BeagleBone cape on top to expand it's capability. Featuring a powerful quad-core RISC-V processor BeagleV Ahead is designed as an affordable RISC-V enabled pocket-size computer for anybody who want's to dive deep into the new RISC-V ISA.



8.1.1 Pinout Diagrams

Choose the cape header to see respective pinout diagram.

P8 cape header

P9 cape header

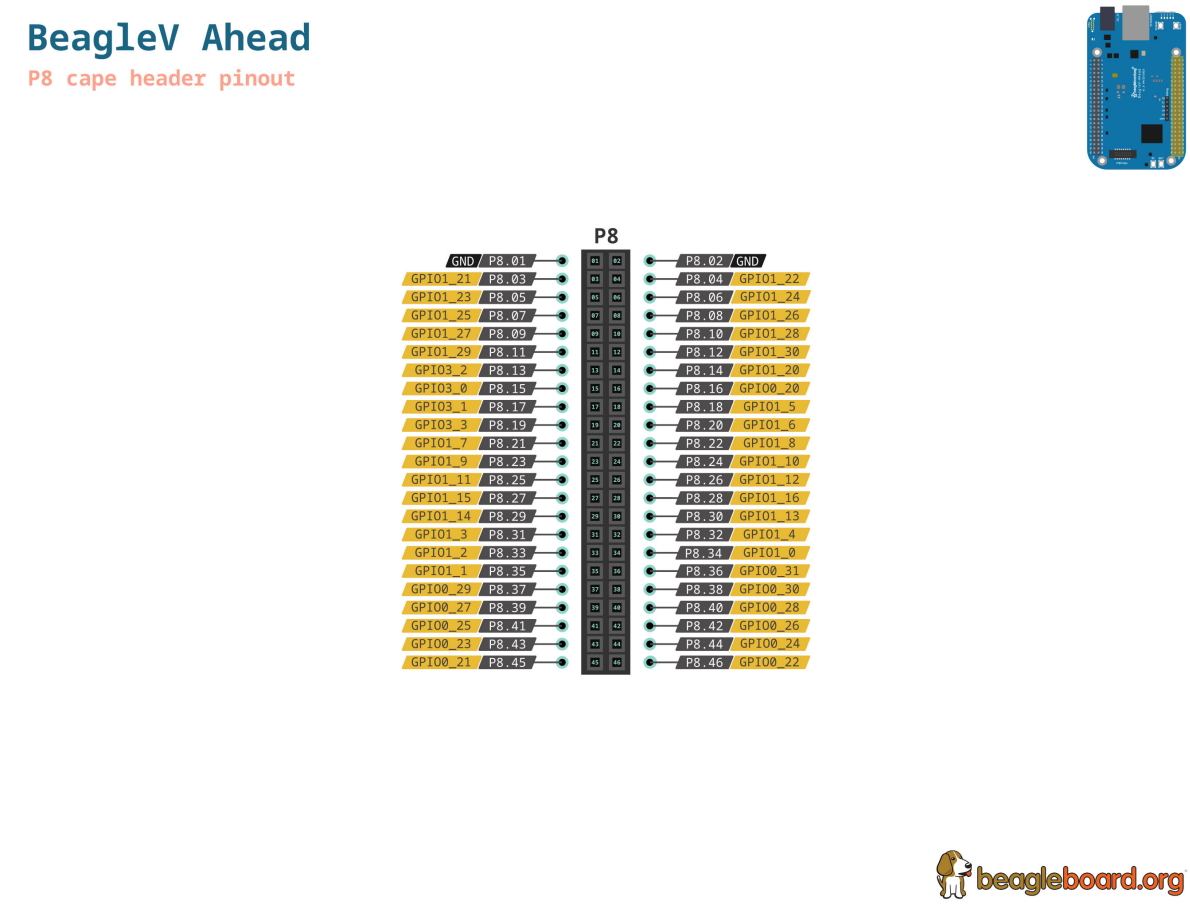


Fig. 8.1: BeagleV Ahead P8 cape header pinout

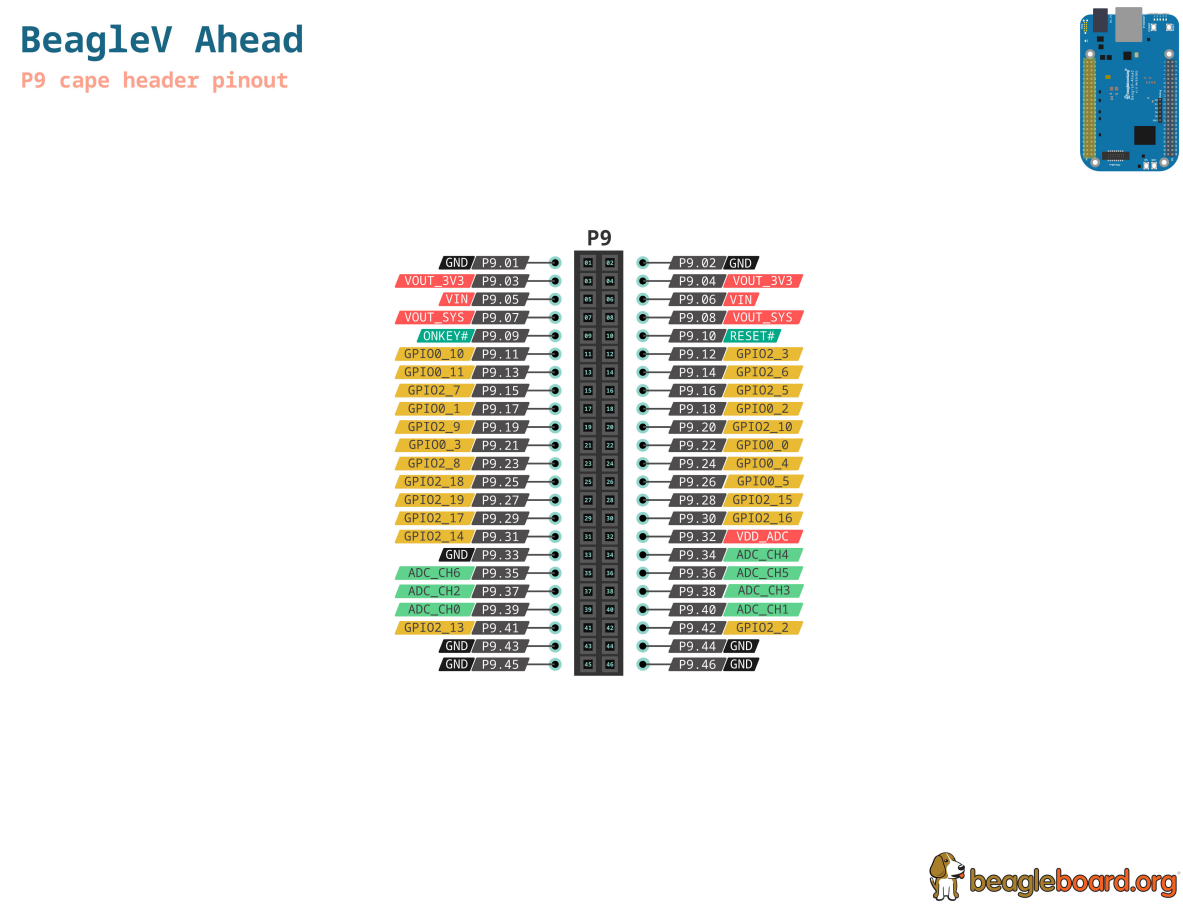


Fig. 8.2: BeagleV Ahead P9 cape header pinout

8.1.2 Detailed overview

BeagleV Ahead is build around T-Head TH1520 RISC-V SoC with quad-core Xuantie C910 processor clocked at 1.85GHz with a 4 TOPS NPU, support for 64-bit DDR, and audio processing using a single core C906.

Todo: remove "<To-Do>" items in the table below.

Table 8.1: BeagleV Ahead features

Feature	Description
Processor	T-Head TH1520 (quad-core Xuantie C910 processor)
PMIC	DA9063
Memory	4GB LPDDR4
Storage	16GB eMMC
WiFi/Bluetooth	• PHY: AP6203BM • Antennas: 2.4GHz & 5GHz
Ethernet	• PHY: Realtek RTL8211F-VD-CG Gigabit Ethernet phy • Connector: integrated magnetics RJ-45
microUSB 3.0	• Connectivity: USB OTG, Flash support • Power: Input: 5V @ <To-Do>, Output: 5V @ <To-Do>
HDMI	• Transmitter: TH1520 Video out system • Connector: Mini HDMI
Other connectors	• microSD • mikroBUS shuttle connector (I2C/UART/SPI/ADC/PWM/GPIO) • 2 x CSI connector compatible with BeagleBone AI-64, Raspberry Pi Zero / CM4 (22-pin) • DSI connector

Board components location

This section describes the key components on the board, their location and function.

Front components location

Table 8.2: BeagleV Ahead board front components location

Feature	Description
Power LED	Power (Board ON) indicator
JTAG (TH1520)	TH1520 SoC JTAG debug port
Barrel jack	Power input
GigaBit Ethernet	1Gb/s Wired internet connectivity
User LEDs	Five user LEDs, *Power and boot* section provides more details. These LEDs are connect to the TH1520 SoC
Reset button	Press to reset BeagleV Ahead board (TH1520 SoC)
Power button	Press to shut-down (OFF), hold down to boot (ON)
P8 & P9 cape header	Expansion headers for BeagleBone capes.
UART debug header	6 pin UART debug header
USB boot button	Hold and reset board (power cycle) to flash eMMC via USB port
SD boot button	Hold and reset board (power cycle) to boot from SD Card
mikroBUS shuttle	16pin mikroBUS shuttle connector for interfacing mikroE click boards
16GB eMMC	Flash storage
RTL8211F	Gigabit IEEE 802.11 Ethernet PHY

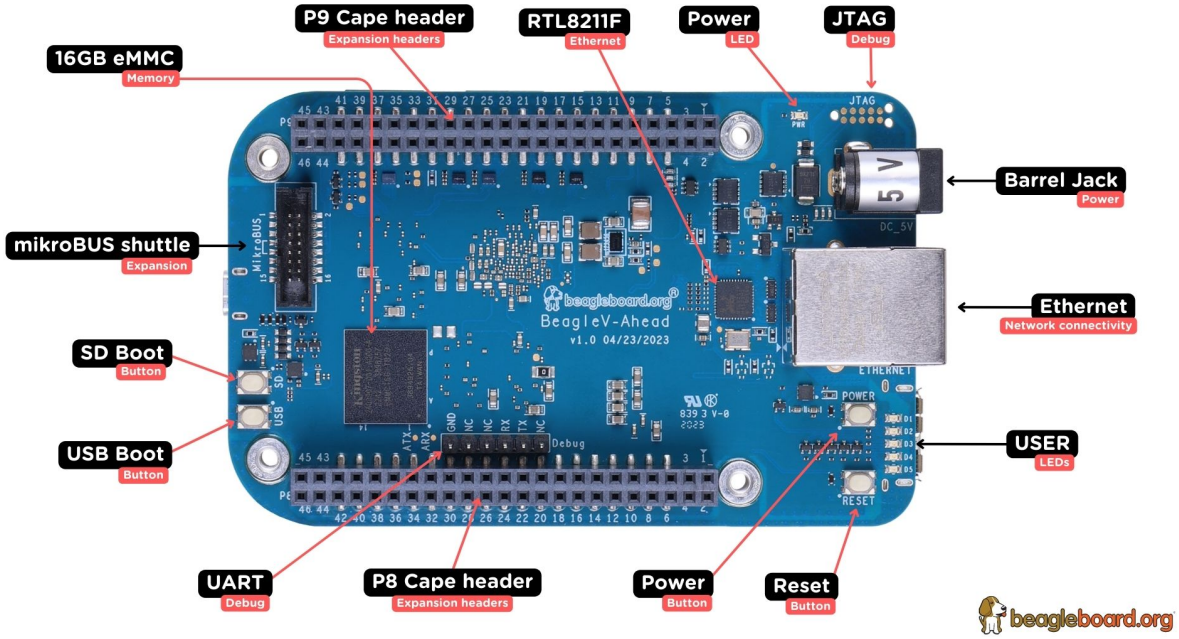


Fig. 8.3: BeagleV Ahead board front components location

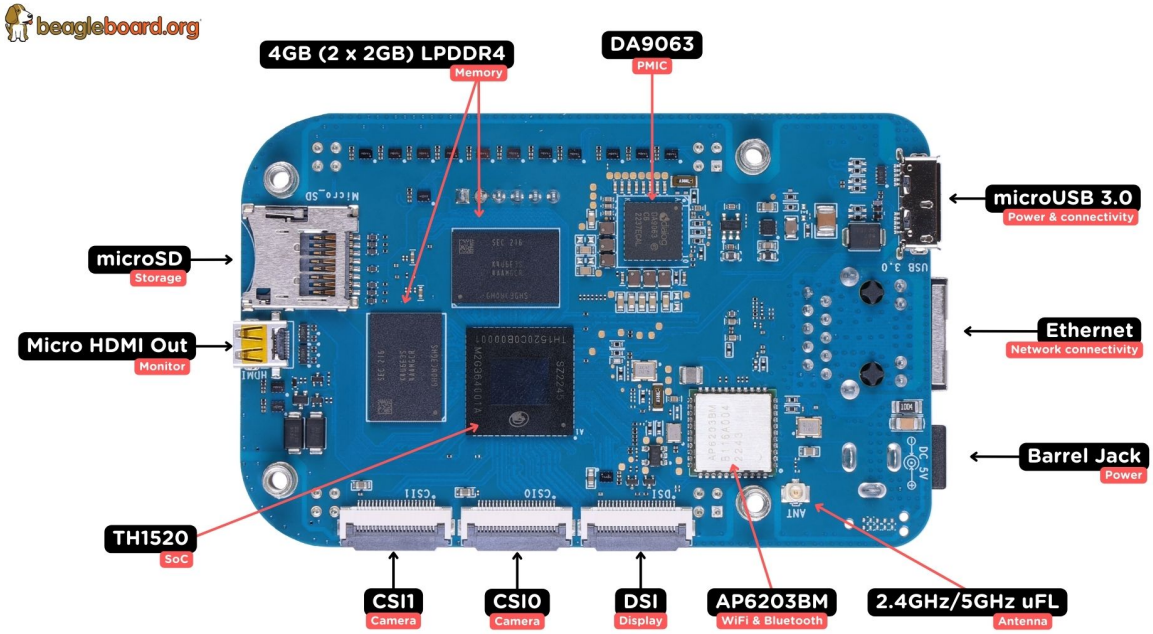


Fig. 8.4: BeagleV Ahead board back components location

Back components location

Table 8.3: BeagleV Ahead board back components location

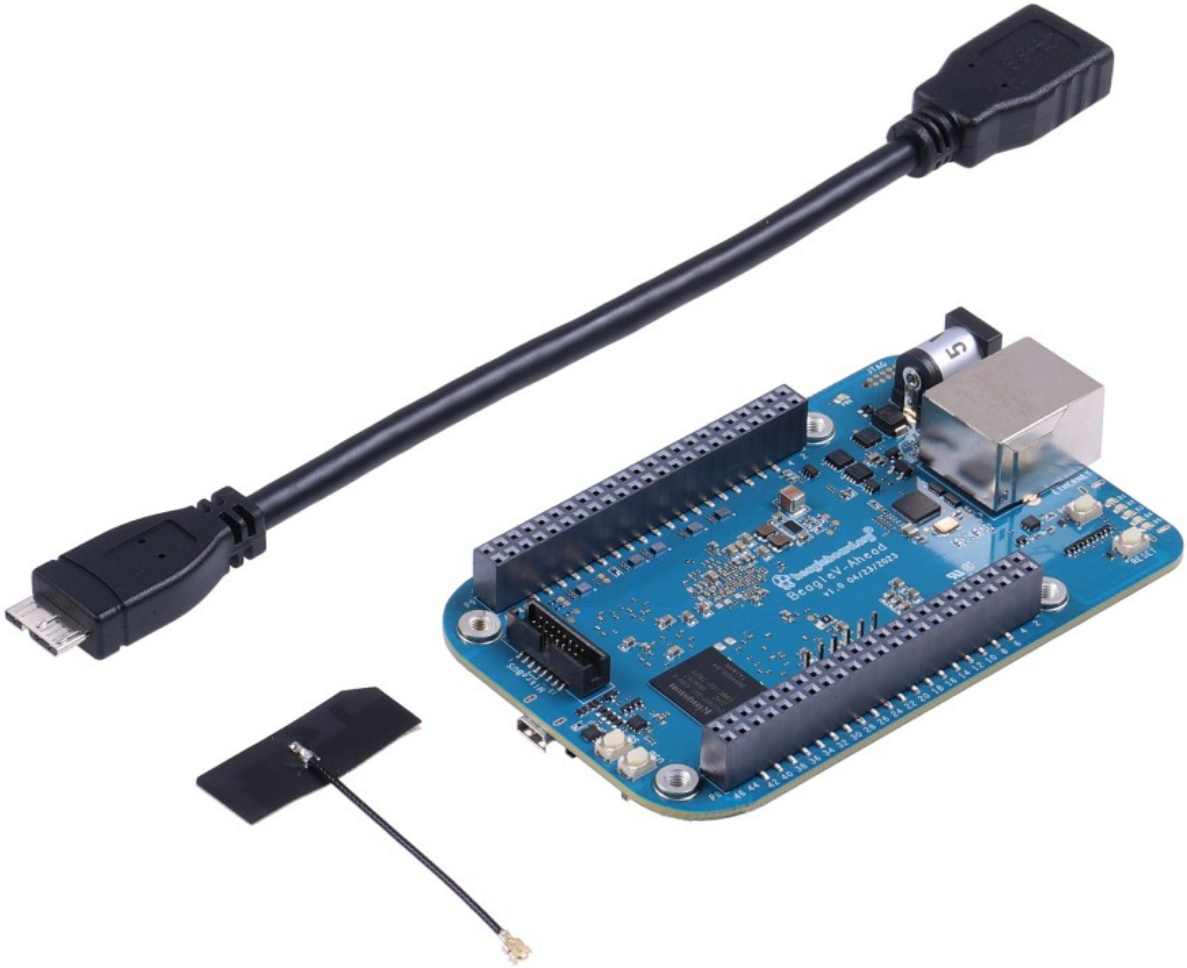
Feature	Description
DA9063	Dialog semi Power Management Integrated Circuit (PMIC)
microUSB 3.0	Power & USB connectivity as client or Host (OTG)
Antenna connector	2.4GHz/5GHz uFL connector
AP6203BM	Ampak WiFi & BlueTooth combo
DSI	MIPI Display connector
CSI0 & CSI1	MIPI Camera connectors
TH1520	T-Head quad-core C910 RISC-V SoC
Mini HDMI	HDMI connector
microSD	SDCard holder
4GB RAM	2 x 2GB LPDDR4 RAM

8.2 Quick Start

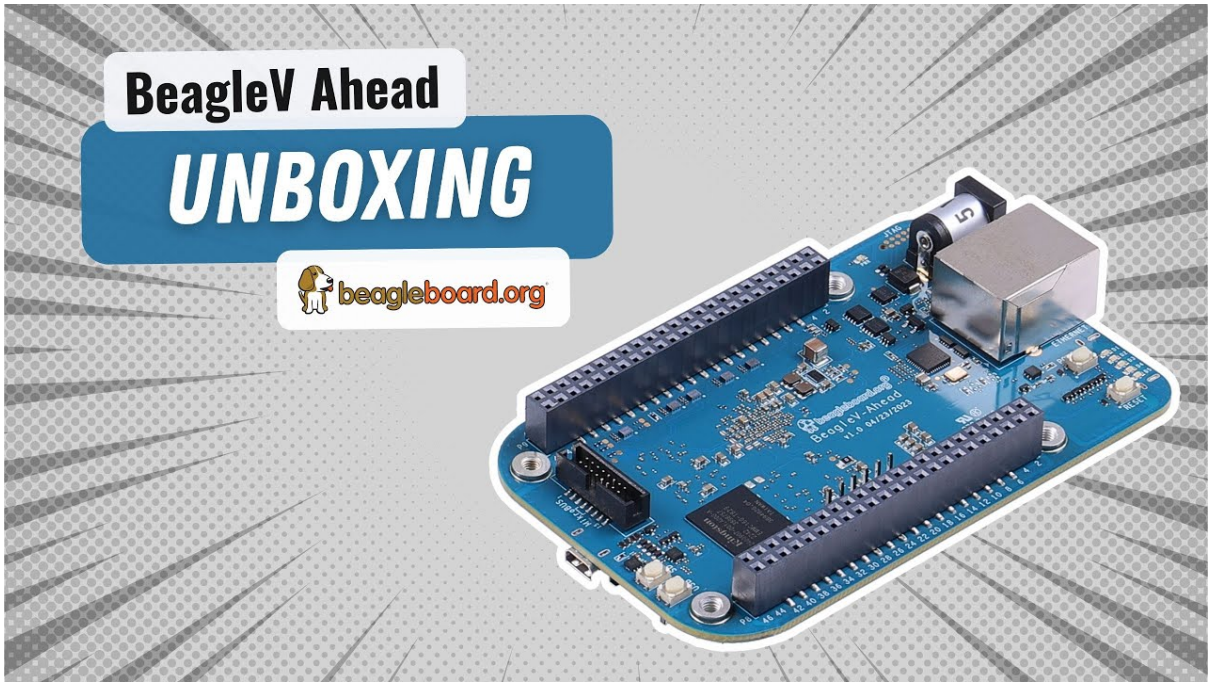
8.2.1 What's included in the box?

When you purchase a brand new BeagleV Ahead, In the box you'll get:

1. BeagleV Ahead board
2. One (1) 2.4GHz/5GHz antenna
3. USB super-speed micro-A plug to type-A receptacle cable (for connecting common USB type-A peripherals)
4. Quick-start card



8.2.2 Unboxing



8.2.3 Antenna guide

Warning: uFL antenna connectors are very delicate and should be handled with care.

Connecting antenna

Disconnecting antenna

To use WiFi you are **required** to connect the 2.4GHz/5GHz antenna provided in BeagleV Ahead box. Below is a guide to connect the antenna to your BeagleV Ahead board.

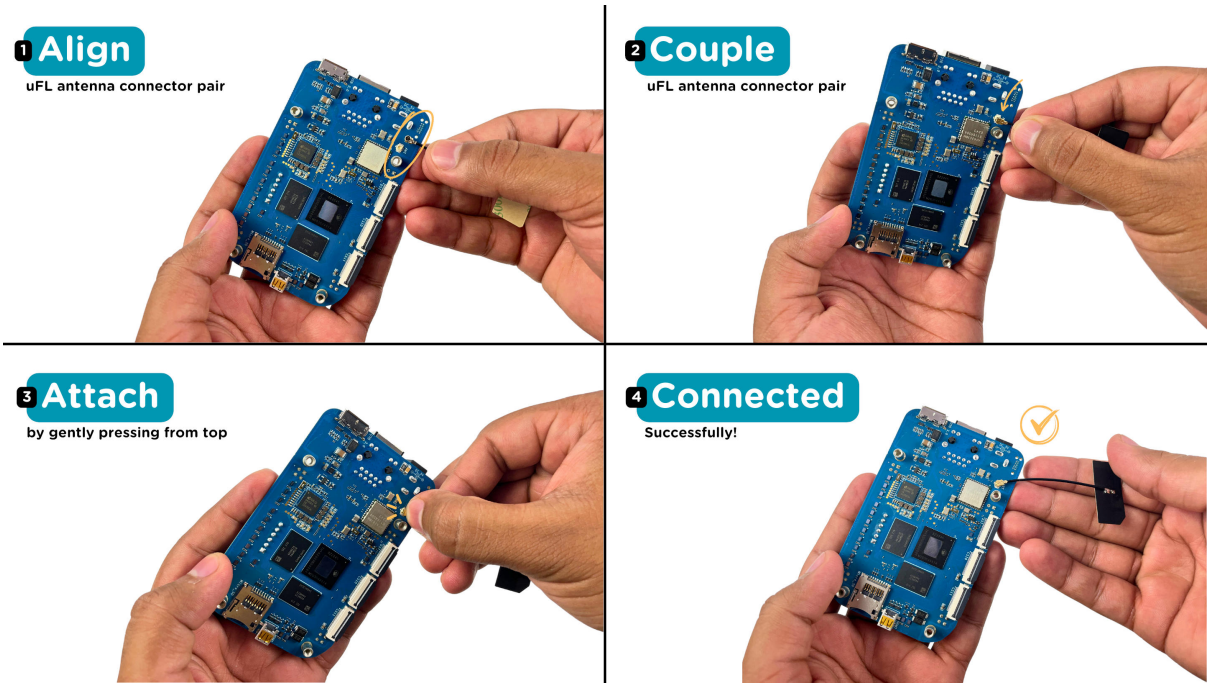


Fig. 8.5: Connecting 2.4GHz/5GHz antenna to BeagleV Ahead.

If for some reason you want to disconnect the antenna from your BeagleV Ahead board you can follow the guide below to remove the antenna without beaking the uFL antenna connector.

8.2.4 Tethering to PC

To connect the board to PC via USB 3.0 port you can use either a standard high-speed micro-B cable or a USB 3.0 super-speed micro-B cable. Connection guide for both are shown below:

Important: high-speed micro-B will support only USB 2.0 speed but super-speed micro-B cable will support USB 3.0 speed.

super-speed micro-B connection (USB 3.0)

high-speed micro-B connection (USB 2.0)

For super speed USB 3.0 connection it's recommended to use super-speed micro-B USB cable. To get a super-speed micro-B cable you can checkout links below:

1. USB 3.0 Micro-B Cable - 1m (sparkfun)
2. Stewart Connector super-speed micro-B (DigiKey)
3. CNC Tech super-speed micro-B (DigiKey)

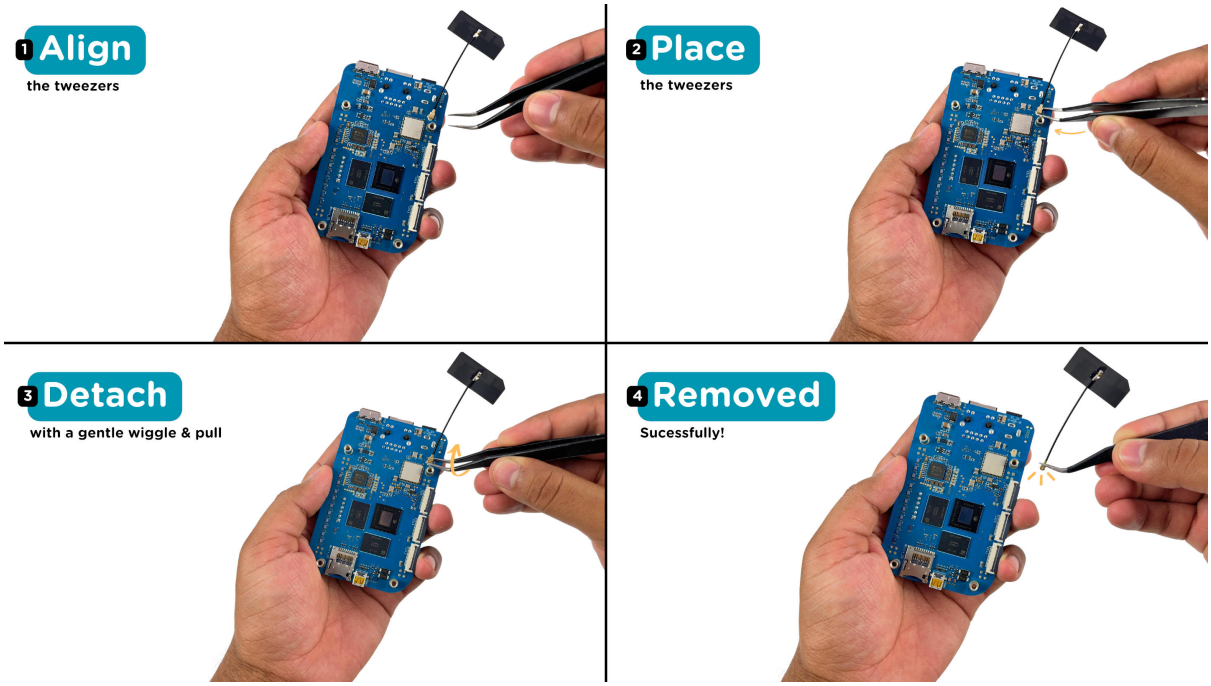


Fig. 8.6: Removing 2.4GHz/5GHz antenna to BeagleV Ahead.

4. Assmann WSW Components super-speed micro-B (DigiKey)

Note: If you only have a high-speed micro-B cable you can checkout high-speed micro-B connection guide.

For USB 2.0 connection it's recommended to use high-speed micro-B USB cable. To get a high-speed micro-B cable you can checkout links below:

1. USB micro-B Cable - 6 Foot (sparkfun)
2. Stewart Connector high-speed micro-B (DigiKey)
3. Assmann WSW Components high-speed micro-B (DigiKey)
4. Cvilux USA high-speed micro-B (DigiKey)

Note: Make sure the high-speed micro-B cable you have is a data cable as some high-speed micro-B cables are power only.

8.2.5 Flashing eMMC

Note: To flash your BeagleV Ahead you need either a super-speed micro-B or high-speed micro-B cable as shown in section above.

Download latest software image

To download the latest software image visit https://www.beagleboard.org/distros and search for BeagleV Ahead as shown below.

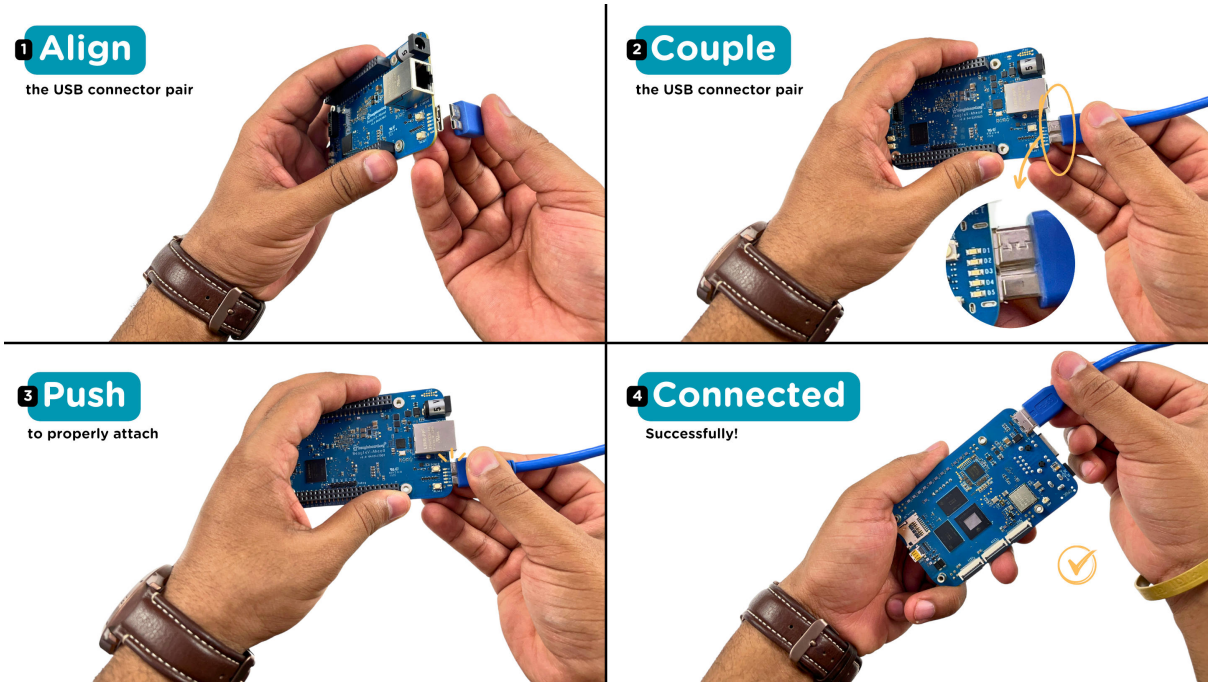


Fig. 8.7: super-speed micro-B (USB 3.0) connection guide for BeagleV Ahead.

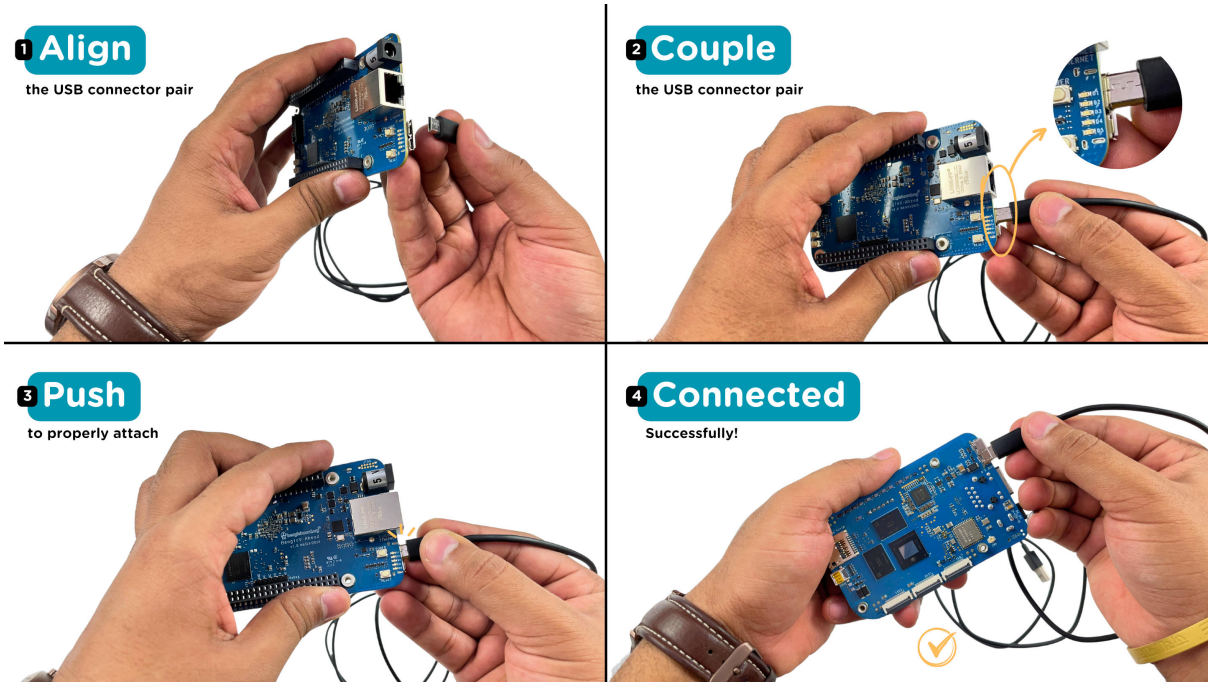


Fig. 8.8: high-speed micro-B (USB 2.0) connection guide BeagleV Ahead.

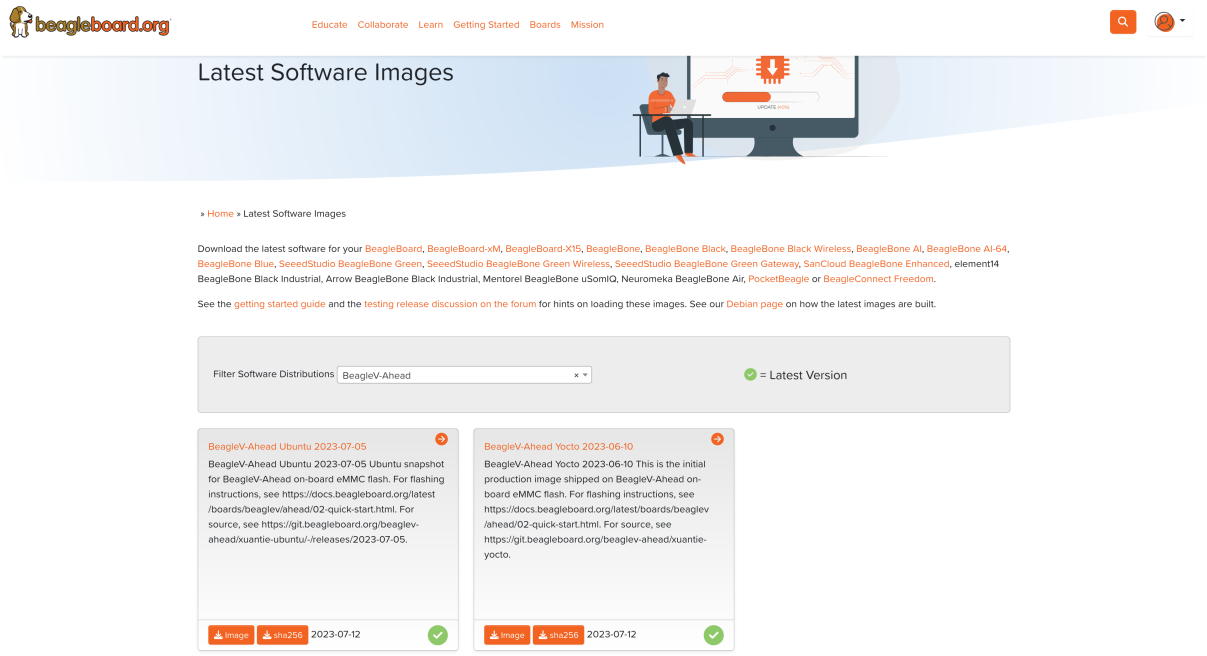


Fig. 8.9: Download latest software image for BeagleV Ahead board

Put BeagleV Ahead in USB flash mode

Note: Only super-speed micro-B is shown in graphic below but you can use a high-speed micro-B cable. Only difference will be lower flash speeds.

To put your BeagleV Ahead board into eMMC flash mode you can follow the steps below :-

1. Press and hold USB button.
2. Connect to PC with super-speed micro-B or high-speed micro-B cable.
3. Release USB button.

Important: If you want to put the board into eMMC flashing while it is already connected to a PC you can follow these steps:

1. Press and hold USB button.
2. Press reset button once.
3. Release USB button.

Flash the latest image on eMMC

Linux

Windows

Mac

First you need to install android platform tools which includes *adb* and *fastboot*.

- Debian/Ubuntu-based Linux users can type the following command:

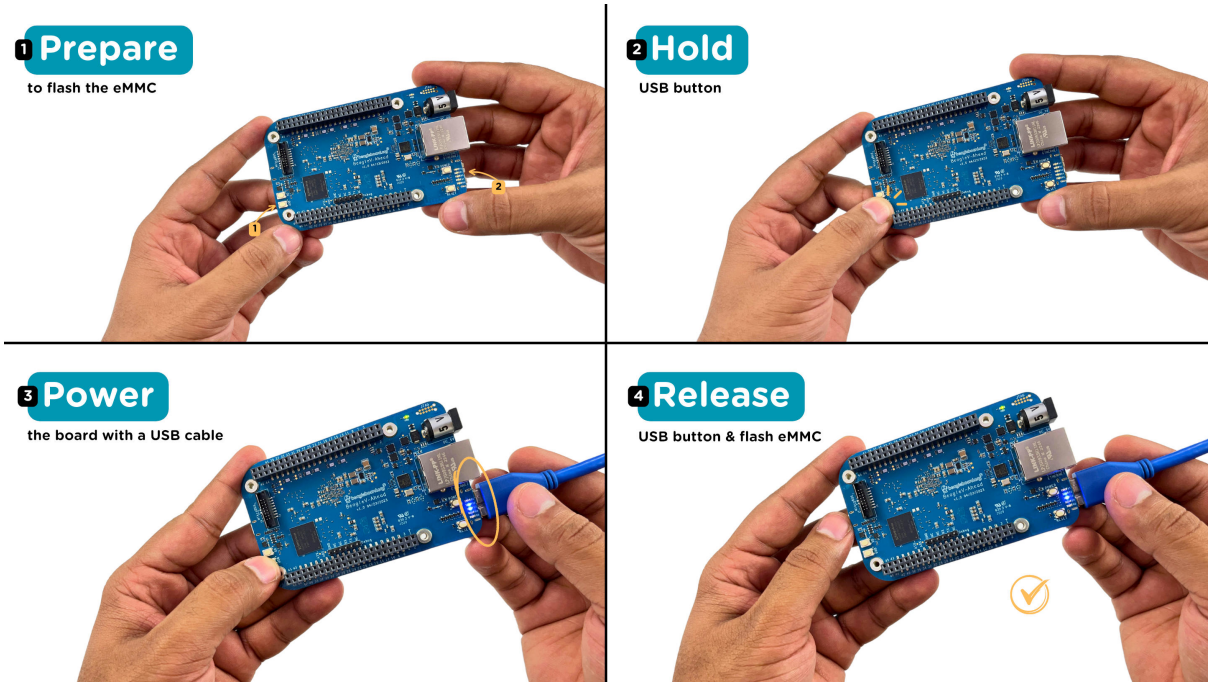


Fig. 8.10: Connecting BeagleV Ahead to flash eMMC

```
sudo apt-get install android-sdk-platform-tools
```

- Fedora/SUSE-based Linux users can type the following command:

```
sudo dnf install android-tools
```

Now unzip the latest software image zip file you have downloaded from https://www.beagleboard.org/distros which contains four files shown below:

```
[lorforlinux@fedora deploy] $ ls
boot.ext4  fastboot_emmc.sh  root.ext4  u-boot-with-spl.bin
```

Important: Make sure your board is in flash mode, you can follow the guide above to do that.

To flash the board you just have to exexute the script *fastboot_emmc.sh* as root and provide your passoword:

```
[lorforlinux@fedora deploy] $ sudo ./fastboot_emmc.sh
[sudo] password for lorforlinux:
```

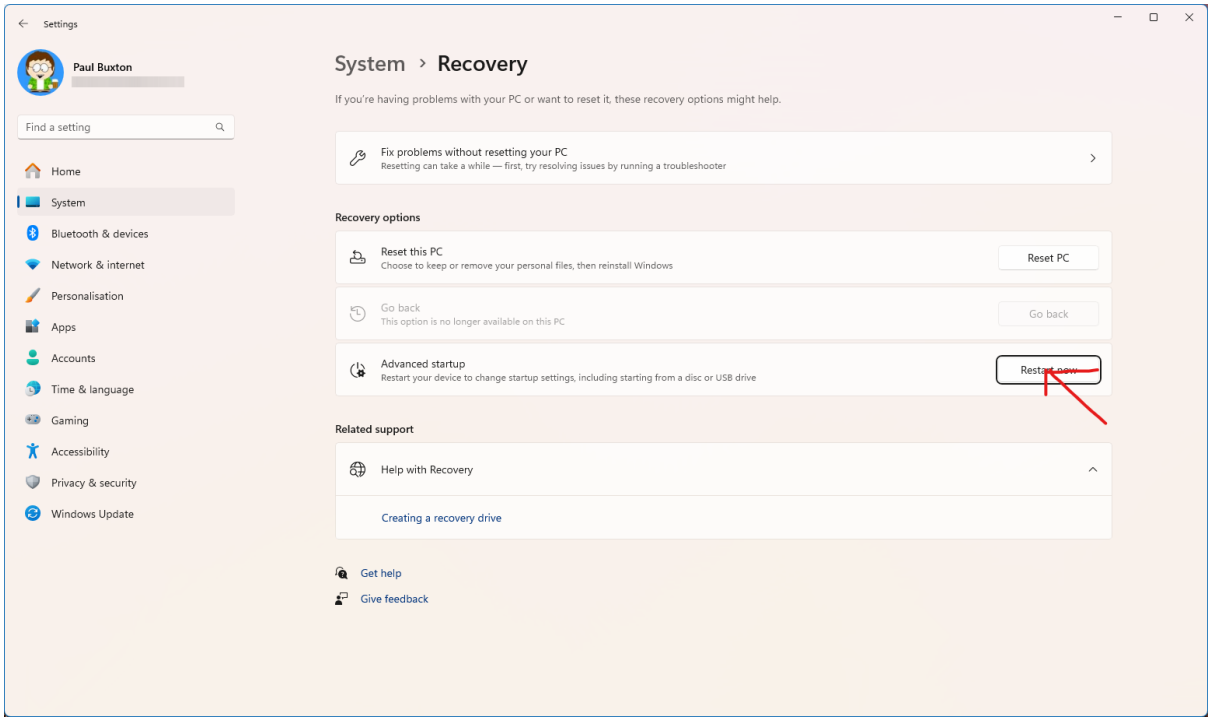
To flash the image using Windows you need the following prerequisites :-

- Android platform tools https://dl.google.com/android/repository/platform-tools-latest-windows.zip
- THead Android Driver which is in the burn_tool.zip archive which can be downloaded from https://mega.nz/folder/phoQlBTZ#cZeQ3qZ__pDvP94PT3_bGA

Extract these using the *'Right Click' - 'Extract All'* menu option on each file.

To install the Android driver you will need to boot your system with driver certification turned off. To do this open the Windows Settings app and select *'System, Recovery'*. In the window there click the '*Restart Now'* button in the Advanced Startup section.

At this point your machine will restart into the advanded recovery menu. Here select *'Troubleshoot' -> 'Advanced Options' -> 'Start up Settings'*. Then click on *'Restart'*.



This will restart Windows with a menu where you can now select *'(7) Disable Driver Signature enforcement'*. Select this option and Windows will now startup with driver certification enforcement disabled (this is temporary and will revert when you next reboot).

Once Windows has restarted, open up the Windows Device Manager window. The easiest way is to Right Click on *'This PC'* in the File Manager, select *'More Options'* then *'Manage'*.

Connect up your board, and put it into flashing mode as desribed above. The board should show up as a USB Download Gadget in the Management console.

Right click on this and select *'Update driver'*, and point at the folder where you extraced the THead Android driver. Windows will warn that the driver is not certified, select *'Install this Software Anyway'* to install the driver.

You can now use the fastboot.exe in the Android Platform tools archive to flash the image (instructions below assume the image files are in the same folder as fastboot.exe)

```
fastboot.exe flash ram u-boot-with-spl.bin
fastboot.exe reboot
fastboot.exe oem format
fastboot.exe flash uboot u-boot-with-spl.bin
fastboot.exe flash boot boot.ext4
fastboot.exe flash root thead-image-linux-light-beagle-*.rootfs.ext4
fastboot.exe reboot
```

You have now flashed your device!

Todo: add instructions for flashing in Mac.

8.2.6 Access UART debug console

Note: It has been noticed that 6pin FTDI cables like this doesn't seem work with BeagleV Ahead debug port and there might be other cables/modules that will show garbage when connected to the board.

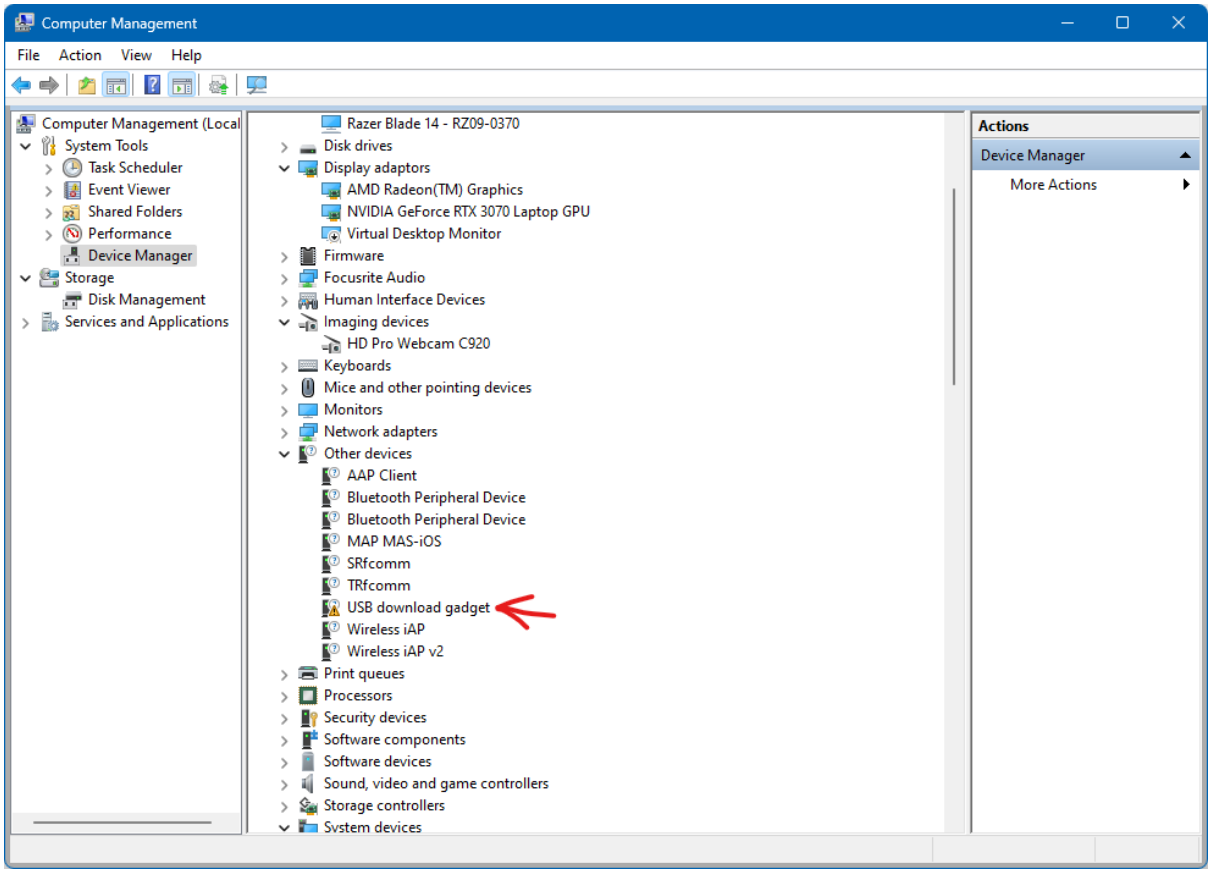


Fig. 8.11: Windows Device Manager

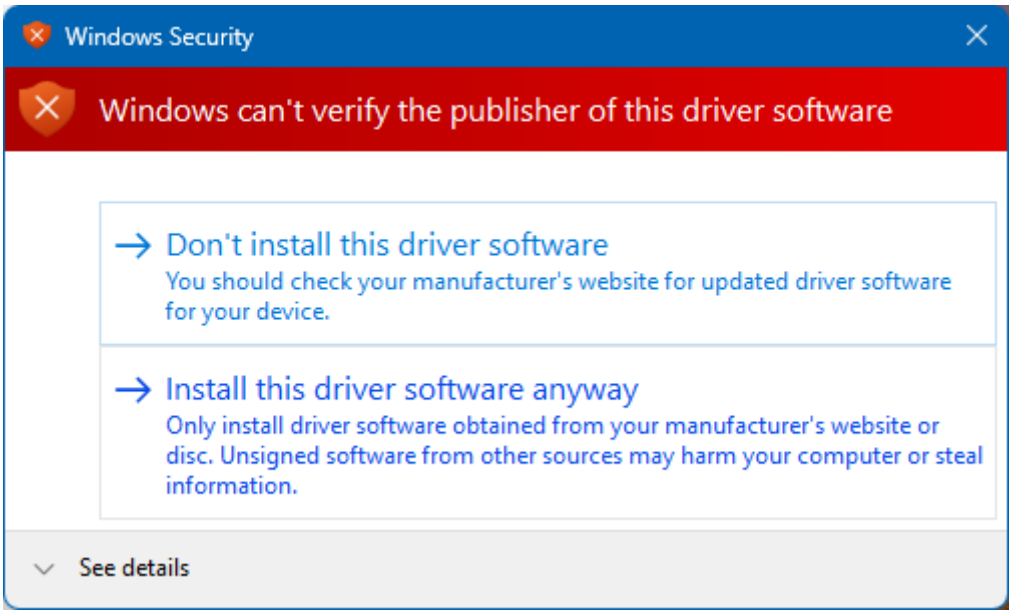


Fig. 8.12: Unverified Driver Warning

Some tested devices that are working good includes:

1. Adafruit CP2102N Friend - USB to Serial Converter
2. Raspberry Pi Debug Probe Kit for Pico and RP2040

To access a BeagleV Ahead serial debug console you can connected a USB to UART to your board as shown below:

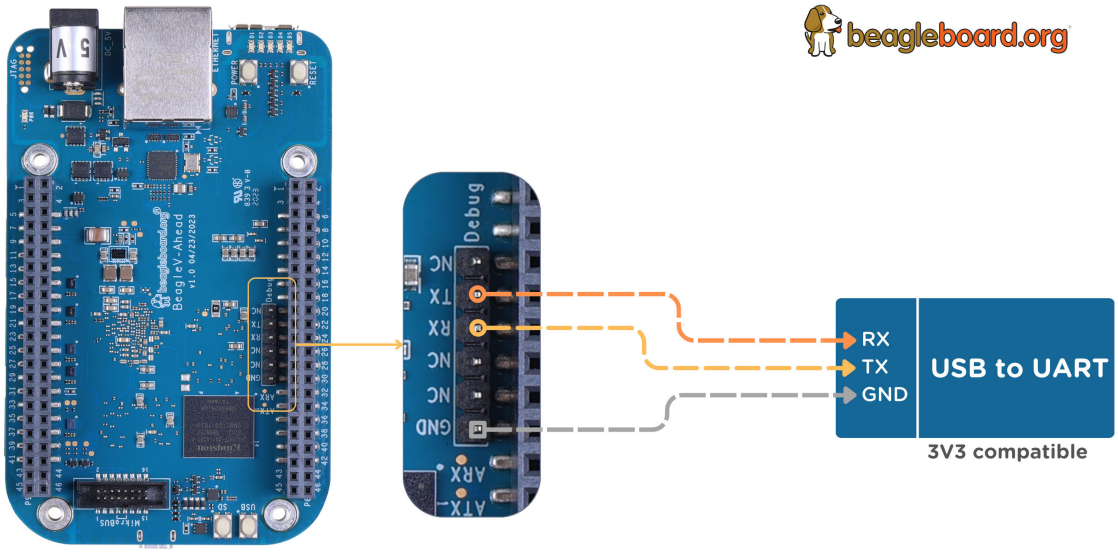


Fig. 8.13: BeagleV Ahead UART debug port connection

To see the board boot log and access your BeagleV Ahead's console you can use application like `tio` to access the conole. If you are using Linux your USB to UART converter may appear as `/dev/ttyUSB`. It will be different for Mac and Windows operatig systems. To find serial port for your system you can checkout this guide.

```
[lorforlinux@fedora ~] $ tio /dev/ttyUSB0
tio v2.5
Press ctrl-t q to quit
Connected
```

8.2.7 Connect USB gadgets

A super-speed micro-B (male) to USB A (female) OTG cable included in the box can be used to connect USB gadgets to your BeagleV Ahead board. When you do this, you'll be required to power the board via Barrel jack.

Important: To properly power the board and USB gadgets you must power the board with 5V @ 2A power supply.

8.2.8 Connect to WiFi

Yocto

After getting access to the UART debug console you will be prompted with,

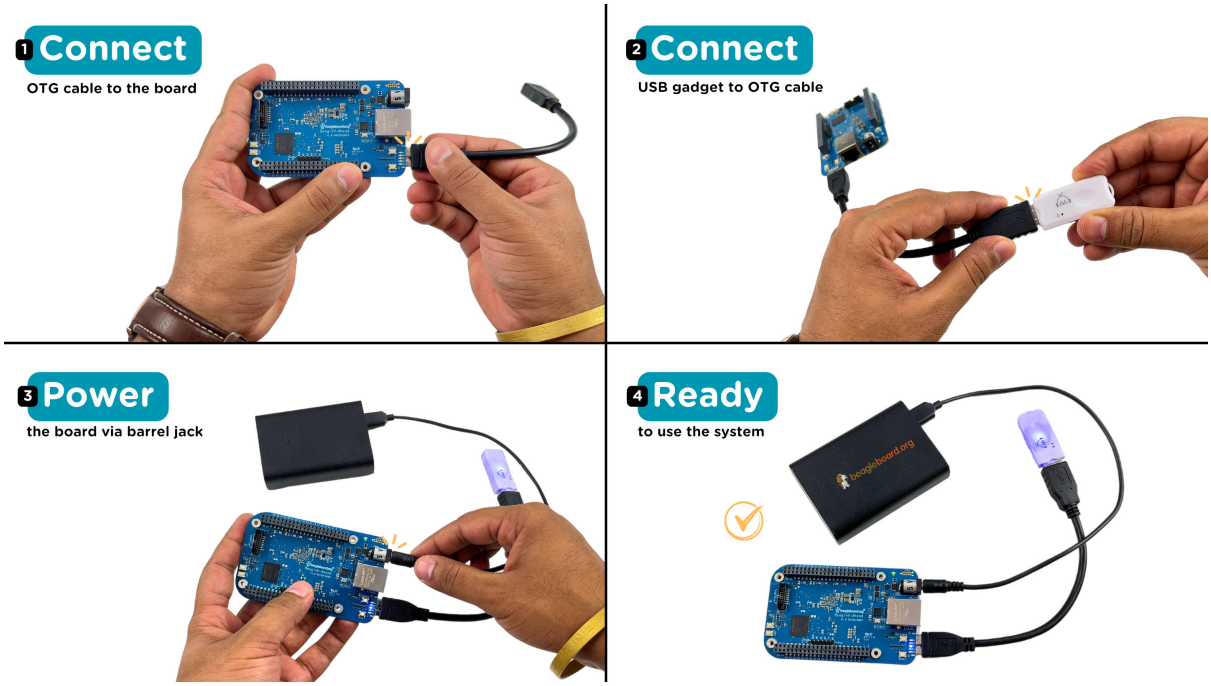


Fig. 8.14: USB OTG to connect USB gadgets to BeagleV Ahead board

```
THEAD C910 Release Distro 1.1.2 BeagleV ttyS0

BeagleV login:
```

Here you have to simply type `root` and press enter to start uisng your BeagleV Head board. Once you are in, to connect to any WiFi access point you have to edit the `/etc/wpa_supplicant.conf`

```
root@BeagleV:~# nano /etc/wpa_supplicant.conf
```

In the `wpa_supplicant.conf` file you have to provide `ssid` and `psk`. Here `ssid` is your WiFi access point name and `psk` is the passoword. It should look as shown below:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
ap_scan=1
update_config=1

network={
        ssid="My WiFi" ①
        psk="passoword" ②
        key_mgmt=WPA-PSK
}
```

① WiFi access point name

② WiFi passoword

Once you are done with editing the file you can save the file with `CTRL+O` and exit the nano editor with `CTRL+X`. Once you are back to terminal reconfigure the `wlan0` wireless interface which will trigger it to connect to the access point with the credentials you have added to `wpa_supplicant.conf`. Execute the command below to reconfigure `wlan0` wireless interface.

```
root@BeagleV:~# wpa_cli -i wlan0 reconfigure
OK
```

After executing this you can check if internet is working by executing `ping 8.8.8.8` as shown below:

```
root@BeagleV:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=118 time=13.676 ms
64 bytes from 8.8.8.8: seq=1 ttl=118 time=17.050 ms
64 bytes from 8.8.8.8: seq=2 ttl=118 time=14.367 ms
64 bytes from 8.8.8.8: seq=3 ttl=118 time=19.320 ms
64 bytes from 8.8.8.8: seq=4 ttl=118 time=14.796 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 13.676/15.841/19.320 ms
```

Important: Due to a software issue Yocto might now assign any ip address to wlan0 wireless interface thus even if you are connected successfully to the access point of your choice you will still not be able to connect to the internet. Particularly If you are not getting any pings back when you execute `ping 8.8.8.8` you must execute the commands below:

1. `root@BeagleV:~# cp /lib/systemd/network/80-wifi-station.network.example /lib/systemd/network/80-wifi-station.network`
2. `root@BeagleV:~# networkctl reload`

this should fix the no internet issue on your BeagleV Ahead board!

8.2.9 Demos and Tutorials

- *Using CSI Cameras*

8.3 Design & specifications

If you want to know how BeagleV Ahead board is designed and what are it's high-level specifications then this chapter is for you. We are going to discuss each hardware design element in detail and provide high-level device specifications in a short and crisp form as well.

8.3.1 Block diagram

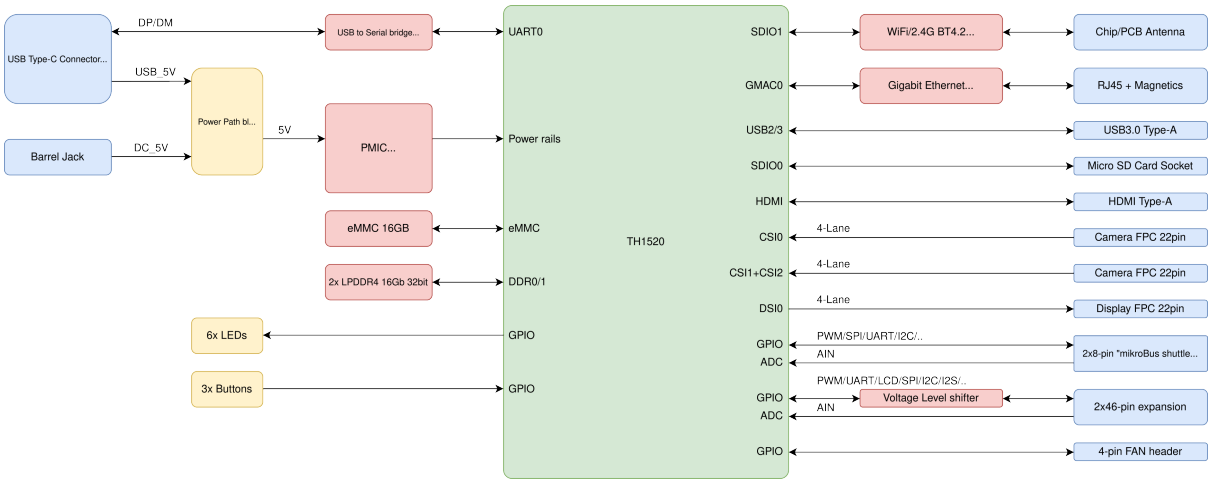


Fig. 8.15: System block diagram

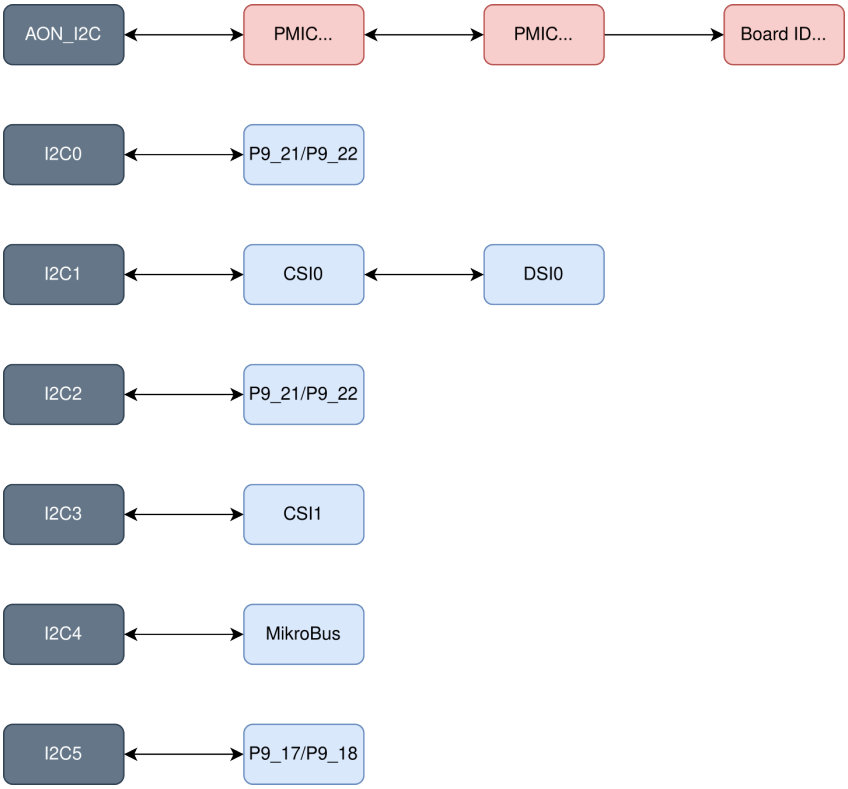


Fig. 8.16: I2C-Usage diagram

8.3.2 System on Chip (SoC)

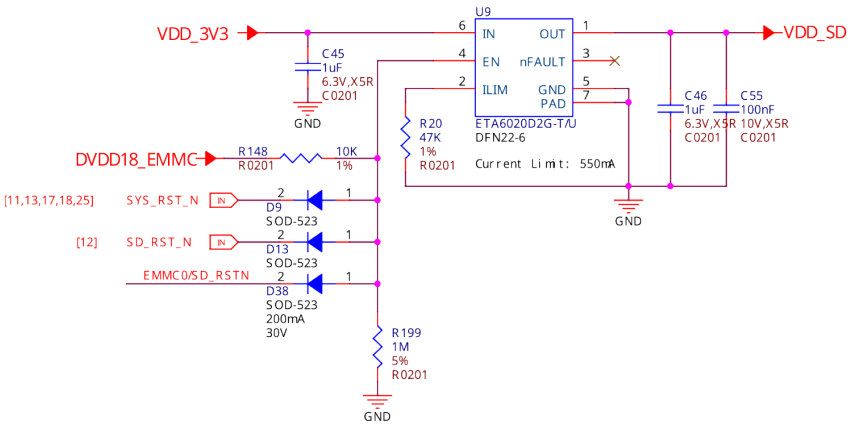


Fig. 8.17: SoC eMMC power switch

8.3.3 Power management

Barrel jack

0.8V DCDC buck

3.3V DCDC buck

1.8V LDO

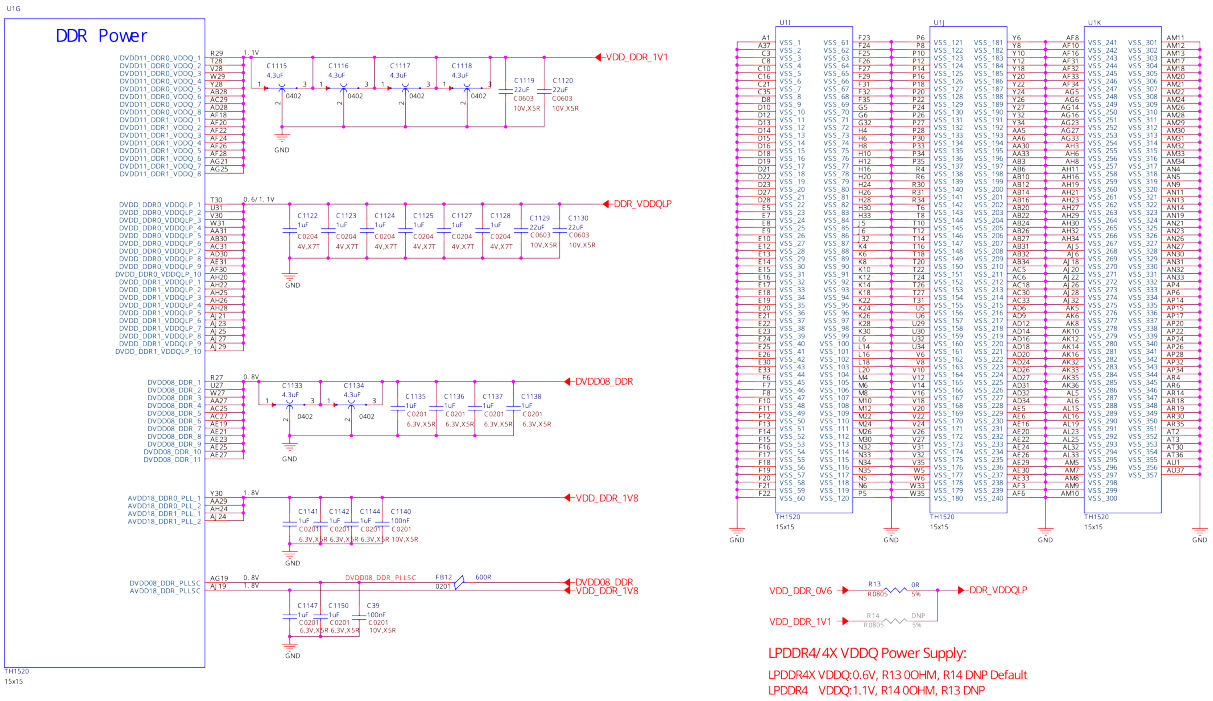


Fig. 8.18: SoC DDR Power

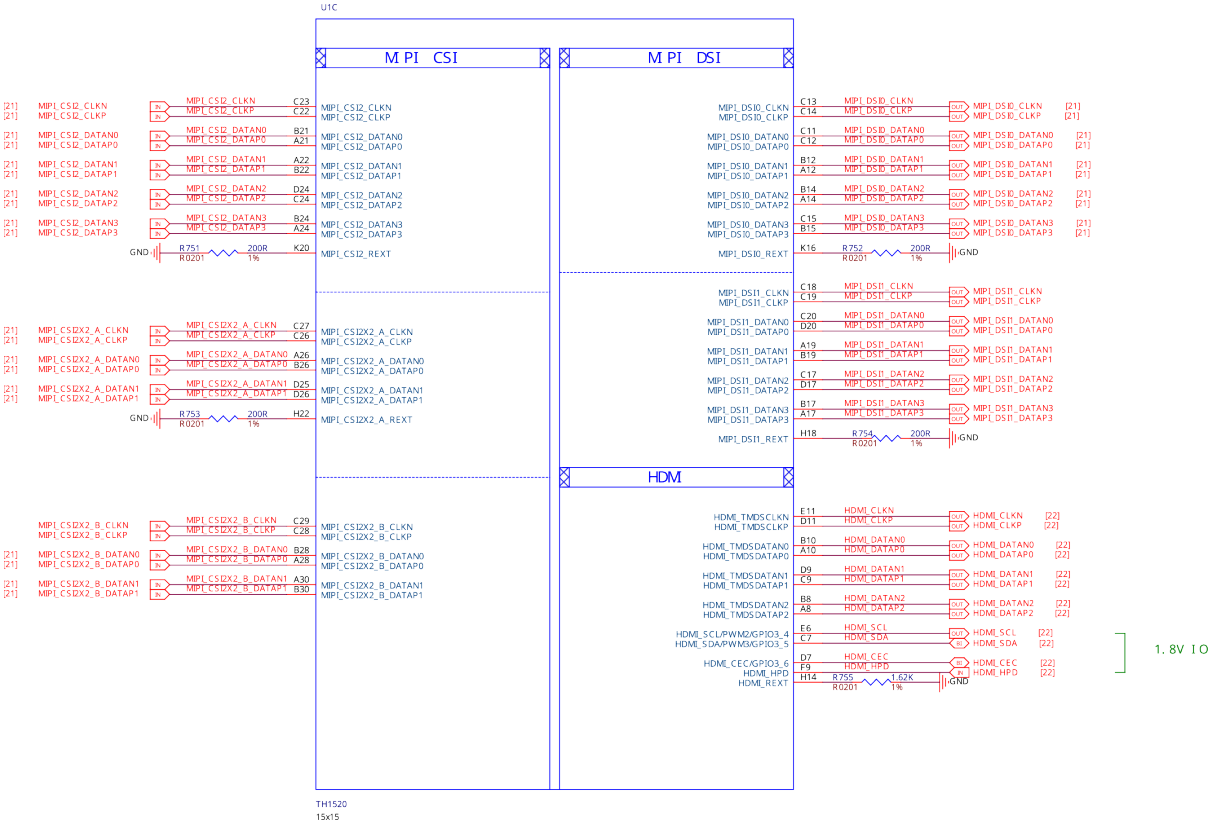


Fig. 8.19: SoC MIPI CSI DSI HDMI

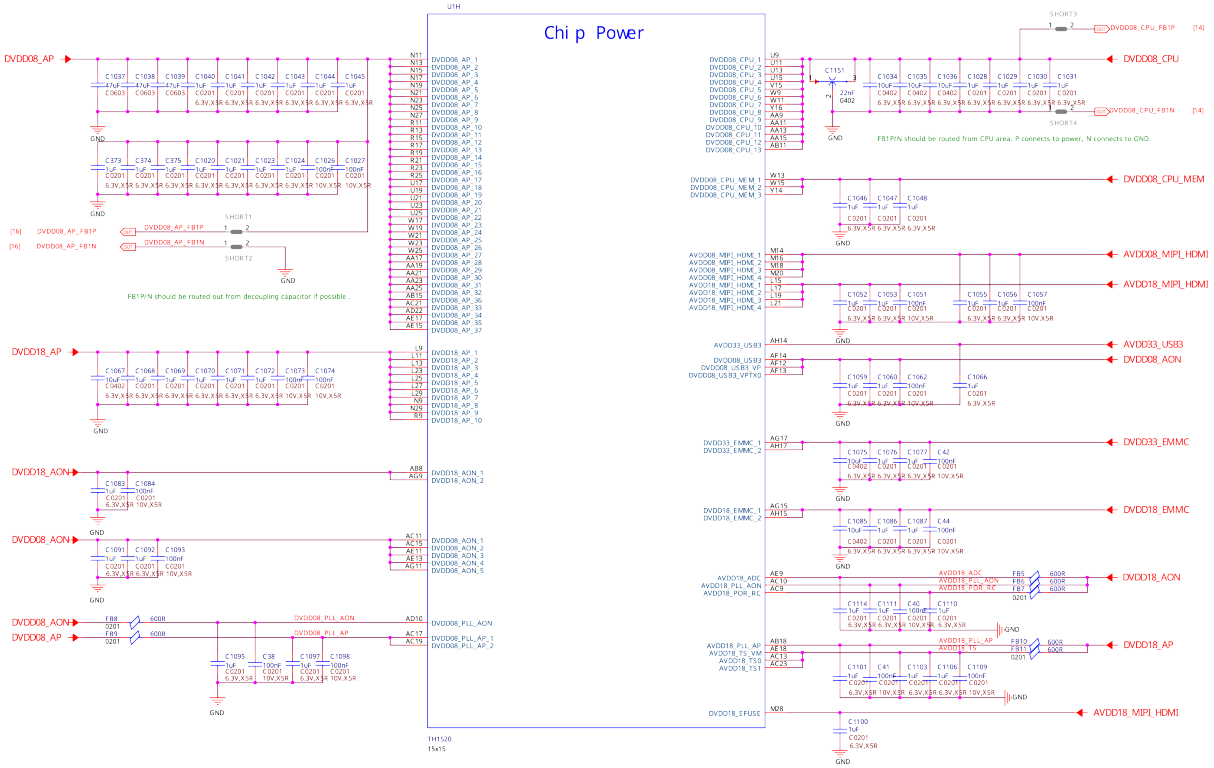


Fig. 8.20: SoC power

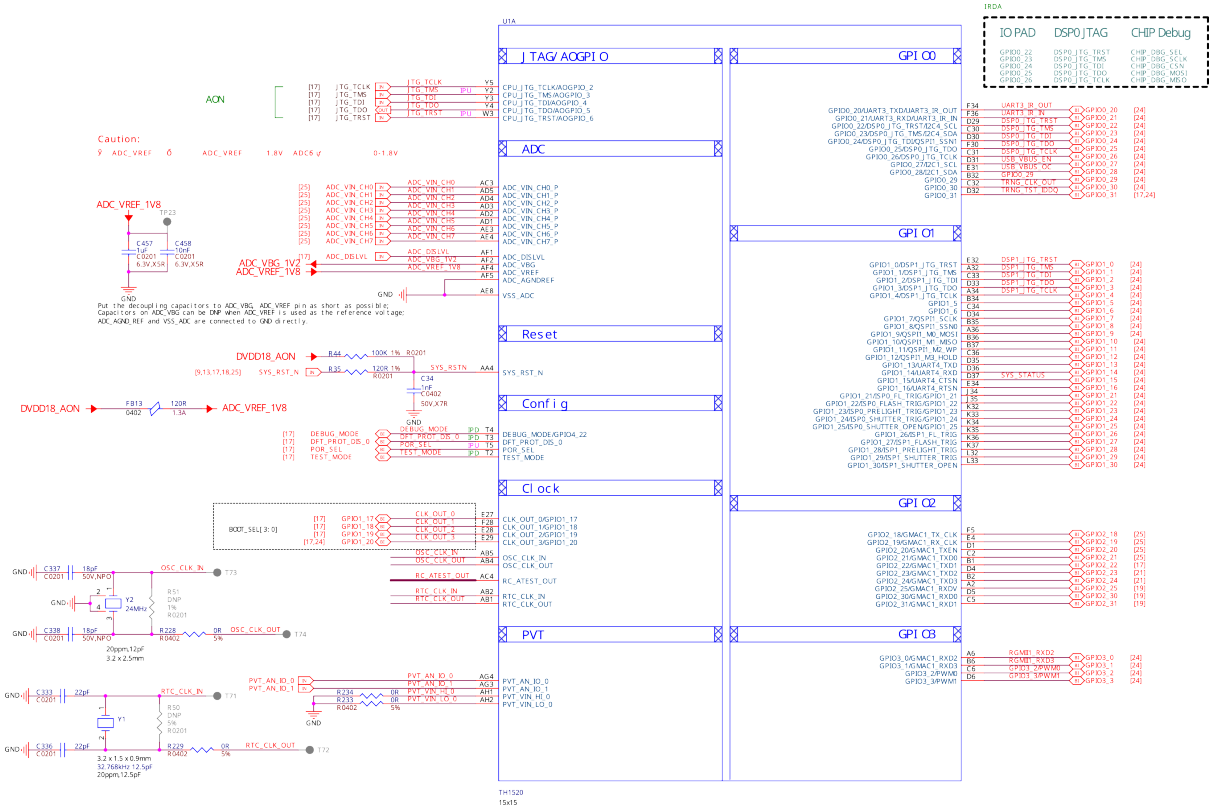


Fig. 8.21: SoC sys, ADC, and Clock

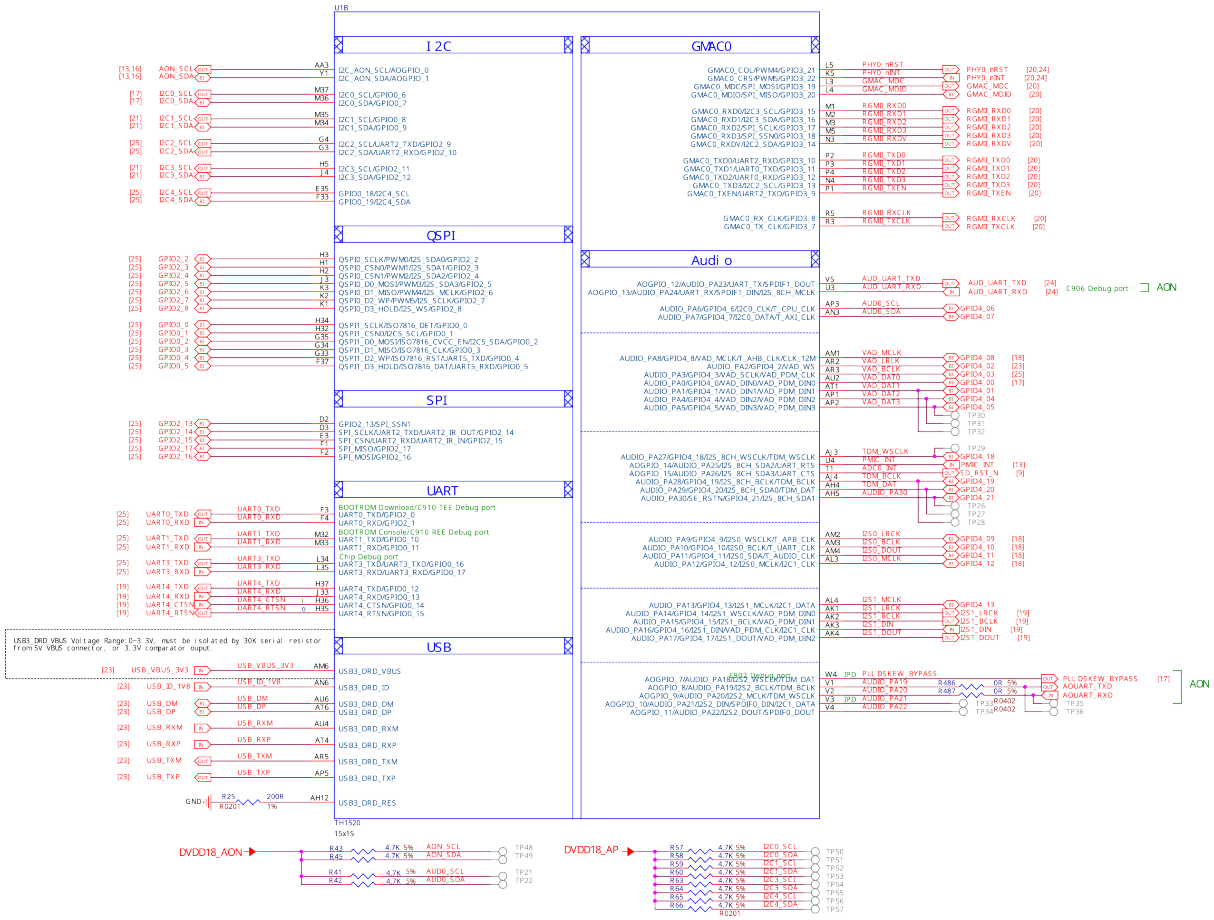


Fig. 8.22: SoC USB GMAC Audio

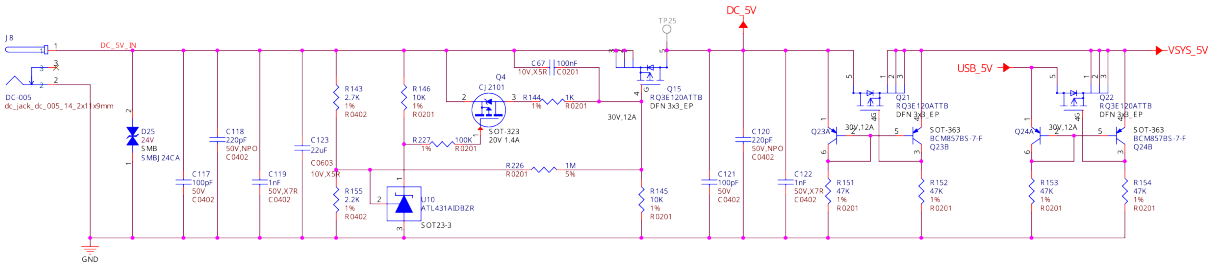


Fig. 8.23: Barrel jack power input

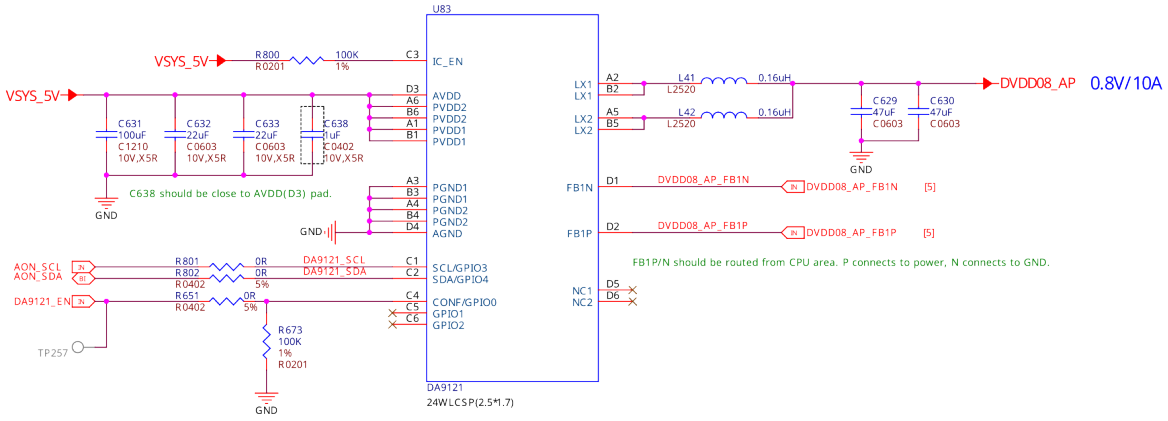


Fig. 8.24: 0.8V DCDC buck converter

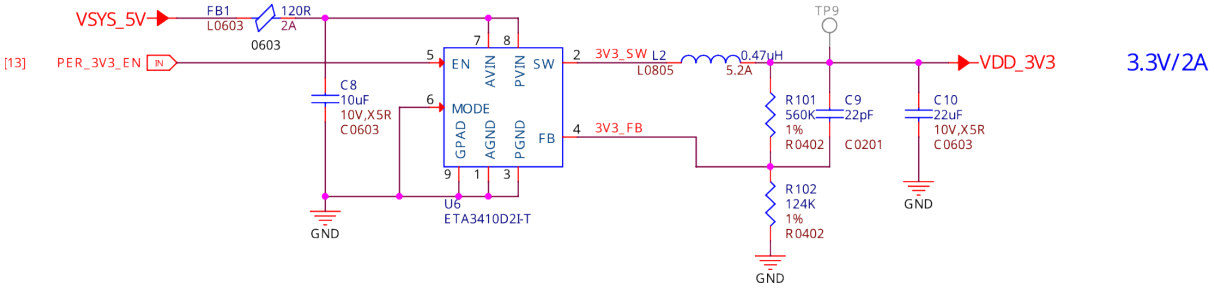


Fig. 8.25: 3.3V DCDC buck converter

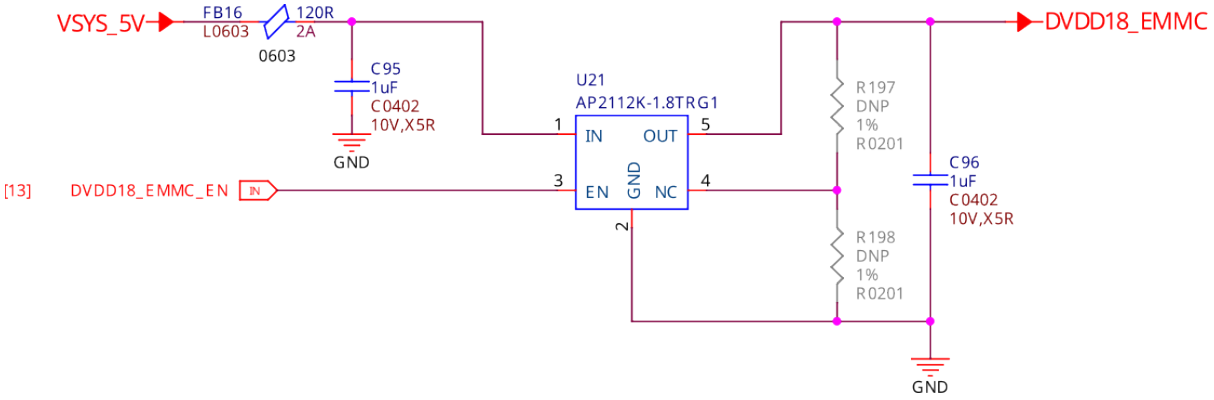


Fig. 8.26: 1.8V LDO regulator

PMIC

8.3.4 General Connectivity and Expansion

microUSB 3.0 port

P8 & P9 cape header pins

mikroBUS shuttle connector

P8, P9, and mikroBUS helper circuitry

8.3.5 Buttons and LEDs

Boot select buttons

User LEDs and Power LED

Power and reset button

8.3.6 Wired and wireless connectivity

Ethernet

WiFi & Bluetooth

8.3.7 Memory, Media and Data storage

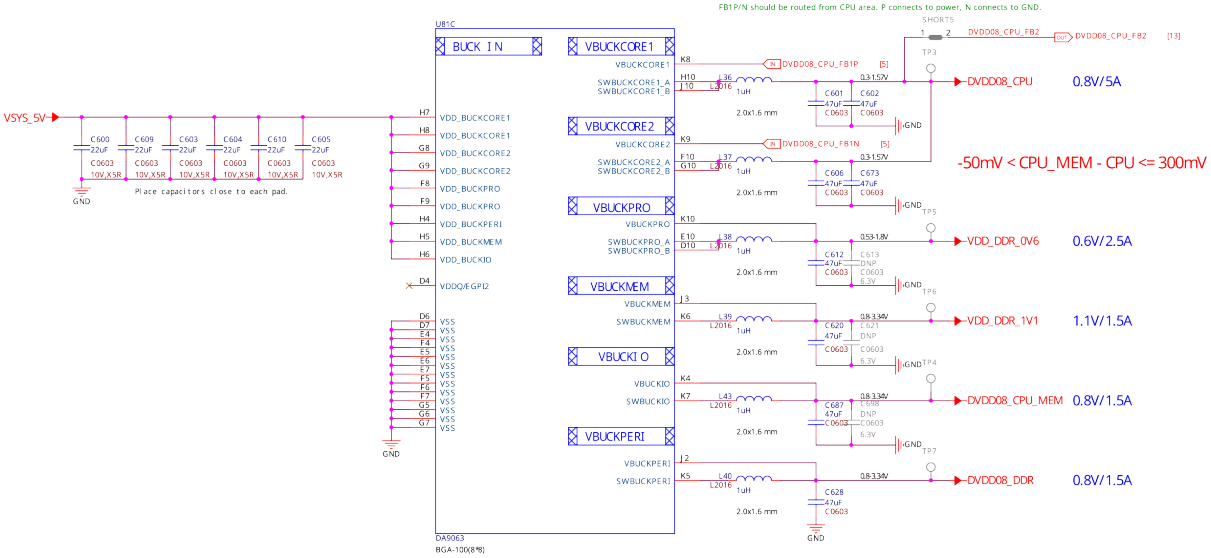


Fig. 8.27: PMIC Buck

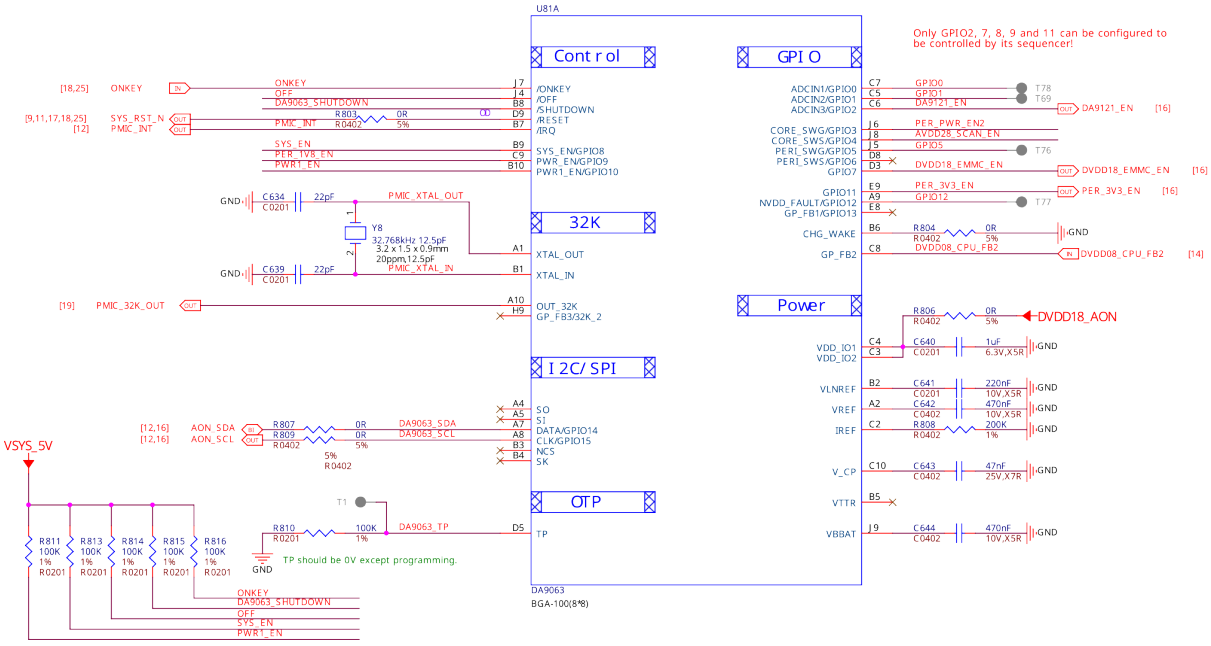


Fig. 8.28: PMIC Control

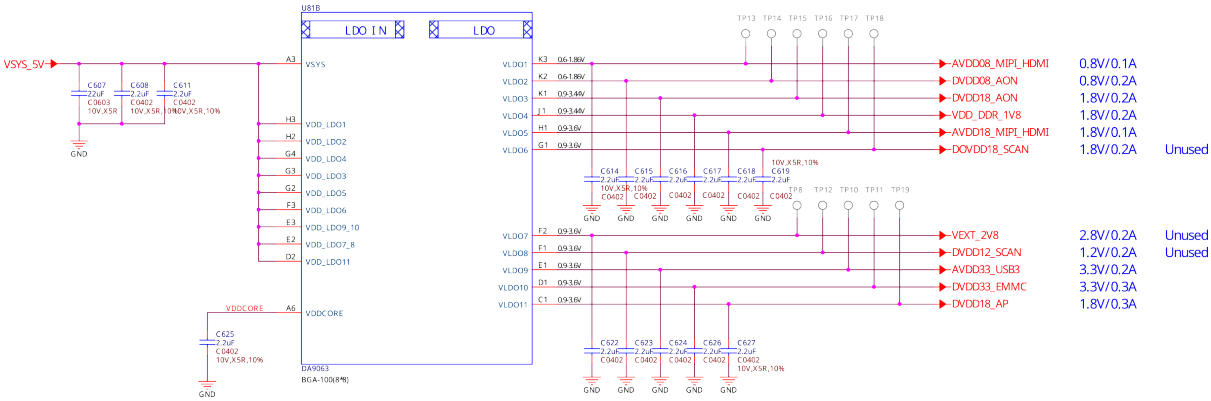


Fig. 8.29: PMIC LDO

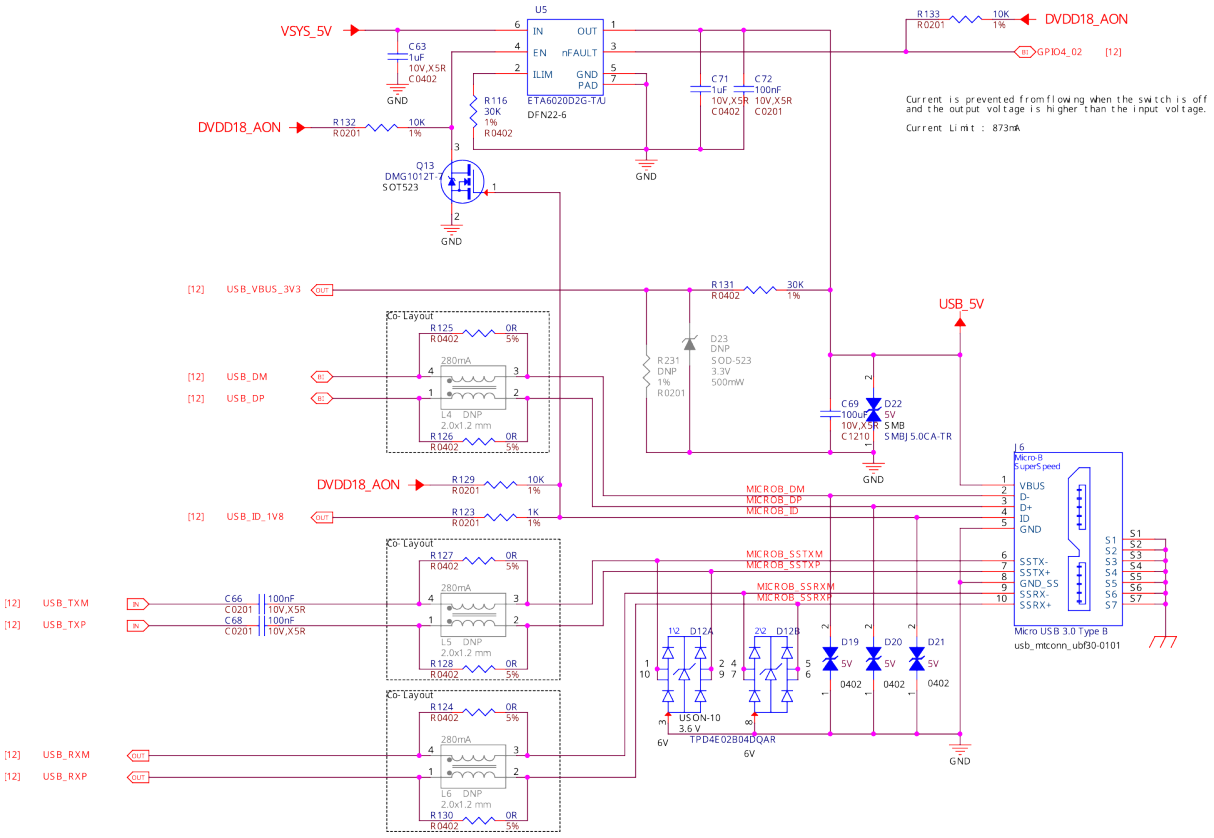


Fig. 8.30: microUSB 3.0 port

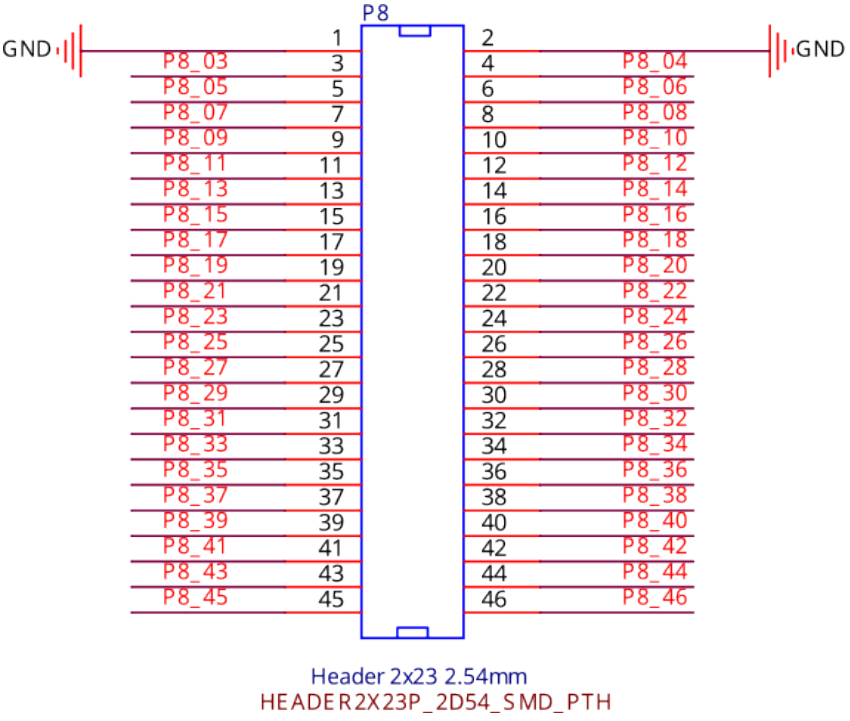


Fig. 8.31: P8 cape header

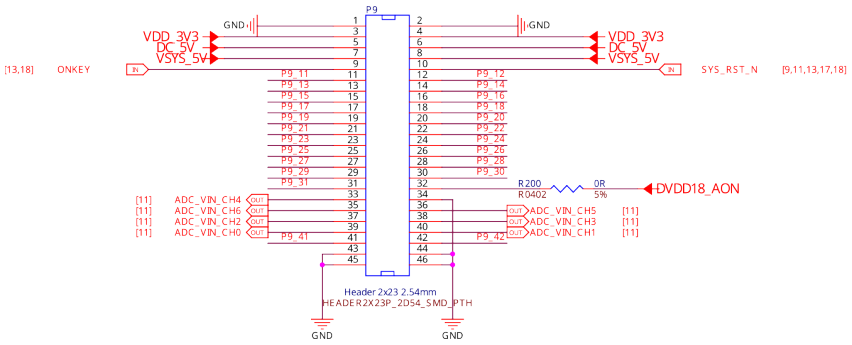
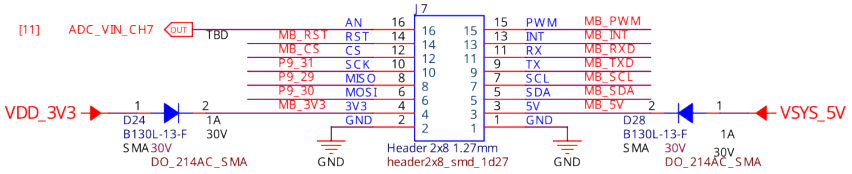
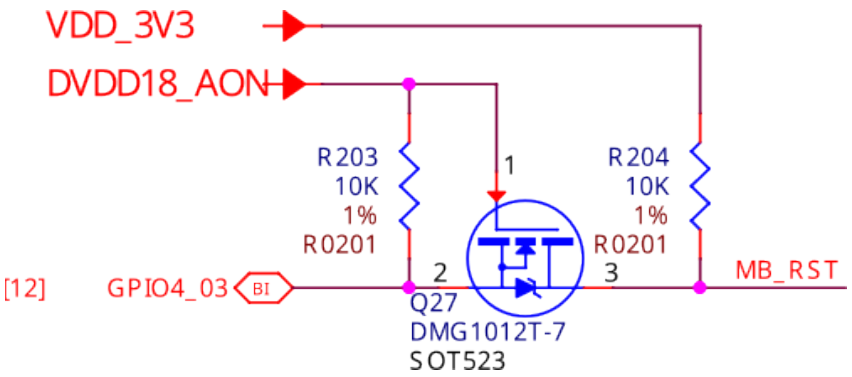


Fig. 8.32: P9 cape header





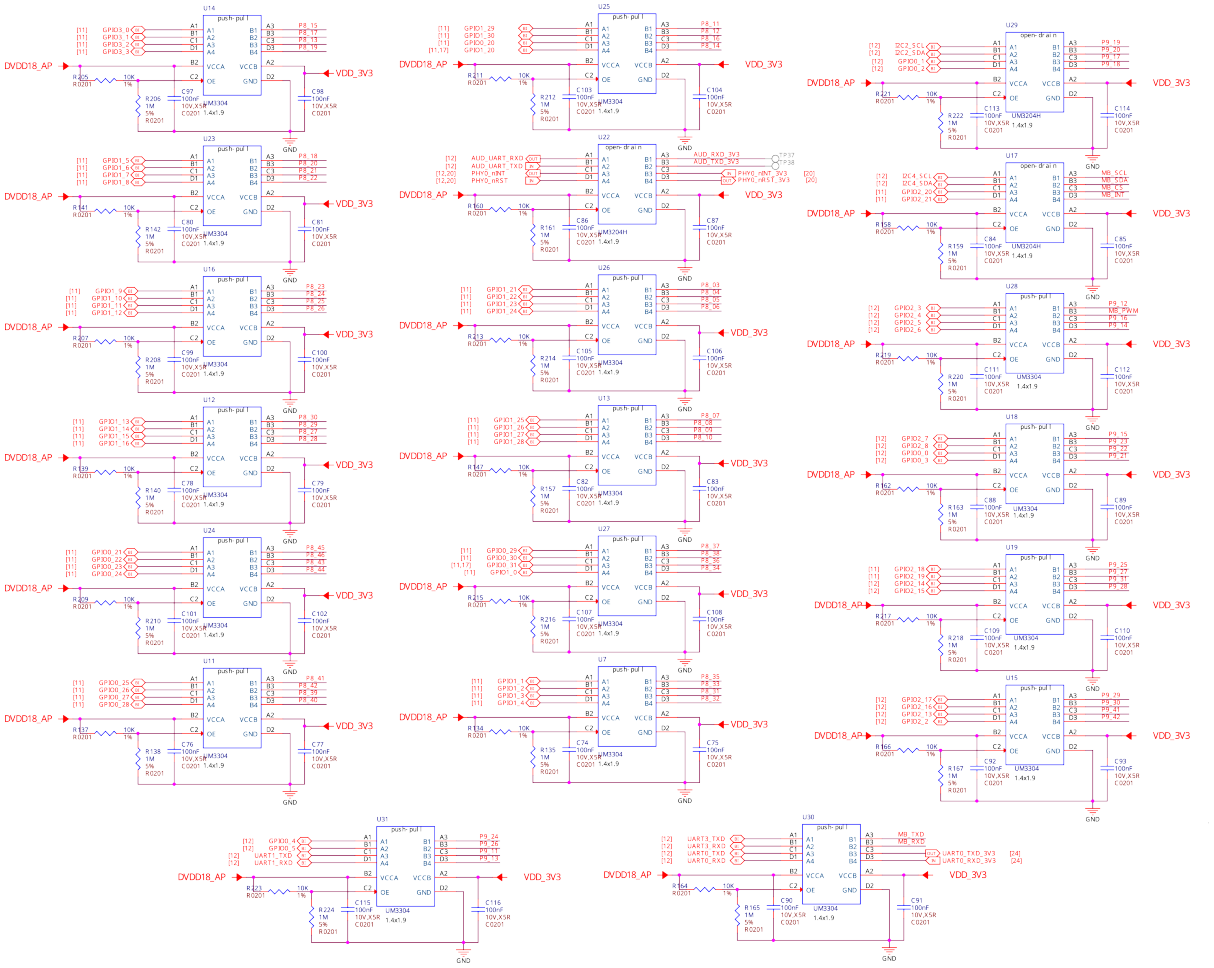
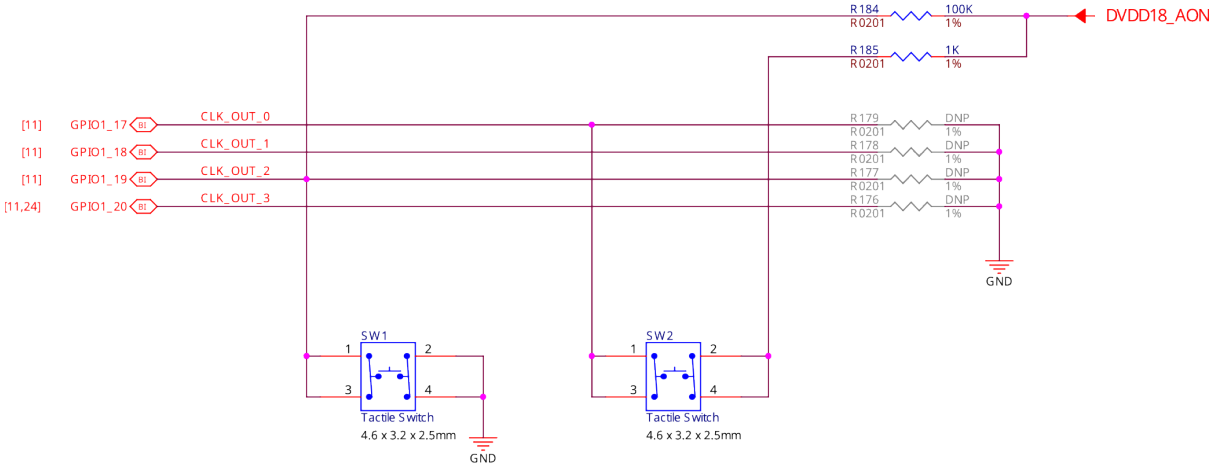


Fig. 8.33: P8, P9, and mikroBUS level shifters



CLK_OUT3	CLK_OUT2	CLK_OUT1	CLK_OUT0	Boot Mode Setting
X	0	X	X	USB Download
X	1	0	0	eMMC Boot
X	1	0	1	SD Boot, SDIO0
X	1	1	0	SPI NAND boot, QSPI0, CS0
X	1	1	1	SPI NOR Boot, QSPI0, CS0

Fig. 8.34: Boot select buttons

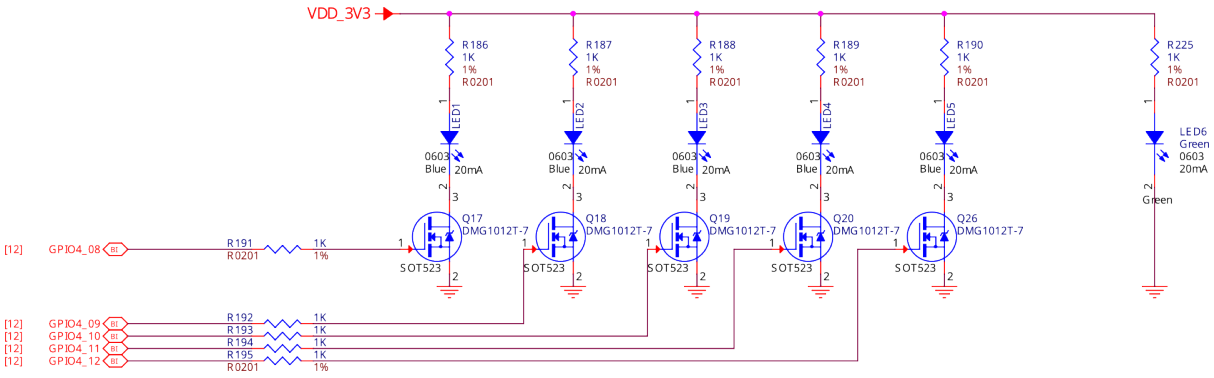


Fig. 8.35: User LEDs and power LED



Fig. 8.36: Power and reset button

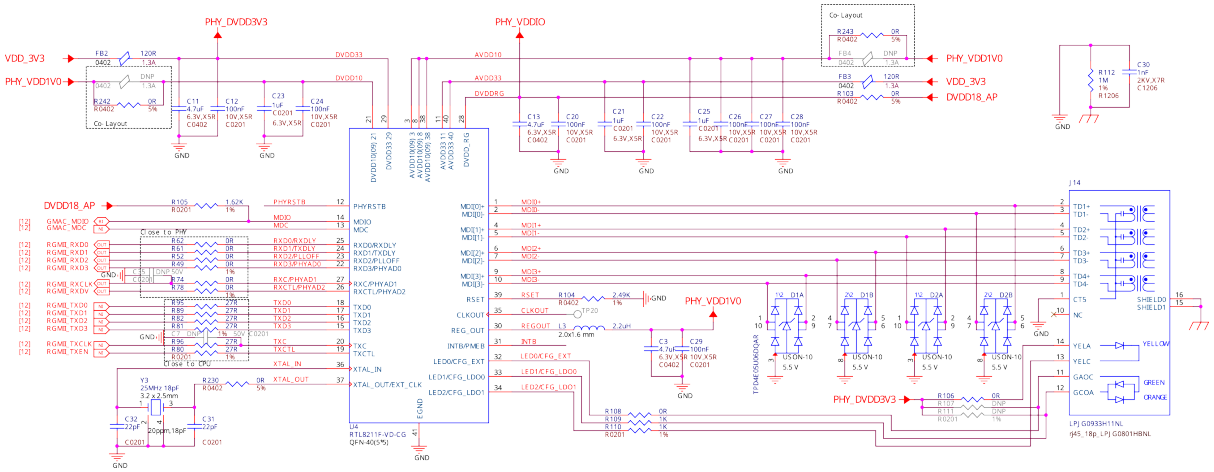


Fig. 8.37: Ethernet

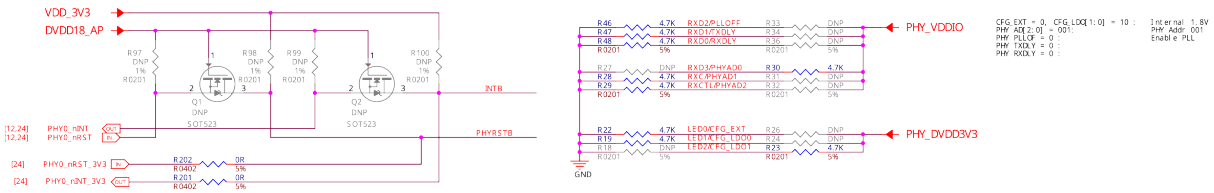


Fig. 8.38: Ethernet LevelShifter and Strapping

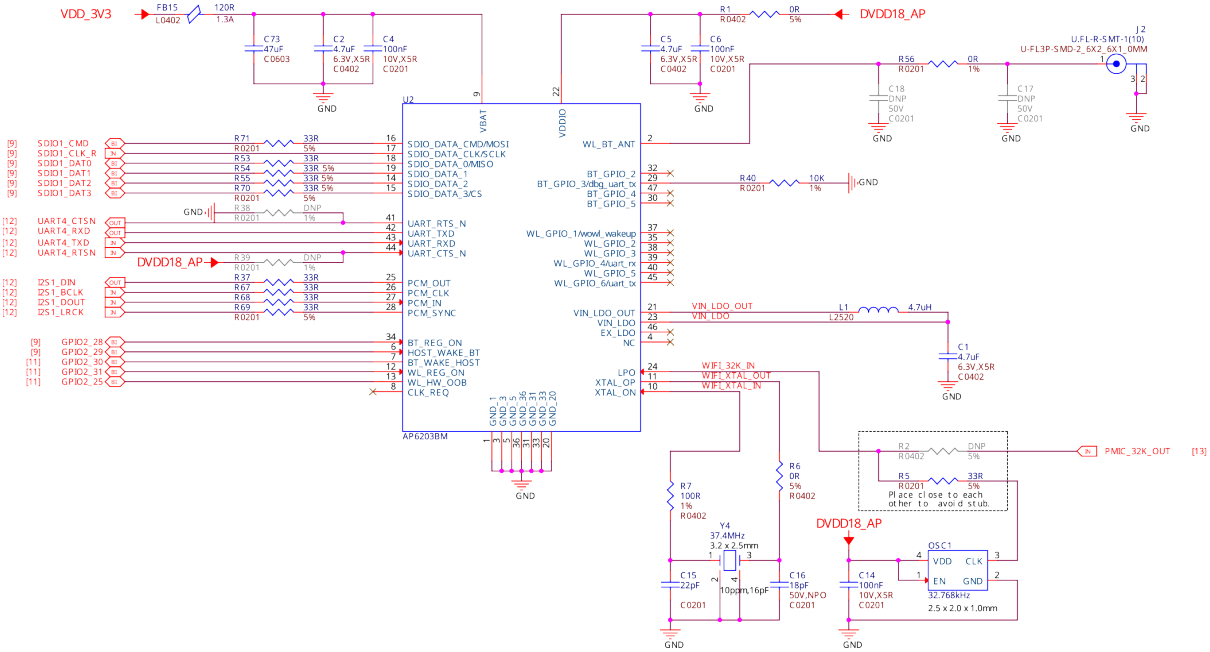


Fig. 8.39: WiFi and Bluetooth

DDR memory

eMMC

microSD

EEPROM

8.3.8 Multimedia I/O

CSI0

CSI1

DSI

CSI & DSI level shifter

HDMI

8.3.9 Debug

UART debug port

JTAG debug port

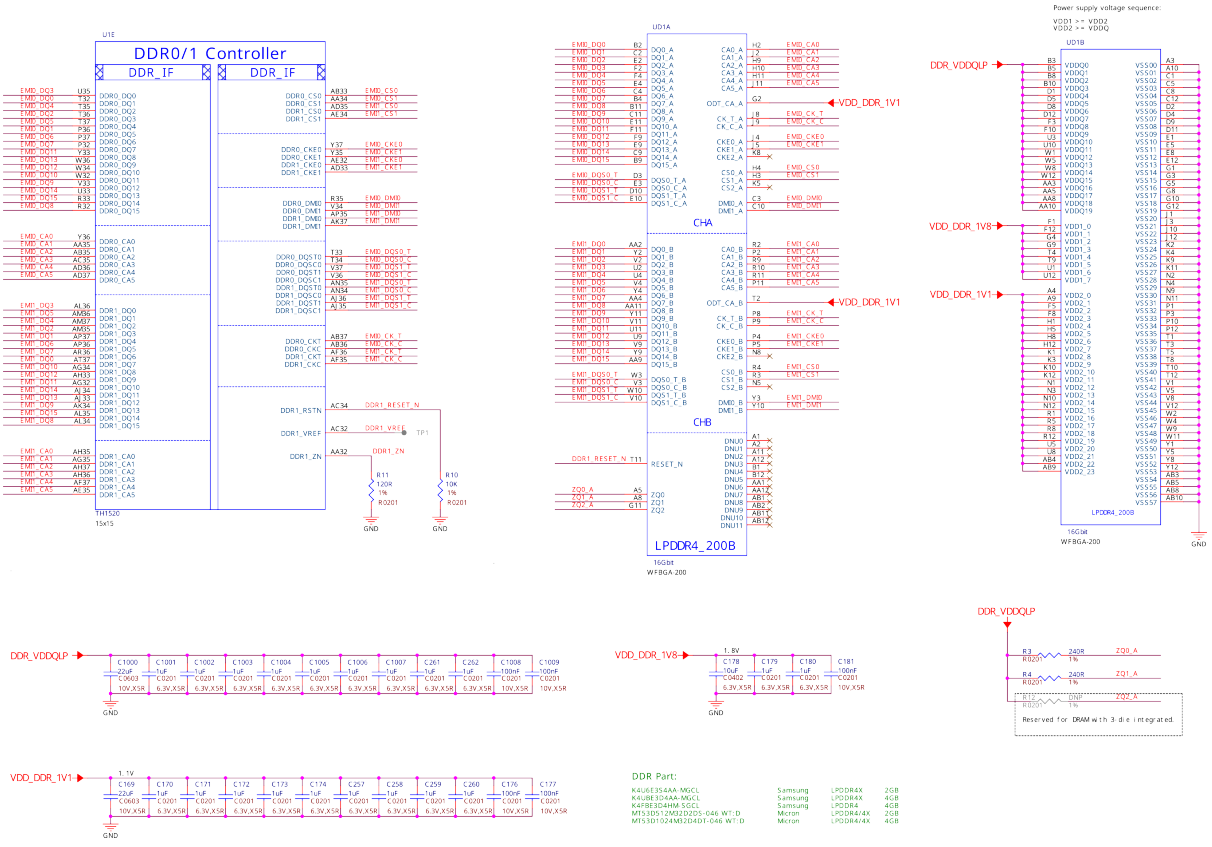


Fig. 8.40: 2GB DDR4 Memory chip1

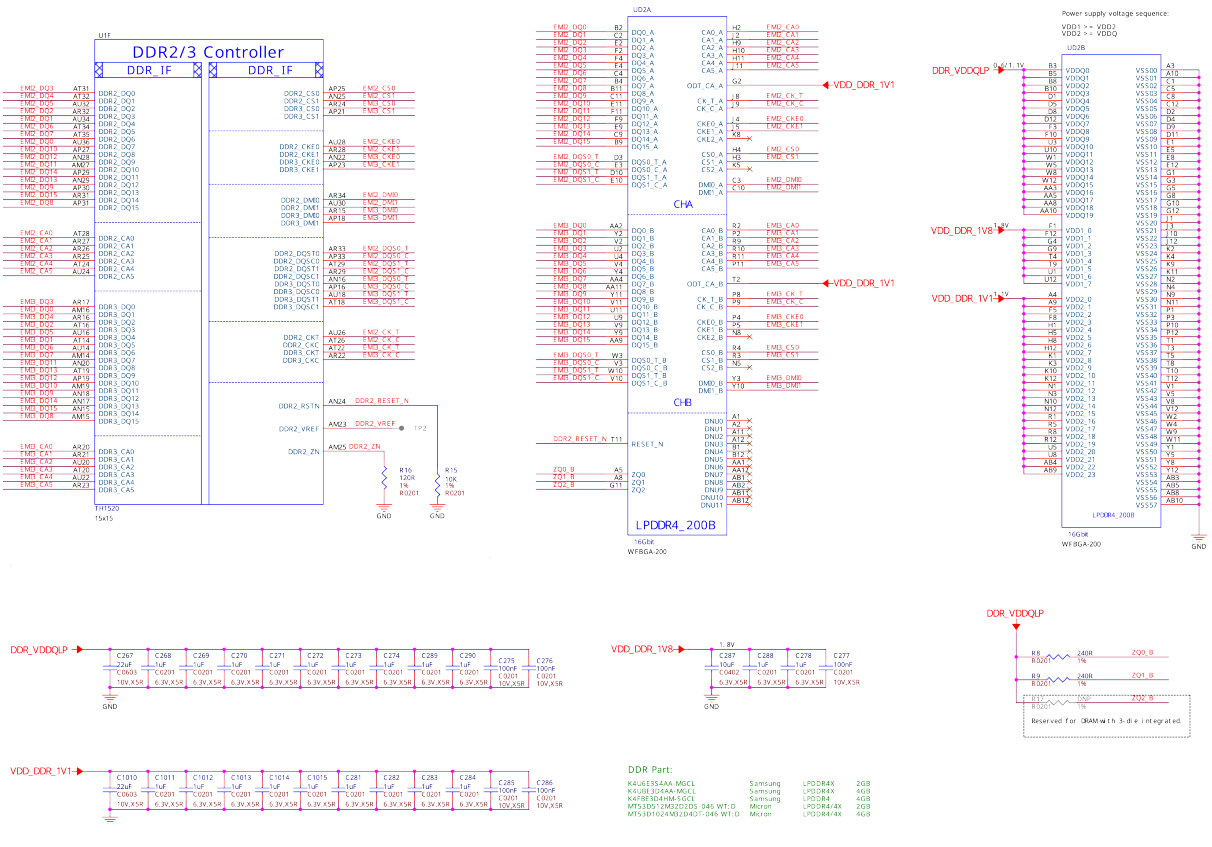


Fig. 8.41: 2GB DDR4 Memory chip2

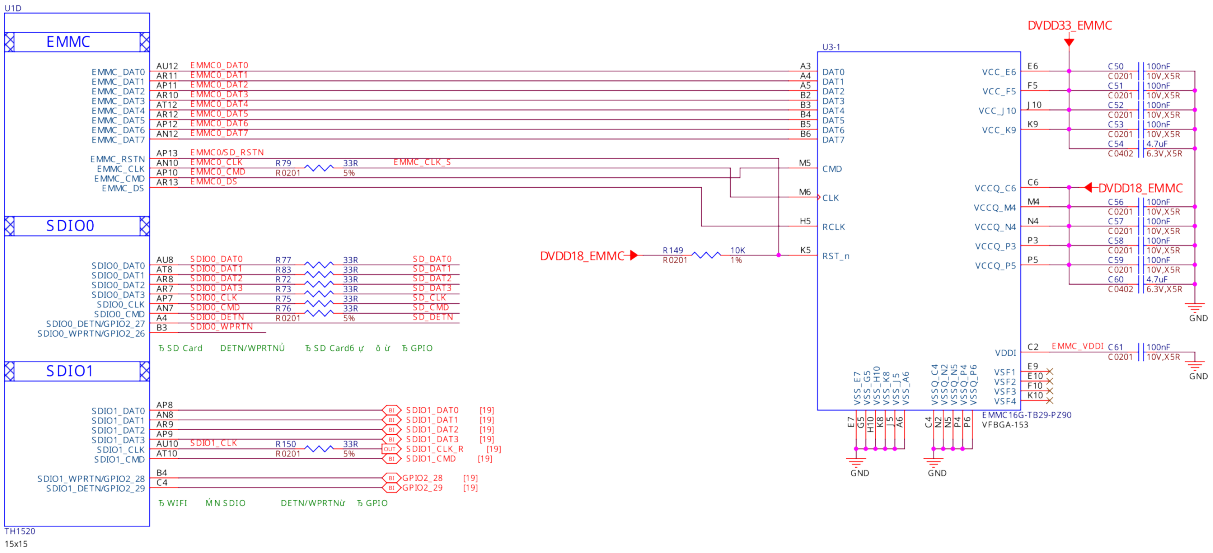


Fig. 8.42: 16GB eMMC

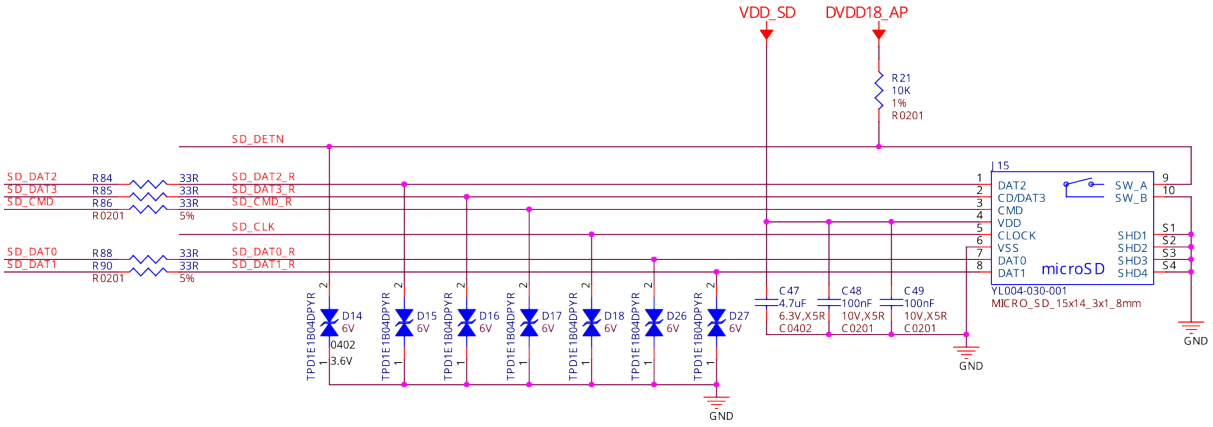


Fig. 8.43: microSD card connector

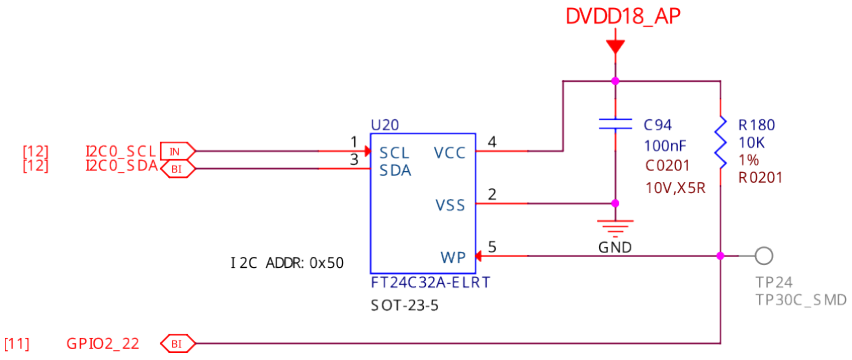


Fig. 8.44: 16GB EEPROM

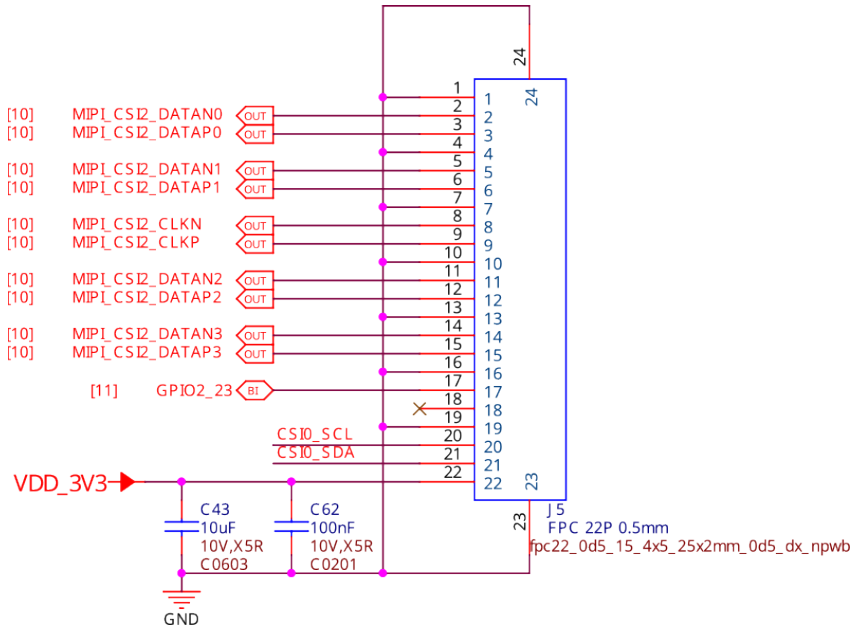


Fig. 8.45: CSI0 camera interface

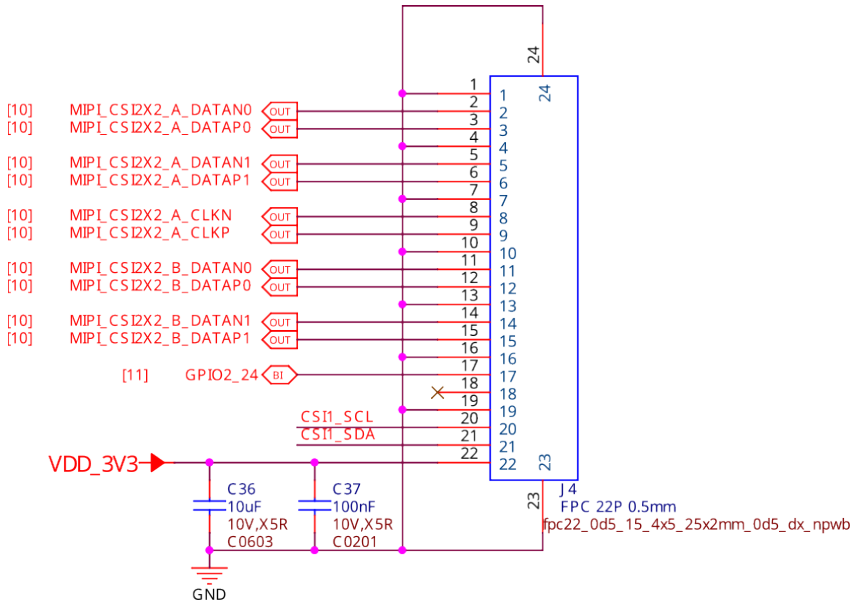


Fig. 8.46: CSI1 camera interface

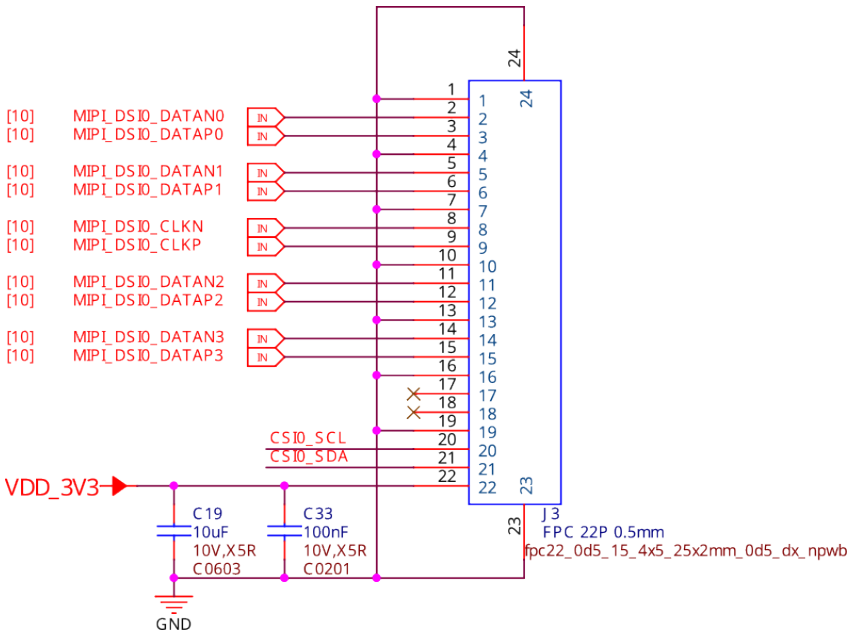


Fig. 8.47: DSI display interface

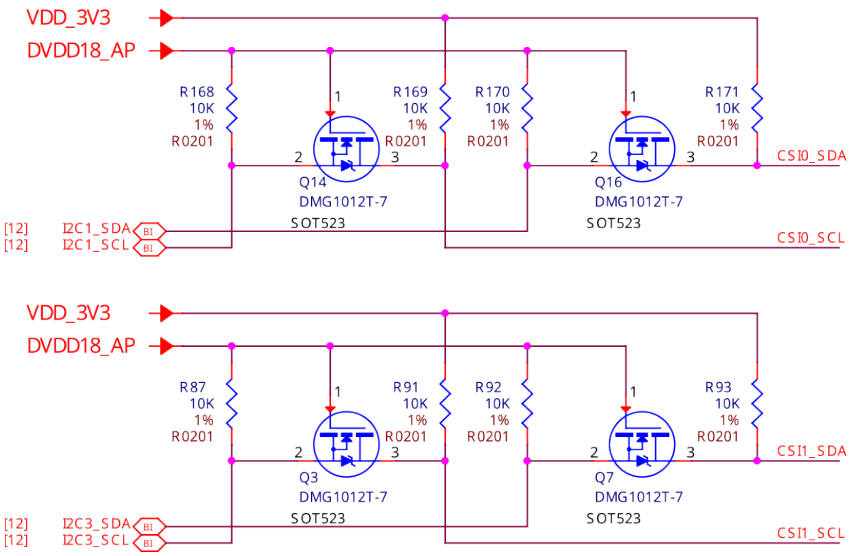


Fig. 8.48: CSI & DSI level shifter

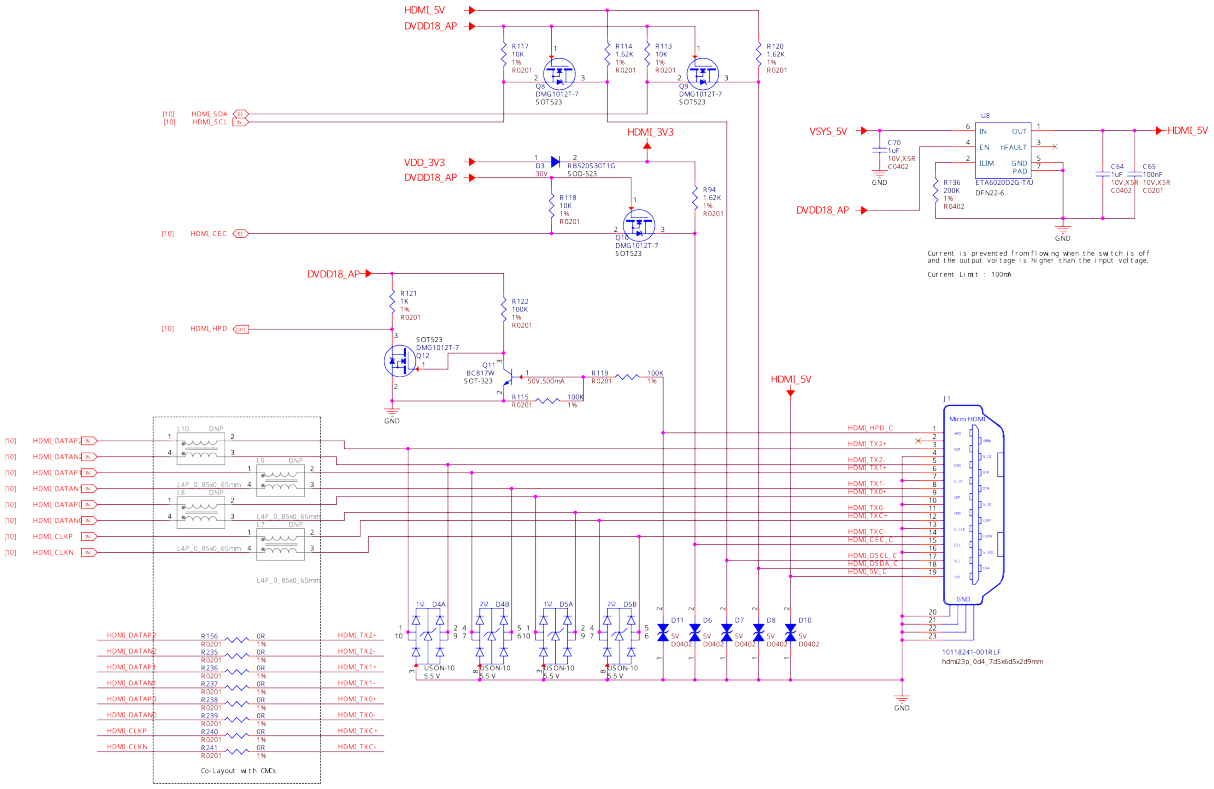


Fig. 8.49: HDMI display interface

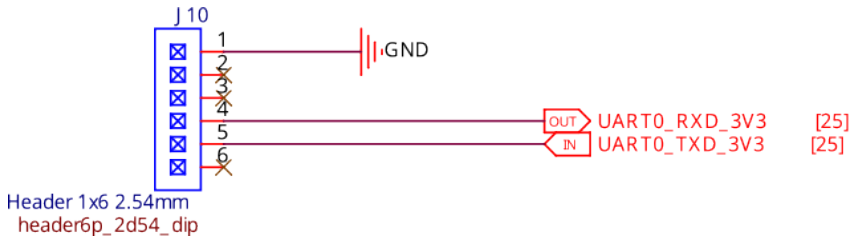


Fig. 8.50: UART Debug port

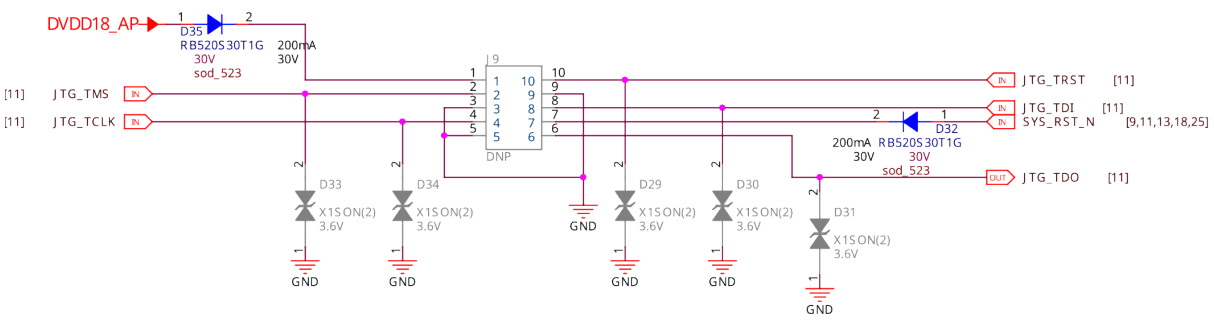
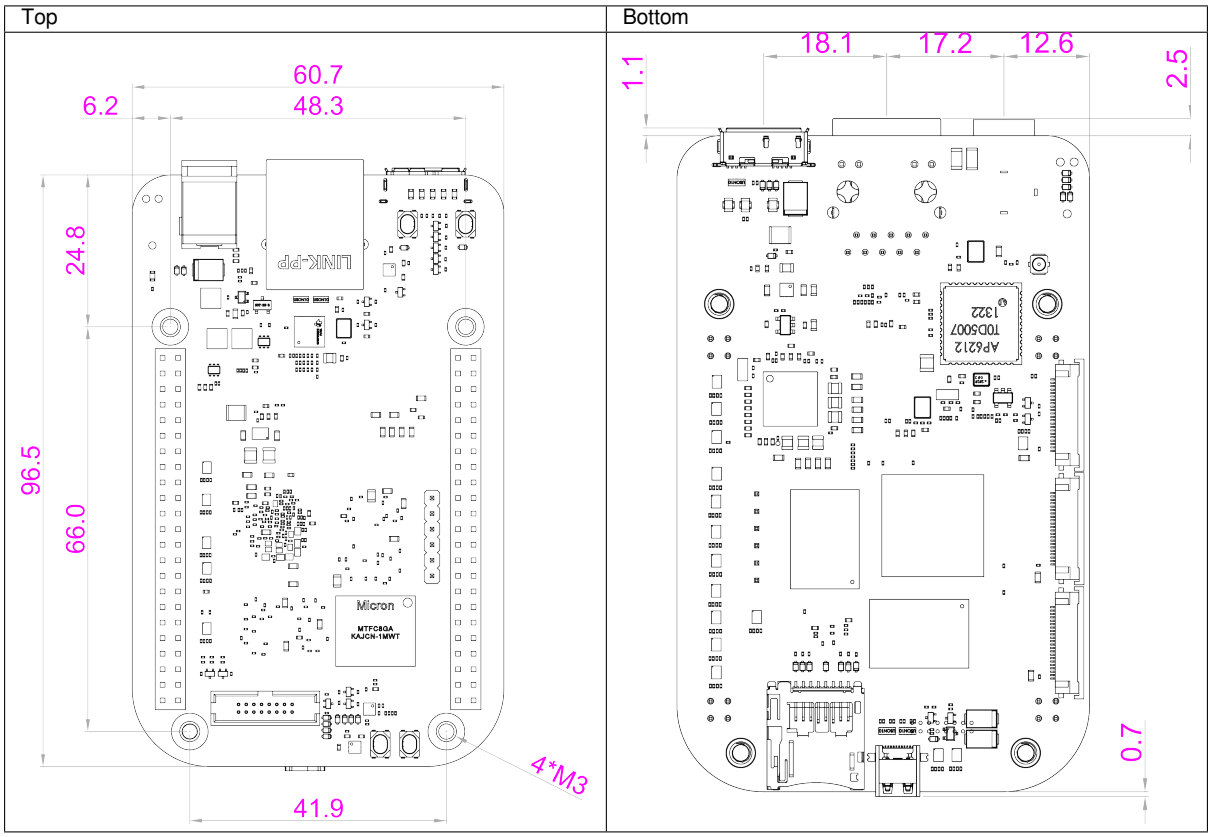


Fig. 8.51: JTAG debug port

8.3.10 Mechanical Specifications



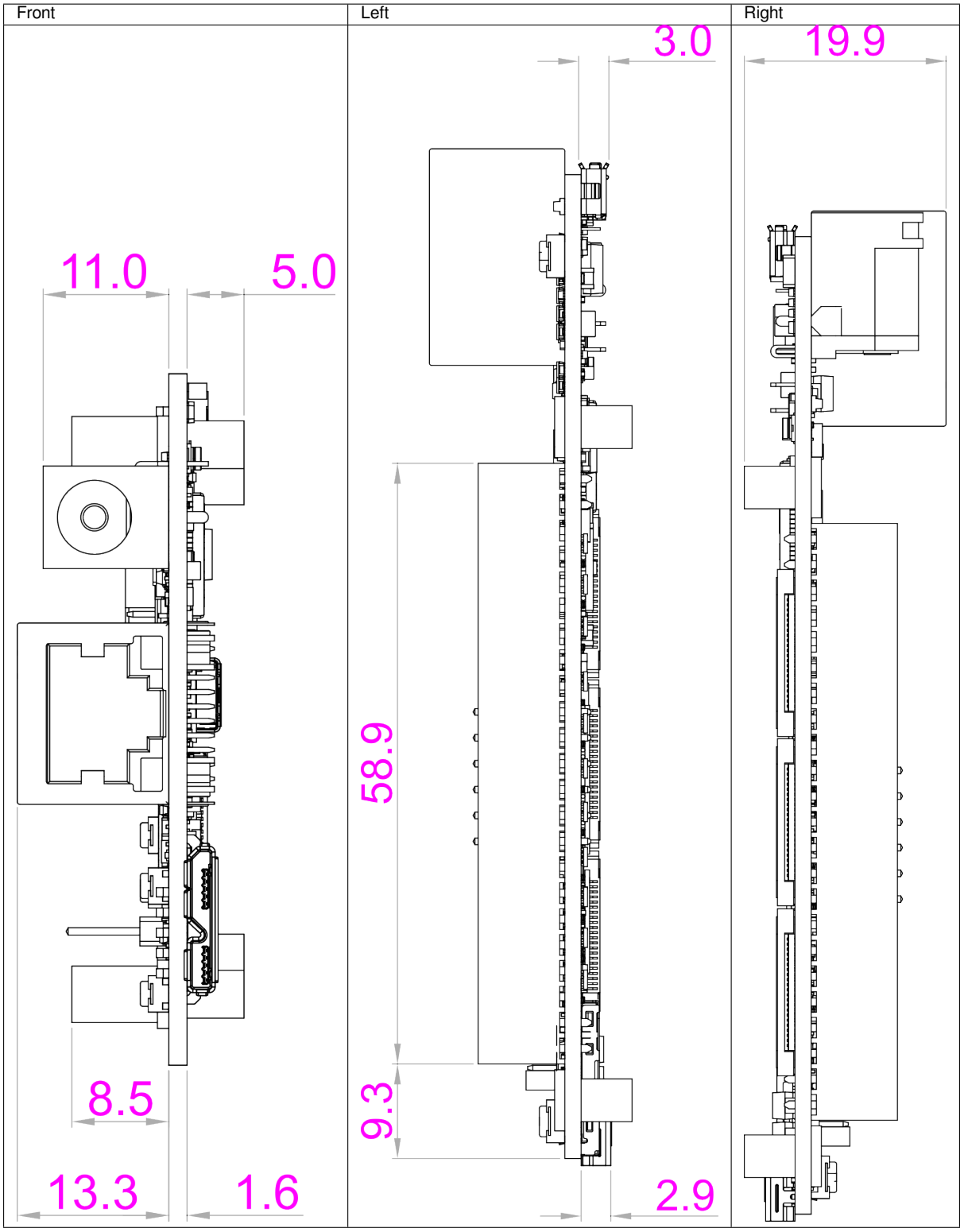

Front
Left
Right
11.0
5.0
8.5
13.3
1.6
3.0
58.9
9.3
2.9
19.9

Table 8.4: Dimensions & weight

Parameter	Values
Size	96.5×60.7×19.9mm
Max heigh	21.1mm
PCB Size	96.5x60.5*1.6mm
PCB Layers	10 layers
PCB Thickness	1.6mm
RoHS compliant	yes
Gross Weight	128.8g
Net weight	49.7g

8.4 Expansion

8.4.1 Cape Headers

The expansion interface on the board is comprised of two headers P8 (46 pin) & P9 (46 pin). All signals on the expansion headers are **3.3V** unless otherwise indicated.

Note: Do not connect 5V logic level signals to these pins or the board will be damaged.

Note: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

Connector P8

The following tables show the pinout of the **P8** expansion header. The SW is responsible for setting the default function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **GPIO** row is the expected gpio identifier number in the Linux kernel.

Each row includes the gpiochipX and pinY in the format of *X Y*. You can use these values to directly control the GPIO pins with the commands shown below.

```
# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset X Y=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset X Y=0

For Example:

+---------+----------+
| Pin     | P8.03    |
+=========+==========+
| GPIO    | 1 20     |
+---------+----------+

Use the commands below for controlling this pin (P8.03) where X = 1 and Y =␣
↪20
```

(continues on next page)

(continued from previous page)

```
# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset 1 20=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset 1 20=0
```

The **BALL** row is the pin number on the processor.

The **REG** row is the offset of the control register for the processor pin.

The **MODE #** rows are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

NOTES:

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

P8.01-P8.02

P8.01	P8.02
GND	GND

P8.03-P8.05

Pin	P8.03	P8.04	P8.05
Name	GPIO1_21	GPIO1_22	GPIO1_23
BALL	J34	J35	K32
GPIO	1 21	1 22	1 23
REG	GPIO1_21_MUX	GPIO1_22_MUX	GPIO1_23_MUX
Mode 0	GPIO1_21	GPIO1_22	GPIO1_23
MODE 1	~	~	~
MODE 2	ISP0_FL_TRIG	ISP0_FLASH_TRIG	ISP0_PRELIGHT_TRIG
MODE 3	GPIO1_21	GPIO1_22	GPIO1_23
MODE 4	~	~	~
MODE 5	~	~	~

P8.06-P8.09

Pin	P8.06	P8.07	P8.08	P8.09
Name	GPIO1_24	GPIO1_25	GPIO1_26	GPIO1_27
BALL	K33	K34	K35	K36
GPIO	1 24	1 25	1 26	1 27
REG	GPIO1_24_MUX	GPIO1_25_MUX	GPIO1_26_MUX	GPIO1_27_MUX
Mode 0	GPIO1_24	GPIO1_25	GPIO1_26	GPIO1_27
MODE 1	~	~	~	~
MODE 2	ISP0_SHUTTER_TRIG	ISP0_SHUTTER_OPEN	ISP1_FL_TRIG	ISP1_FLASH_TRIG
MODE 3	GPIO1_24	GPIO1_25	~	~
MODE 4	~	~	~	~
MODE 5	~	~	~	~

P8.10-P8.13

Pin	P8.10	P8.11	P8.12	P8.13
Name	GPIO1_28	GPIO1_29	GPIO1_30	GPIO3_2
BALL	K37	L32	L33	C6
GPIO	1 28	1 29	1 30	3 2
REG	GPIO1_28_MUX	GPIO1_29_MUX	GPIO1_30_MUX	GPIO3_2_MUX
MODE 0	GPIO1_28	GPIO1_29	GPIO1_30	GPIO3_2
MODE 1	~	~	~	PWM0
MODE 2	ISP1_PRELIGHT_TRIG	ISP1_SHUTTER_TRIG	ISP1_SHUTTER_OPEN	~
MODE 3	~	~	~	~
MODE 4	~	~	~	~
MODE 5	~	~	~	~

P8.14-P8.16

Pin	P8.14	P8.15	P8.16
Name	CLK_OUT_3	GPIO3_0	GPIO0_20
BALL	E29	A6	F34
GPIO	1 20	3 0	0 20
REG	CLK_OUT_3_MUX	GPIO3_0_MUX	GPIO0_20_MUX
MODE 0	BOOT_SEL3	GPIO3_0	GPIO0_20
MODE 1	CLK_OUT_3	GMAC1_RXD2	UART3_TXD
MODE 2	~	~	UART3_IR_OUT
MODE 3	GPIO1_20	~	~
MODE 4	~	~	~
MODE 5	~	~	~

P8.17-P8.19

Pin	P8.17	P8.18	P8.19
Name	GPIO3_1	GPIO1_5	GPIO3_3
BALL	B6	B34	D6
GPIO	3 1	1 5	3 3
REG	GPIO3_1_MUX	GPIO1_5_MUX	GPIO3_3_MUX
MODE 0	GPIO3_1	GPIO1_5	GPIO3_3
MODE 1	GMAC1_RXD3	~	PWM1
MODE 2	~	~	~
MODE 3	~	~	~
MODE 4	~	DPU_COLOR_16	~
MODE 5	~	DPU1_COLOR_16	~

P8.20-P8.22

Pin	P8.20	P8.21	P8.22
Name	GPIO1_6	GPIO1_7	GPIO1_8
BALL	C34	D34	B35
GPIO	1 6	1 7	1 8
REG	GPIO1_6_MUX	GPIO1_7_MUX	GPIO1_8_MUX
MODE 0	GPIO1_6	GPIO1_7	GPIO1_8
MODE 1	~	QSPI1_SCLK	QSPI1_SSN0
MODE 2	~	~	~
MODE 3	~	~	~
MODE 4	DPU_COLOR_17	DPU_COLOR_18	DPU_COLOR_19
MODE 5	DPU1_COLOR_17	DPU1_COLOR_18	DPU1_COLOR_19

P8.23-P8.26

Pin	P8.23	P8.24	P8.25	P8.26
Name	GPIO1_9	GPIO1_10	GPIO1_11	GPIO1_12
BALL	A36	B36	B37	C36
GPIO	1 9	1 10	1 11	1 12
REG	GPIO1_9_MUX	GPIO1_10_MUX	GPIO1_11_MUX	GPIO1_12_MUX
MODE 0	GPIO1_9	GPIO1_10	GPIO1_11	GPIO1_12
MODE 1	QSPI1_M0_MOSI	QSPI1_M1_MISO	QSPI1_M2_WP	QSPI1_M3_HOLD
MODE 2	~	~	~	~
MODE 3	~	~	~	~
MODE 4	DPU_COLOR_20	DPU_COLOR_21	DPU_COLOR_22	DPU_COLOR_23
MODE 5	DPU1_COLOR_20	DPU1_COLOR_21	DPU1_COLOR_22	DPU1_COLOR_23

P8.27-P8.29

Pin	P8.27	P8.28	P8.29
Name	GPIO1_15	GPIO1_16	GPIO1_14
BALL	D37	E34	D36
GPIO	1 15	1 16	1 14
REG	GPIO1_15_MUX	GPIO1_16_MUX	GPIO1_14_MUX
MODE 0	GPIO1_15	GPIO1_16	GPIO1_14
MODE 1	UART4_CTSN	UART4_RTSN	UART4_RXD
MODE 2	~	~	~
MODE 3	~	~	~
MODE 4	DPU_VSYNC	DPU_PIXELCLK	DPU_HSYNC
MODE 5	DPU1_VSYNC	DPU1_PIXELCLK	DPU1_HSYNC

P8.30-P8.32

Pin	P8.30	P8.31	P8.32
Name	GPIO1_13	GPIO1_3	GPIO1_4
BALL	D35	D33	A34
GPIO	1 13	1 3	1 4
REG	GPIO1_13_MUX	GPIO1_3_MUX	GPIO1_4_MUX
MODE 0	GPIO1_13	GPIO1_3	GPIO1_4
MODE 1	UART4_TXD	DSP1_JTG_TDO	DSP1_JTG_TCLK
MODE 2	~	~	~
MODE 3	~	~	~
MODE 4	DPU_COLOR_EN	DPU_COLOR_14	DPU_COLOR_15
MODE 5	DPU1_COLOR_EN	DPU1_COLOR_14	DPU1_COLOR_15

P8.33-P8.35

Pin	P8.33	P8.34	P8.35
Name	GPIO1_2	GPIO1_0	GPIO1_1
BALL	C33	E32	A32
GPIO	1 2	1 0	1 1
REG	GPIO1_2_MUX	GPIO1_0_MUX	GPIO1_1_MUX
MODE 0	GPIO1_2	GPIO1_0	GPIO1_1
MODE 1	DSP1_JTG_TDI	DSP1_JTG_TRST	DSP1_JTG_TMS
MODE 2	~	~	~
MODE 3	~	~	~
MODE 4	DPU_COLOR_13	DPU_COLOR_11	DPU_COLOR_12
MODE 5	DPU1_COLOR_13	DPU1_COLOR_11	DPU1_COLOR_12

P8.36-P8.38

Pin	P8.36	P8.37	P8.38
Name	GPIO0_31	GPIO0_29	GPIO0_30
BALL	D32	B32	C32
GPIO	0 31	0 29	0 30
REG	GPIO0_31_MUX	GPIO0_29_MUX	GPIO0_30_MUX
MODE 0	GPIO0_31	GPIO0_29	GPIO0_30
MODE 1	~	~	~
MODE 2	~	~	~
MODE 3	~	~	~
MODE 4	DPU_COLOR_10	DPU_COLOR_8	DPU_COLOR_9
MODE 5	DPU1_COLOR_10	DPU1_COLOR_8	DPU1_COLOR_9

P8.39-P8.41

Pin	P8.39	P8.40	P8.41
Name	GPIO0_27	GPIO0_28	GPIO0_25
BALL	D31	E31	F30
GPIO	0 27	0 28	0 25
REG	GPIO0_27_MUX	GPIO0_28_MUX	GPIO0_25_MUX
MODE 0	GPIO0_27	GPIO0_28	GPIO0_25
MODE 1	~	~	DSP0_JTG_TDO
MODE 2	I2C1_SCL	I2C1_SDA	~
MODE 3	~	~	~
MODE 4	DPU_COLOR_6	DPU_COLOR_7	DPU_COLOR_4
MODE 5	DPU1_COLOR_6	DPU1_COLOR_7	DPU1_COLOR_4

P8.42-P8.44

Pin	P8.42	P8.43	P8.44
Name	GPIO0_26	GPIO0_23	GPIO0_24
BALL	C31	C30	D30
GPIO	0 26	0 23	0 24
REG	GPIO0_26_MUX	GPIO0_23_MUX	GPIO0_24_MUX
MODE 0	GPIO0_26	GPIO0_23	GPIO0_24
MODE 1	DSP0_JTG_TCLK	DSP0_JTG_TMS	DSP0_JTG_TDI
MODE 2	~	I2C4_SDA	QSPI1_SSN1
MODE 3	~	~	~
MODE 4	DPU_COLOR_5	DPU_COLOR_2	DPU_COLOR_3
MODE 5	DPU1_COLOR_5	DPU1_COLOR_2	DPU1_COLOR_3

P8.45-P8.46

Pin	P8.45	P8.46
Name	GPIO0_21	GPIO0_22
BALL	F36	D29
GPIO	0 21	0 22
REG	GPIO0_21_MUX	GPIO0_22_MUX
MODE 0	GPIO0_21	GPIO0_22
MODE 1	UART3_RXD	DSP0_JTG_TRST
MODE 2	UART3_IR_IN	I2C4_SCL
MODE 3	~	~
MODE 4	DPU_COLOR_0	DPU_COLOR_1
MODE 5	DPU1_COLOR_0	DPU1_COLOR_1

Connector P9

The following tables show the pinout of the **P9** expansion header. The SW is responsible for setting the default function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **GPIO** row is the expected gpio identifier number in the Linux kernel.

Each row includes the gpiochipX and pinY in the format of *X Y*. You can use these values to direcly control the GPIO pins with the commands shown below.

```
# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset X Y=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset X Y=0

For Example:

+---------+----------+
| Pin     | P9.11    |
+=========+==========+
| GPIO    | 1 1      |
+---------+----------+

Use the commands below for controlling this pin (P9.11) where X = 1 and Y = 1

# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset 1 20=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset 1 20=0
```

The **BALL** row is the pin number on the processor.

The **REG** row is the offset of the control register for the processor pin.

The **MODE #** rows are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

If included, the **2nd BALL** row is the pin number on the processor for a second processor pin connected to the same pin on the expansion header. Similarly, all row headings starting with **2nd** refer to data for this second processor pin.

NOTES:

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

P9.01-P9.05

P9.01	P9.02	P9.03	P9.04	P9.05
GND	GND	VOUT_3V3	VOUT_3V3	VIN

P9.06-P9.10

P9.06	P9.07	P9.08	P9.09	P9.10
VIN	VOUT_SYS	VOUT_SYS	ONKEY#	RESET#

P9.11-P9.13

Pin	P9.11	P9.12	P9.13
Name	UART1_TXD	QSPI0_CSN0	UART1_RXD
BALL	M32	H1	M33
GPIO	0 10	2 3	0 11
REG	UART1_TXD_MUX	QSPI0_CSN0_MUX	UART1_RXD_MUX
MODE 0	UART1_TXD	QSPI0_SSN0	UART1_RXD
MODE 1	~	PWM1	~
MODE 2	~	I2S_SDA1	~
MODE 3	GPIO0_10	GPIO2_3	GPIO0_11
MODE 4	~	~	~
MODE 5	~	~	~

P9.14-P9.16

Pin	P9.14	P9.15	P9.16
Name	QSPI0_D1_MISO	QSPI0_D2_WP	QSPI0_D0_MOSI
BALL	K3	K2	J3
GPIO	2 6	2 7	2 5
REG	QSPI0_D1_MISO_MUX	QSPI0_D2_WP_MUX	QSPI0_D0_MOSI_MUX
MODE 0	QSPI0_M1_MISO	QSPI0_M2_WP	QSPI0_M0_MOSI
MODE 1	PWM4	PWM5	PWM3
MODE 2	I2S_MCLK	I2S_SCLK	I2S_SDA3
MODE 3	GPIO2_6	GPIO2_7	GPIO2_5
MODE 4	~	~	~
MODE 5	~	~	~

P9.17-P9.19

Pin	P9.17	P9.18	P9.19
Name	QSPI1_CSN0	QSPI1_D0_MOSI	I2C2_SCL
BALL	H32	G35	G4
GPIO	0 1	0 2	2 9
REG	QSPI1_CSN0_MUX	QSPI1_D0_MOSI_MUX	I2C2_SCL_MUX
MODE 0	QSPI1_SSN0	QSPI1_M0_MOSI	I2C2_SCL
MODE 1	~	ISO7816_CVCC_EN	UART2_TXD
MODE 2	I2S_MCLK	I2C5_SDA	~
MODE 3	GPIO0_1	GPIO0_2	GPIO2_9
MODE 4	EFUSE_SPI_NSS	EFUSE_SPI_SI	~
MODE 5	~	~	~

P9.20-P9.22

Pin	P9.20	P9.21	P9.22
Name	I2C2_SDA	QSPI1_D1_MISO	QSPI1_SCLK
BALL	G3	G34	H34
GPIO	2 10	0 3	0 0
REG	I2C2_SDA_MUX	QSPI1_D1_MISO_MUX	QSPI1_SCLK_MUX
MODE 0	I2C2_SDA	QSPI1_M1_MISO	QSPI1_SCLK
MODE 1	UART2_RXD	ISO7816_CLK	ISO7816_DET
MODE 2	~	~	~
MODE 3	GPIO2_10	GPIO0_3	GPIO0_0
MODE 4	~	EFUSE_SPI_SO	EFUSE_SPI_CLK
MODE 5	~	~	~

P9.23-P9.25

Pin	P9.23	P9.24	P9.25
Name	QSPI0_D3_HOLD	QSPI1_D2_WP	GPIO2_18
BALL	K1	G33	F5
GPIO	2 8	0 4	2 18
REG	QSPI0_D3_HOLD_MUX	QSPI1_D2_WP_MUX	GPIO2_18_MUX
MODE 0	QSPI0_M3_HOLD	QSPI1_M2_WP	GPIO2_18
MODE 1	~	ISO7816_RST	GMAC1_TX_CLK
MODE 2	I2S_WS	UART5_TXD	~
MODE 3	GPIO2_8	GPIO0_4	~
MODE 4	~	EFUSE_BUSY	~
MODE 5	~	~	~

P9.26-P9.28

Pin	P9.26	P9.27	P9.28
Name	QSPI1_D3_HOLD	GPIO2_19	SPI_CSN
BALL	F37	E4	E3
GPIO	0 5	2 19	2 15
REG	QSPI1_D3_HOLD_MUX	GPIO2_19_MUX	SPI_CSN_MUX
MODE 0	QSPI1_M3_HOLD	GPIO2_19	SPI_SSN0
MODE 1	ISO7816_DAT	GMAC1_RX_CLK	UART2_RXD
MODE 2	UART5_RXD	~	UART2_IR_IN
MODE 3	GPIO0_5	~	GPIO2_15
MODE 4	~	~	~
MODE 5	~	~	~

P9.29-P9.31

Pin	P9.29	P9.30	P9.31
Name	SPI_MISO	SPI_MOSI	SPI_SCLK
BALL	F1	F2	D3
GPIO	2 17	2 16	2 14
REG	SPI_MISO_MUX	SPI_MOSI_MUX	SPI_SCLK_MUX
MODE 0	SPI_MISO	SPI_MOSI	SPI_SCLK
MODE 1	~	~	UART2_TXD
MODE 2	~	~	UART2_IR_OUT
MODE 3	GPIO2_17	GPIO2_16	GPIO2_14
MODE 4	~	~	~
MODE 5	~	~	~

P9.32-P9.40

P9.32	P9.34
VDD_ADC	GND

P9.33	P9.35	P9.36	P9.37	P9.38	P9.39	P9.40
ADC_VIN_CH4	ADC_VIN_CH6	ADC_VIN_CH5	ADC_VIN_CH2	ADC_VIN_CH3	ADC_VIN_CH0	ADC_VIN_CH1

P9.41-P9.42

Pin	P9.41	P9.42
Name	GPIO2_13	QSPI0_SCLK
BALL	D2	H3
GPIO	2 13	2 2
REG	GPIO2_13_MUX	QSPI0_SCLK_MUX
MODE 0	GPIO2_13	QSPI0_SCLK
MODE 1	SPI_SSN1	PWM0
MODE 2	~	I2S_SDA0
MODE 3	~	GPIO2_2
MODE 4	~	~
MODE 5	~	~

P9.43-P9.46

P9.43	P9.44	P9.45	P9.46
GND	GND	GND	GND

mikroBUS

Pin	mikroBUS port		Pin
ADC_VIN_CH7	**AN**	**PWM**	QSPI0_CSN1 (MODE1:PWM2)
AUDIO_PA3 (MODE3:GPIO4_3)	**RST**	**INT**	GPIO2_21 (MODE0:GPIO2_21)
GPIO2_20 (MODE0:GPIO2_20)	**CS**	**RX**	UART3_RXD (MODE1:UART3_RXD)
SPI_SCLK (MODE0:SPI_SCLK)	**SCK**	**TX**	UART3_TXD (MODE1:UART3_TXD)
SPI_MISO (MODE0:SPI_MISO)	**MISO**	**SCL**	GPIO0_18 (MODE1:I2C4_SCL)
SPI_MOSI (MODE0:SPI_MOSI)	**MOSI**	**SDA**	GPIO0_19 (MODE1:I2C4_SDA)
3.3V supply	**3V3**	**5V**	5V supply
Ground	**GND**	**GND**	Ground

8.5 Demos

Todo: We need a CSI capture and DSI display demos

Todo: We need a cape compatibility layer demo

Important: This document is a work on progress.

8.5.1 Using CSI Cameras

Note: CSI support is only available in Yocto image for BeagleV Ahead, to flash latest Yoctor image on your BeagleV Ahead you can checkout *Flashing eMMC* section.

Hardware

IMX219 camera modules has been tested to work well with BeagleV Ahead, some of them are listed below:

1. Raspberry Pi Camera Board v2 - 8 Megapixels (Adafruit)
2. Raspberry Pi NoIR Camera Board v2 - 8 Megapixels (Adafuit)
3. Arducam IMX219 (Robu.in)

In addition to the camer you'll need a 15pin to 22pin cable as well:

1. Raspberry Pi Zero FPC Camera Cable (Adafruit)
2. Raspberry Pi Zero v1.3 Camera Cable (Adafuit)
3. Raspberry Pi Zero V1.3 Camera Cable (Robu.in)

Software

There are several demo applications available for testing CSI, execute commands below to test your IMX219 camera on CSI0 & CSI1 ports:

1. Change directory to demo application location using: `cd /usr/share/vi/isp/test`
2. Set environment variable `export ISP_LOG_LEVEL=3`
3. To test CSI0 execute: `./camera_demo1 2 0 1 0 1920 1080 1 30 7`
4. To test CSI1 execute: `./camera_demo1 0 0 1 0 1920 1080 1 30 7`

When you execure *camera_demo1* then you should see something like this on your console:

```
...
...
IMX219: IMX219_IsiExposureControlIss: g=168.960999, Ti=0.050000
CAMERIC-MI-IRQ: isp mi frame out (59)  fps[0]: 19.74
IMX219: IMX219_IsiExposureControlIss: g=168.960999, Ti=0.050000
CAMERIC-MI-IRQ: isp mi frame out (60)  fps[0]: 19.73
IMX219: IMX219_IsiExposureControlIss: g=168.960999, Ti=0.050000
CAMERIC-MI-IRQ: isp mi frame out (61)  fps[0]: 19.72
IMX219: IMX219_IsiExposureControlIss: g=168.960999, Ti=0.050000
CAMERIC-MI-IRQ: isp mi frame out (62)  fps[0]: 19.72
IMX219: IMX219_IsiExposureControlIss: g=168.960999, Ti=0.050000
CAMERIC-MI-IRQ: isp mi frame out (63)  fps[0]: 19.71
...
...
```

The output above indicates your CSI camera is working well.

Important: Usage of other demo applications will be added to this page as well.

Source for these demo application can be found here

8.6 Support

8.6.1 Certifications and export control

Export designations

Todo: update details

- HS:
- US HS:
- EU HS:

Size and weight

Todo: update details

- Bare board dimensions:
- Bare board weight:
- Full package dimensions:
- Full package weight:

8.6.2 Additional documentation

Hardware docs

For any hardware document like schematic diagram PDF, EDA files, issue tracker, and more you can checkout the BeagleV Ahead design repository.

Software docs

For BeagleV Ahead specific software projects you can checkout all the BeagleV Ahead project repositories group.

Support forum

For any additional support you can submit your queries on our forum, https://forum.beagleboard.org/c/beaglev

Pictures

8.6.3 Change History

Note: This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

Document Changes

For all changes, see https://git.beagleboard.org/docs/docs.beagleboard.io. Frozen releases tested against specific hardware and software revisions are noted below.

Table 8.5: BeagleV Ahead document change history

Rev	Changes	Date	By

Board Changes

For all changes, see https://git.beagleboard.org/beaglev-ahead/beaglev-ahead. Versions released into production are noted below.

Table 8.6: BeagleV Ahead board change history

Rev	Changes	Date	By
		2023-03-08	

Chapter 9

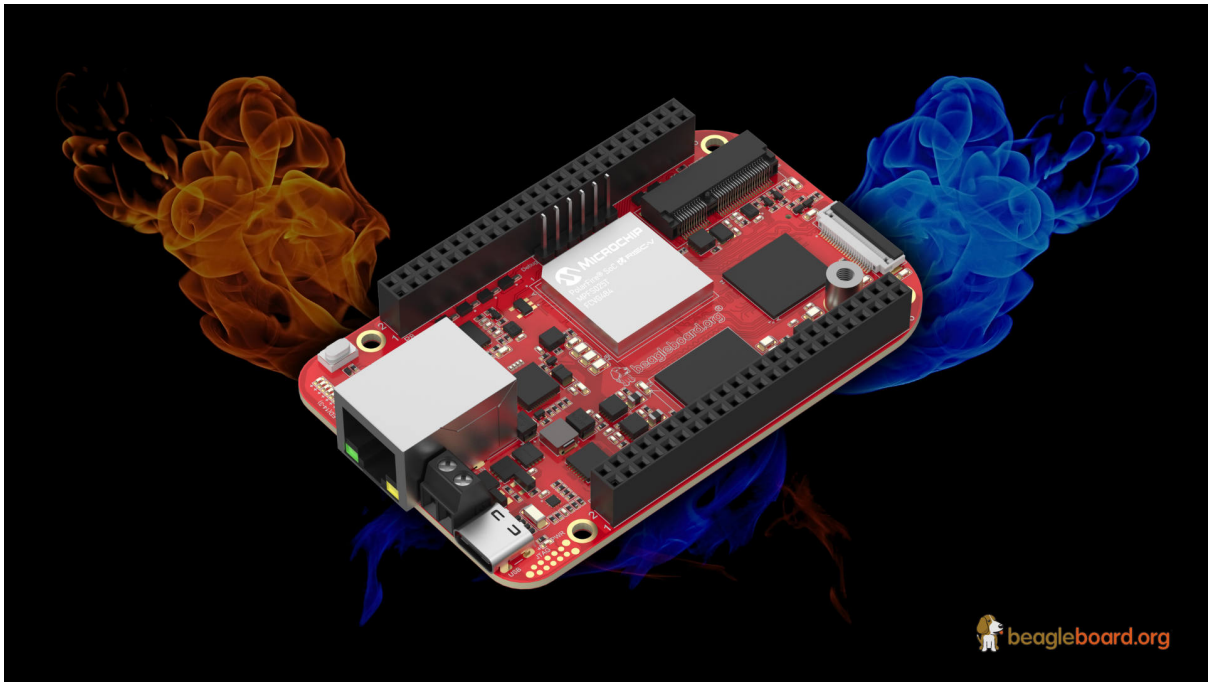
BeagleV-Fire

BeagleV®-Fire is a revolutionary SBC powered by the Microchip's PolarFire® MPFS025T RISC-V System on Chip (SoC) with FPGA fabric. BeagleV®-Fire opens up new horizons for developers, tinkerers, and the open-source community to explore the vast potential of RISC-V architecture and FPGA technology. It has the same P8 & P9 cape header pins as BeagleBone Black allowing you to stack your favorite BeagleBone cape on top to expand it's capability. Built around the powerful and energy-efficient RISC-V instruction set architecture (ISA) along with its versatile FPGA fabric, BeagleV®-Fire SBC offers unparalleled opportunities for developers, hobbyists, and researchers to explore and experiment with RISC-V technology.



License Terms

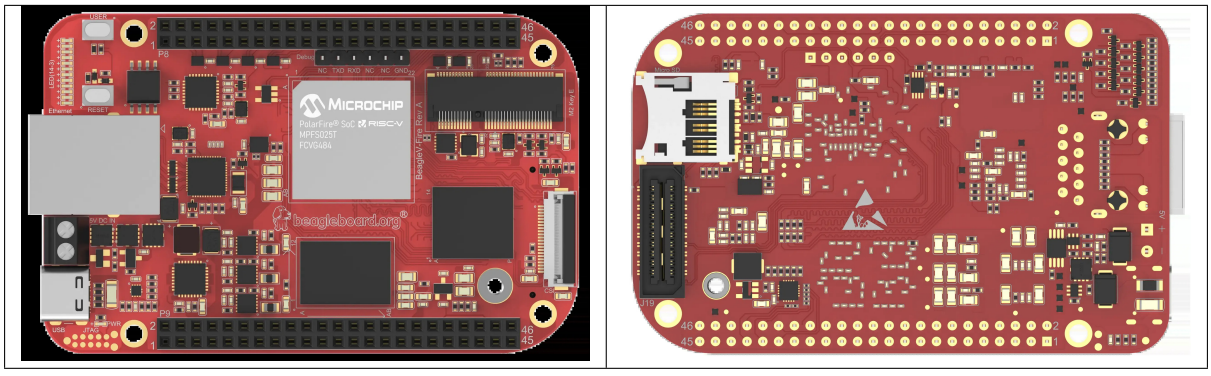
- This documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page
- For export, emissions and other compliance, see *Support*



Important: This is a work in progress, for latest documentation please visit https://docs.beagleboard.org/latest/

9.1 Introduction

BeagleV®-Fire is a revolutionary SBC powered by the Microchip's PolarFire® MPFS025T System on Chip (SoC) with 4x RV64GC Application cores, 1x RV64IMAC monitor/boot core, and FPGA fabric. BeagleV®-Fire opens up new horizons for developers, tinkerers, and the open-source community to explore the vast potential of RISC-V architecture and FPGA technology. It has the same P8 & P9 cape header pins as BeagleBone Black allowing you to stack your favourite BeagleBone cape on top to expand it's capability. Built around the powerful and energy-efficient RISC-V instruction set architecture (ISA) along with its versatile FPGA fabric, BeagleV®-Fire SBC offers unparalleled opportunities for developers, hobbyists, and researchers to explore and experiment with RISC-V technology.



9.1.1 Pinout Diagrams

Choose the cape header to see respective pinout diagram.

P8 cape header

P9 cape header

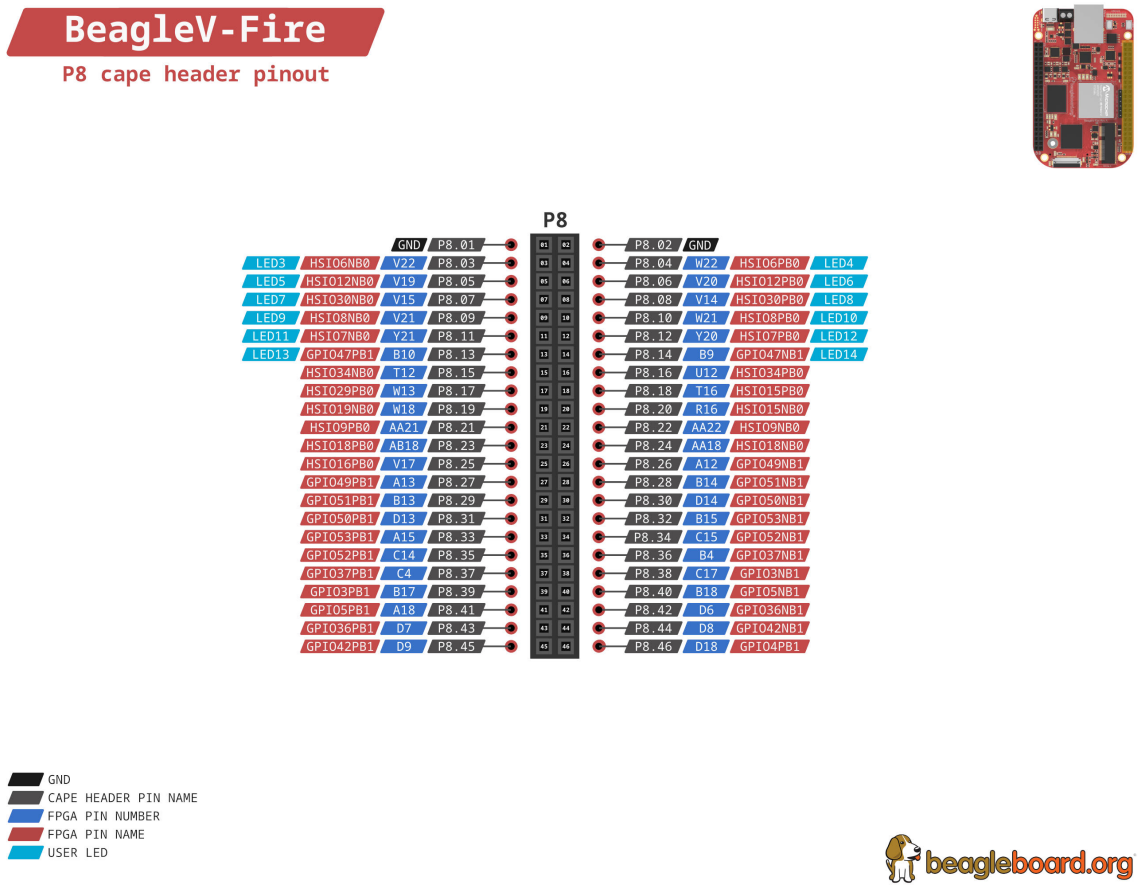


Fig. 9.1: BeagleV-Fire P8 cape header pinout

9.1.2 Detailed overview

Table 9.1: BeagleV-Fire features

Feature	Description
Processor	MPFS025T-FCVG484E
Memory	2GB (1Gb x 16)- 1866MHz 3733Mbps, LPDDR4
Storage	Kingston 16GB eMMC
Wireless	1x M.2 Key E, support 2.4GHz/5GHz WiFi module
Ethernet	• PHY: Realtek RTL8211F-VD-CG Gigabit Ethernet phy • Connector: integrated magnetics RJ-45
USB C	• Connectivity: Flash/programming support • Power: Input: 5V @ 3A
Other connectors	• 1x SYZYGY High speed connector • microSD card slot • CSI connector compatible with BeagleBone AI-64, BeagleV-Ahead, Raspberry Pi Zero / CM4 (22-pin)

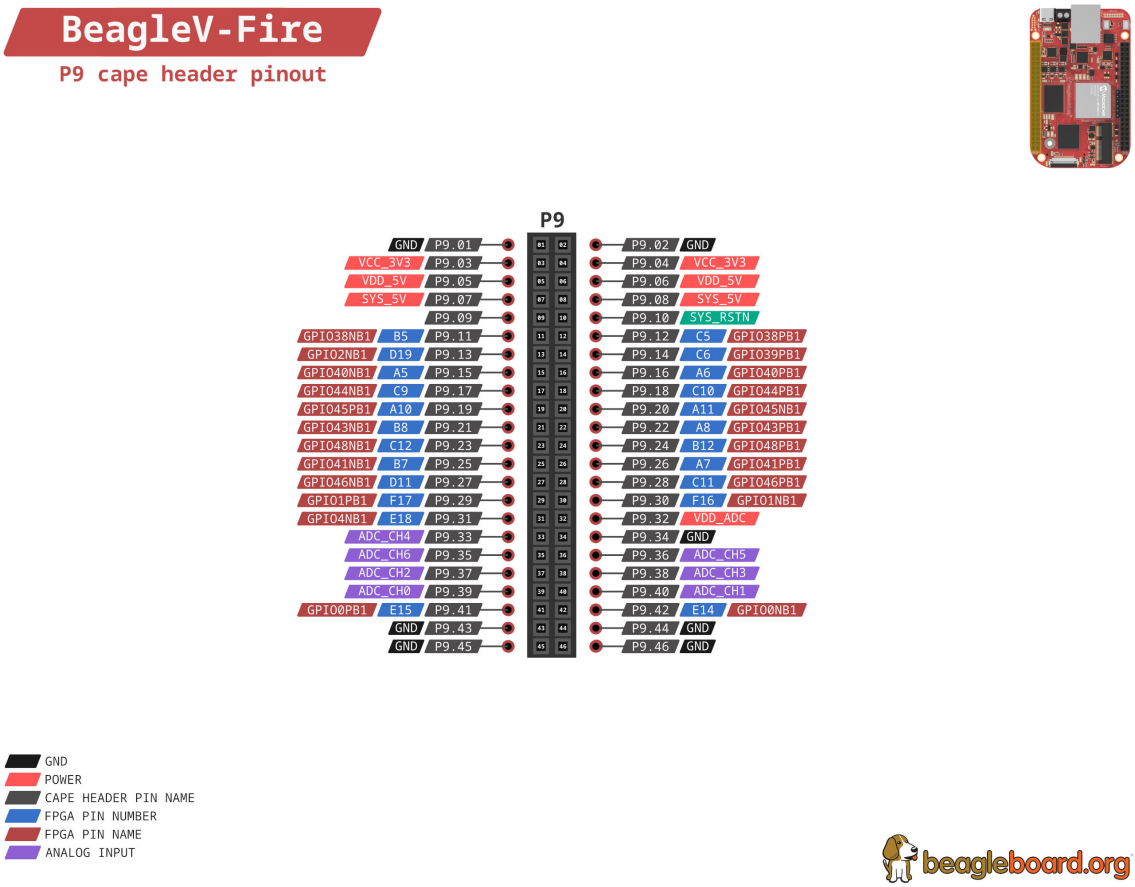


Fig. 9.2: BeagleV-Fire P9 cape header pinout

Board components location

This section describes the key components on the board, their location and function.

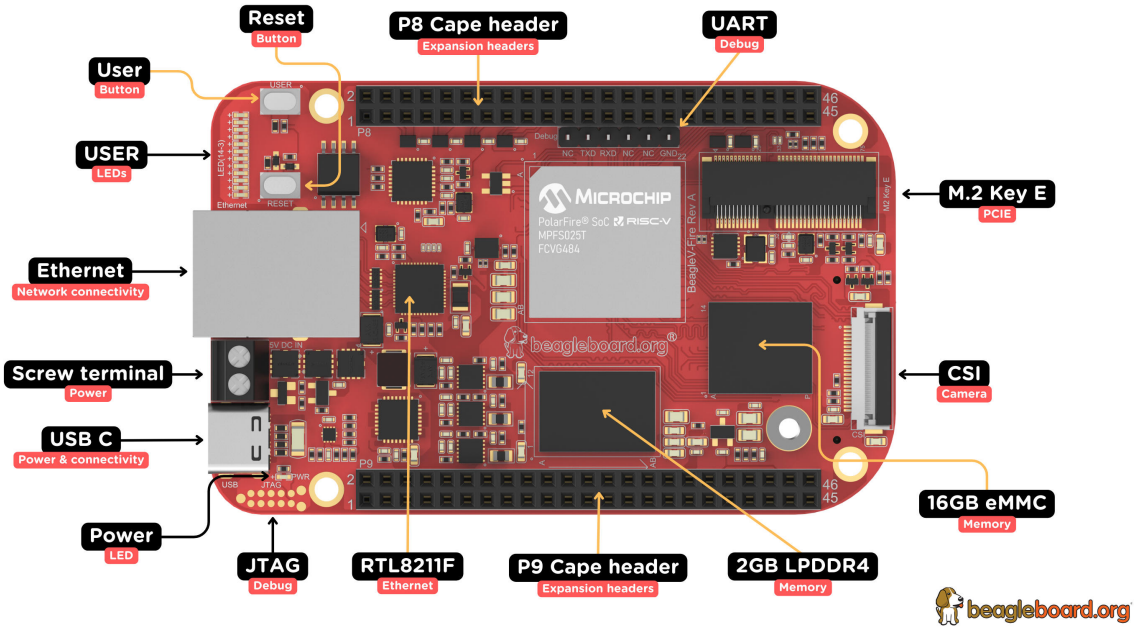


Fig. 9.3: BeagleV-Fire board front components location

Front components location

Table 9.2: BeagleV-Fire board front components location

Feature	Description
Power LED	Power (Board ON) indicator
JTAG (MPFS025T)	MPFS025T SoC JTAG debug port
RTL8211F	Gigabit IEEE 802.11 Ethernet PHY
P8 & P9 cape header	Expansion headers for BeagleBone capes.
2GB RAM	2GB (1Gb x 16)- 1866MHz 3733Mbps, LPDDR4
16GB eMMC	Kingston 16GB eMMC Flash storage
CSI	22pin MIPI Camera connectors
M.2 Key E	PCIE M.2 Key E connector
UART debug header	6 pin UART debug header
Reset button	Press to reset BeagleV-Fire board (MPFS025T SoC)
User button	User defined (custom) action button
User LEDs	12x user programmabkle LEDs to show various board status during boot.
GigaBit Ethernet	1Gb/s Wired internet connectivity
Barrel jack	Power input
USB C	Power, connectivity, and board flashing.

Back components location

Table 9.3: BeagleV-Fire board back components location

Feature	Description
microSD	microSD card slot
SYZYGY	SYZYGY High speed connector

9.2 Quick Start

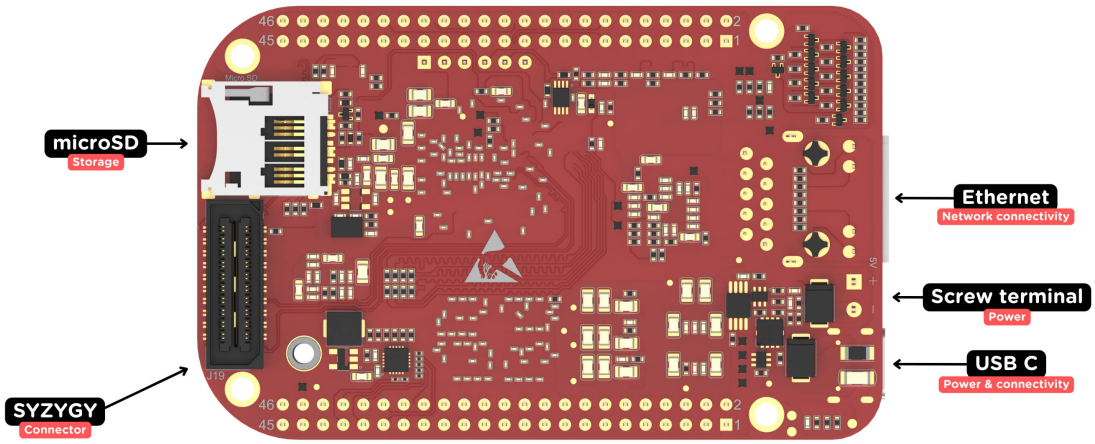


Fig. 9.4: BeagleV-Fire board back components location

9.2.1 What's included in the box?

When you purchase a brand new BeagleV-Fire, In the box you'll get:

Todo: add image & information about box content.

9.2.2 Unboxing



9.2.3 Tethering to PC

To connect BeagleV-Fire board to PC via USB Type C receptacle you need a USB type C cable. Connection guide for the same is shown below:

Tip: To get a USB type C cable you can checkout links below:

1. USB C cable 0.3m (mouser)
2. USB C cable 1.83m (digikey)



Fig. 9.5: BeagleV-Fire tethered connection

9.2.4 Flashing eMMC

Flash the latest image on eMMC

9.2.5 Access UART debug console

Note: Some tested devices that are working good includes:

1. Adafruit CP2102N Friend - USB to Serial Converter
2. Raspberry Pi Debug Probe Kit for Pico and RP2040

To access a BeagleV-Fire serial debug console you can connected a USB to UART to your board as shown below:

To see the board boot log and access your BeagleV-Fire's console you can use application like `tio` to access the conole. If you are using Linux your USB to UART converter may appear as `/dev/ttyUSB`. It will be different for Mac and Windows operatig systems. To find serial port for your system you can checkout this guide.

```
[lorforlinux@fedora ~] $ tio /dev/ttyUSB0
tio v2.5
```

(continues on next page)

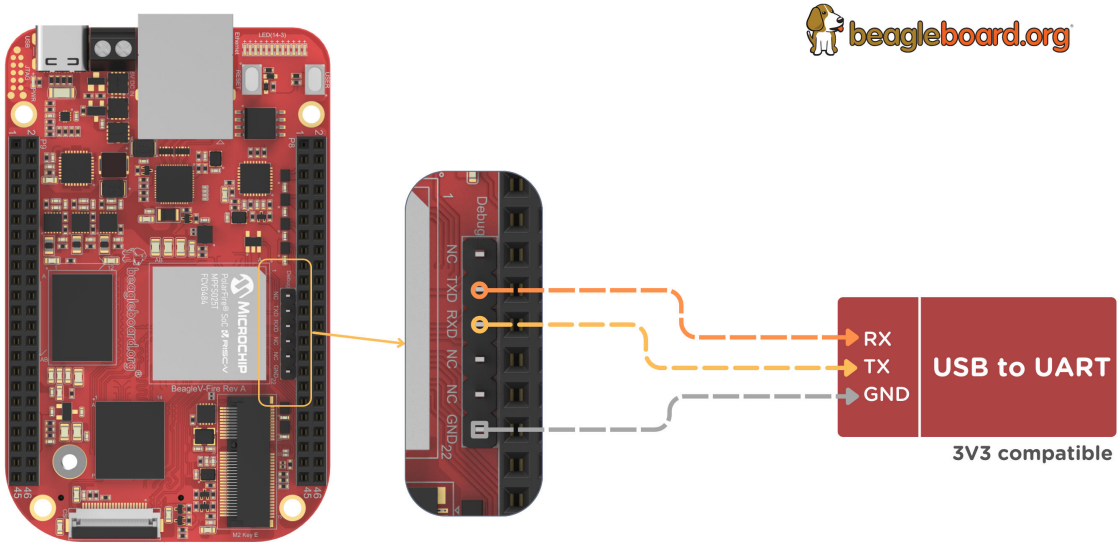


Fig. 9.6: BeagleV-Fire UART debug port connection

(continued from previous page)

```
Press ctrl-t q to quit
Connected
```

9.2.6 Demos and Tutorials

- *How to retrieve BeagleV-Fire's gateware version*
- *Flashing gateware and Linux image*
- *Gateware Design Introduction*
- *Microchip FPGA Tools Installation Guide*

9.3 Design & specifications

If you want to know how BeagleV-Fire board is designed and what are it's high-level specifications then this chapter is for you. We are going to discuss each hardware design element in detail and provide high-level device specifications in a short and crisp form as well.

Tip: For hardware design files and schematic diagram you can checkout BeagleV-Fire GitLab repository: https://git.beagleboard.org/beaglev-fire/beaglev-fire

---

9.3.1 Block diagram

9.3.2 System on Chip (SoC)

9.3.3 Power management

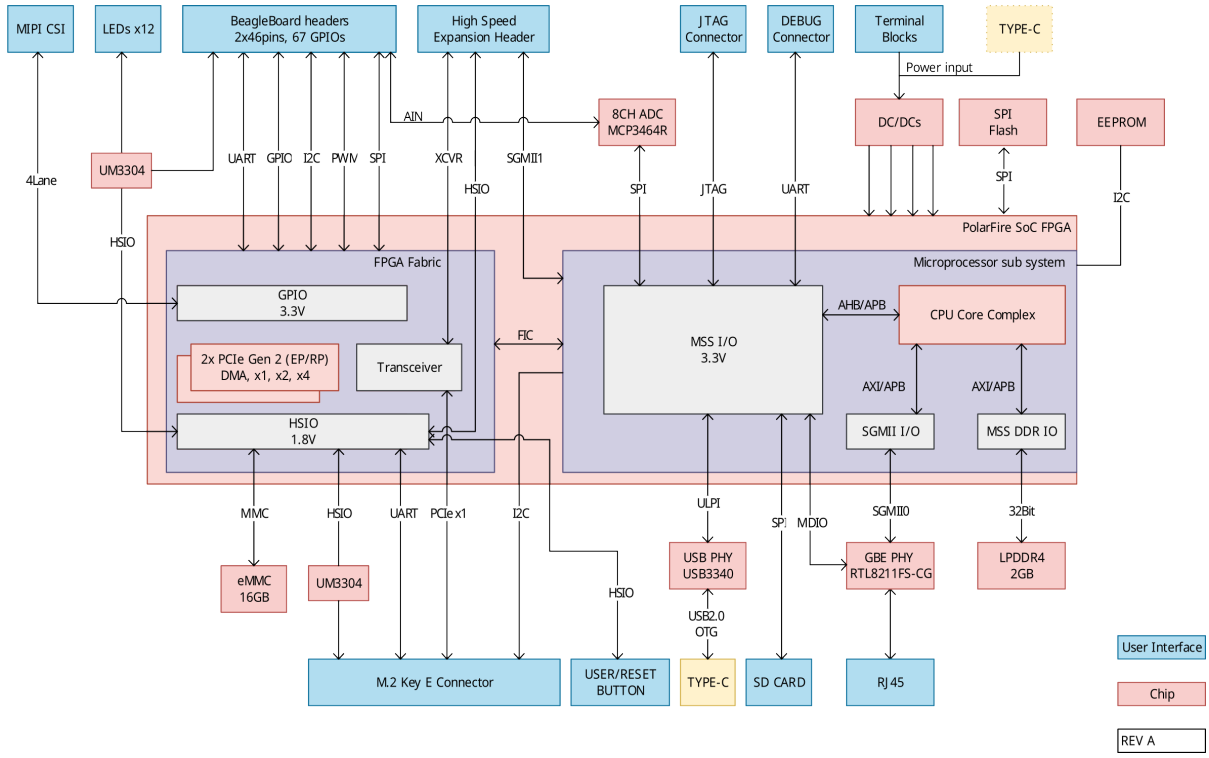


Fig. 9.7: System block diagram

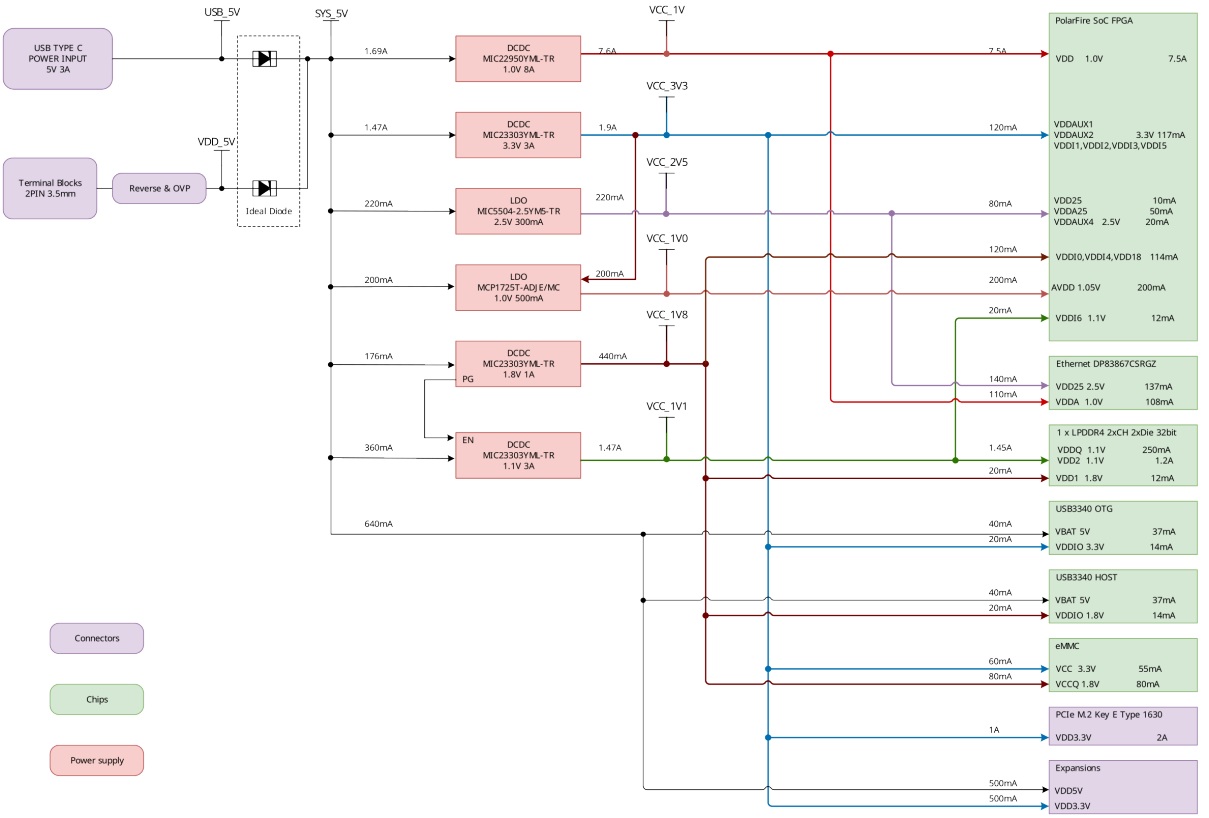


Fig. 9.8: Power tree diagram

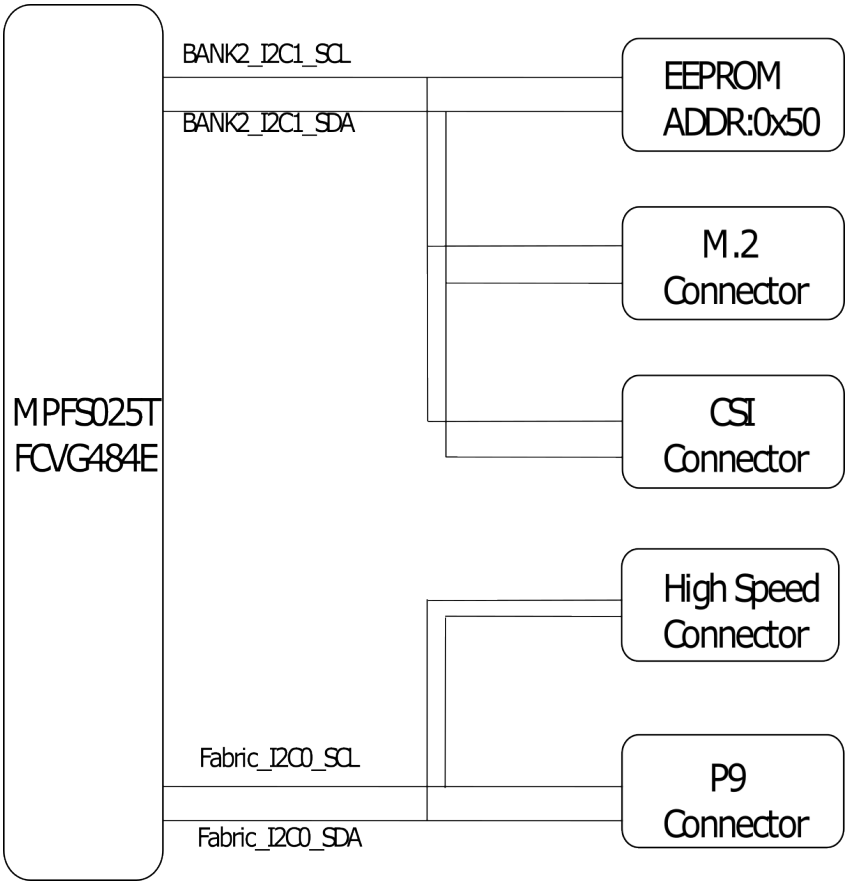


Fig. 9.9: I2C tree diagram

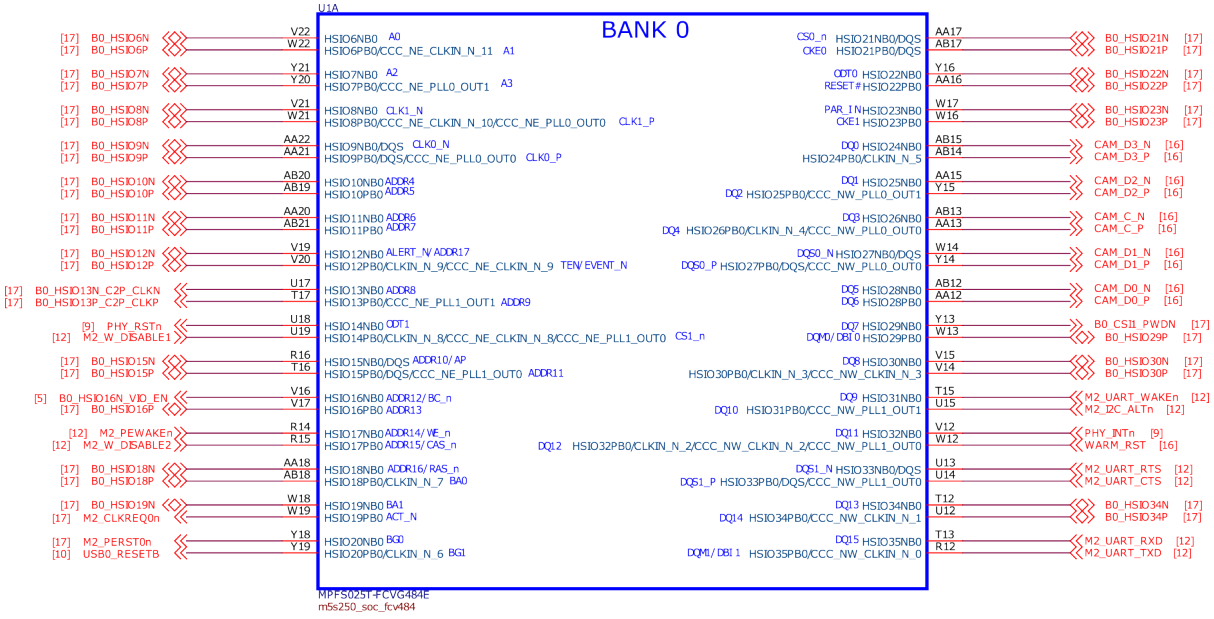


Fig. 9.10: SoC bank0

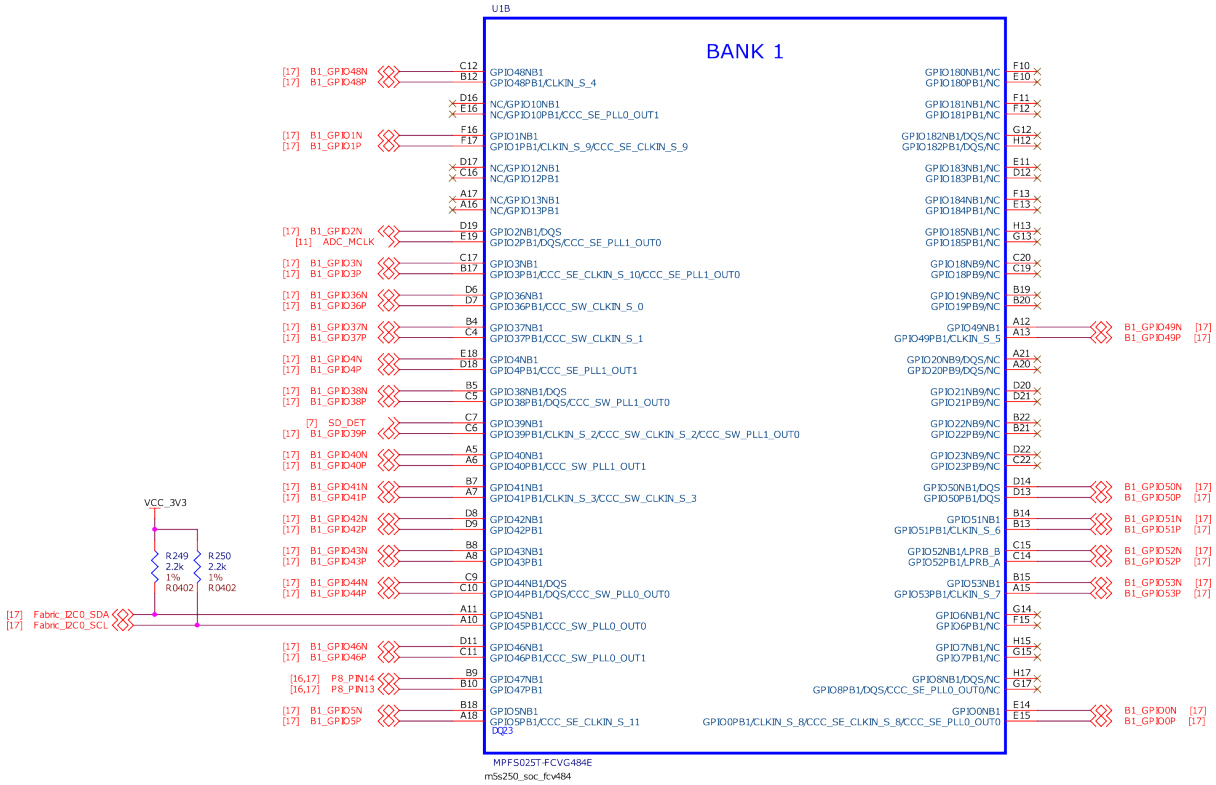


Fig. 9.11: SoC bank1

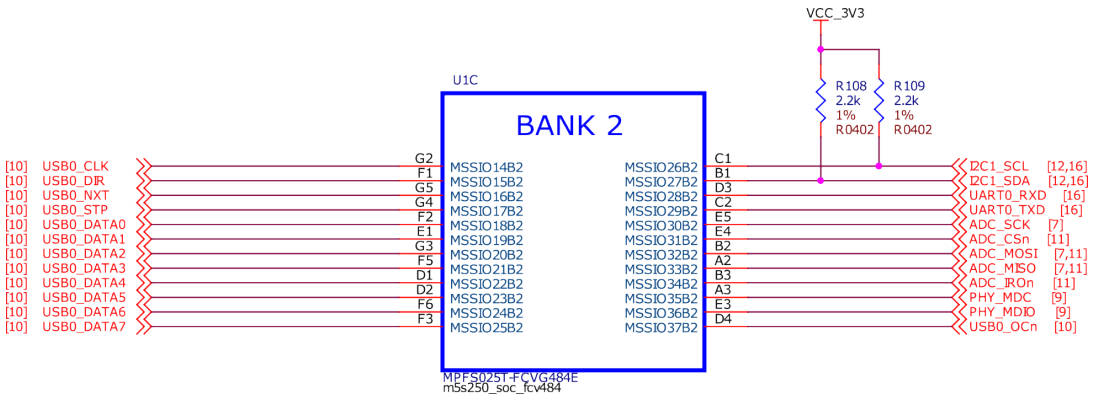


Fig. 9.12: SoC bank2

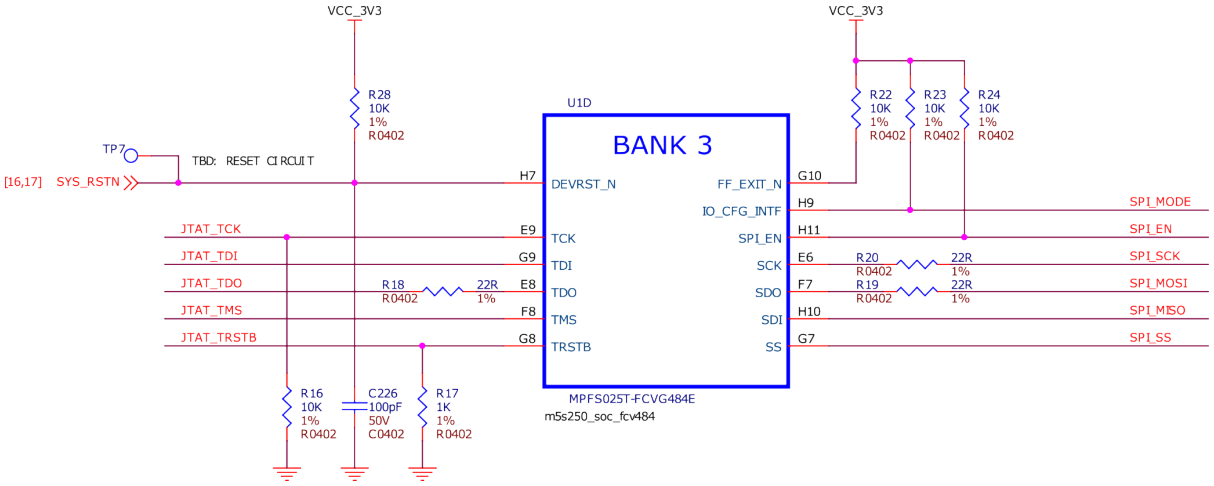


Fig. 9.13: SoC bank3

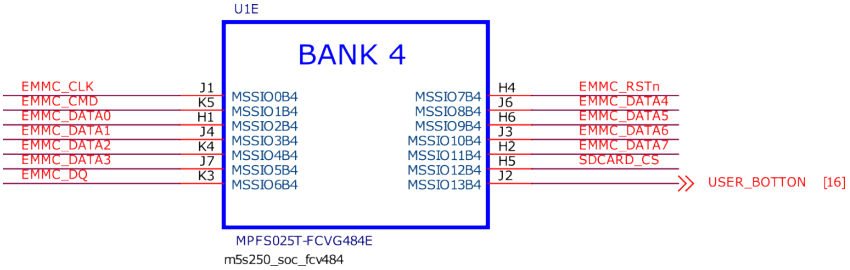


Fig. 9.14: SoC bank4

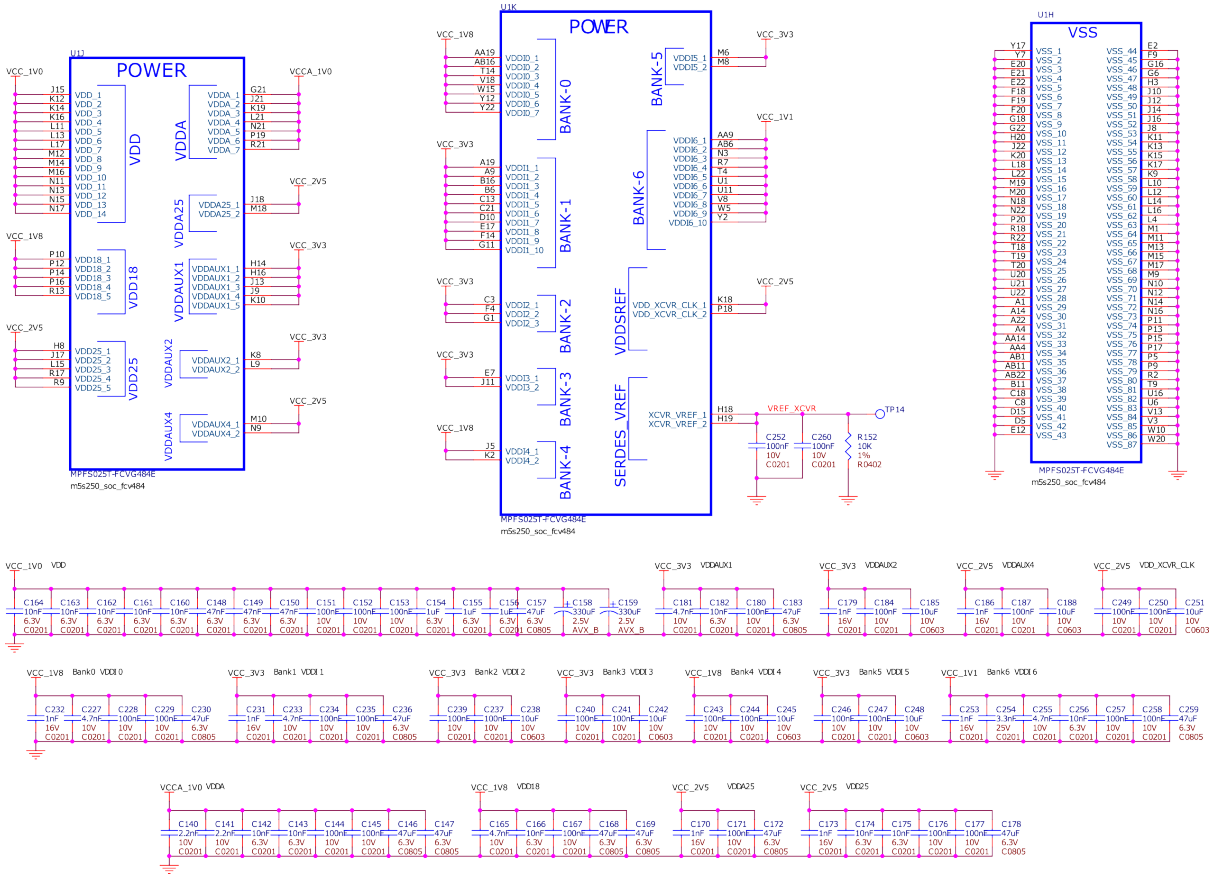


Fig. 9.15: SoC power

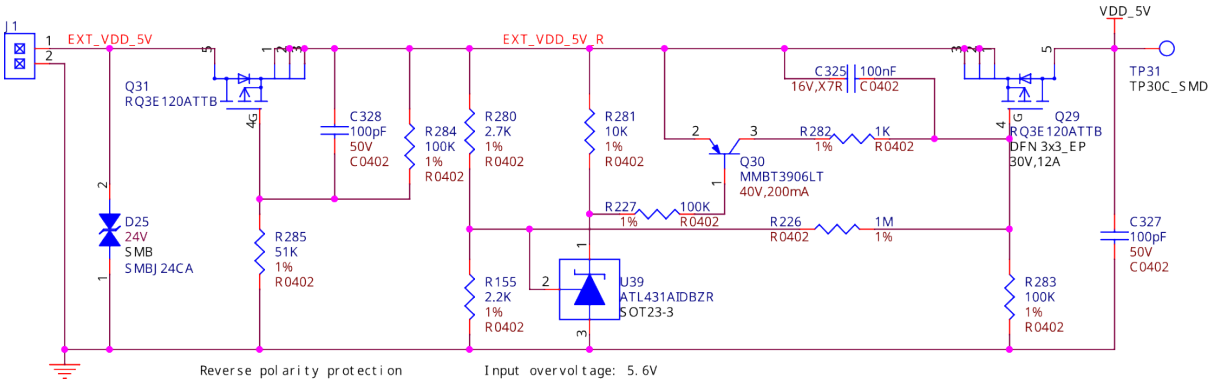


Fig. 9.16: DC 5V input

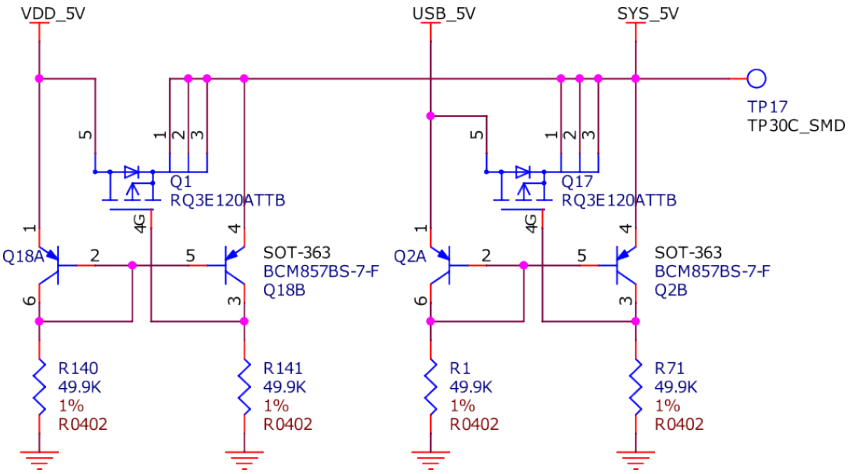


Fig. 9.17: Ideal diode

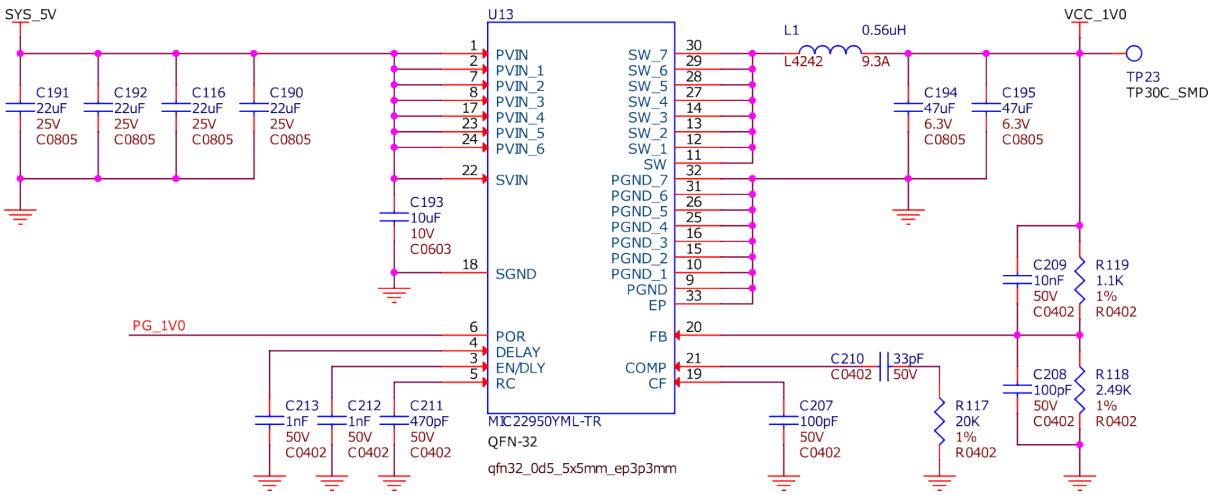


Fig. 9.18: VCC 1V0

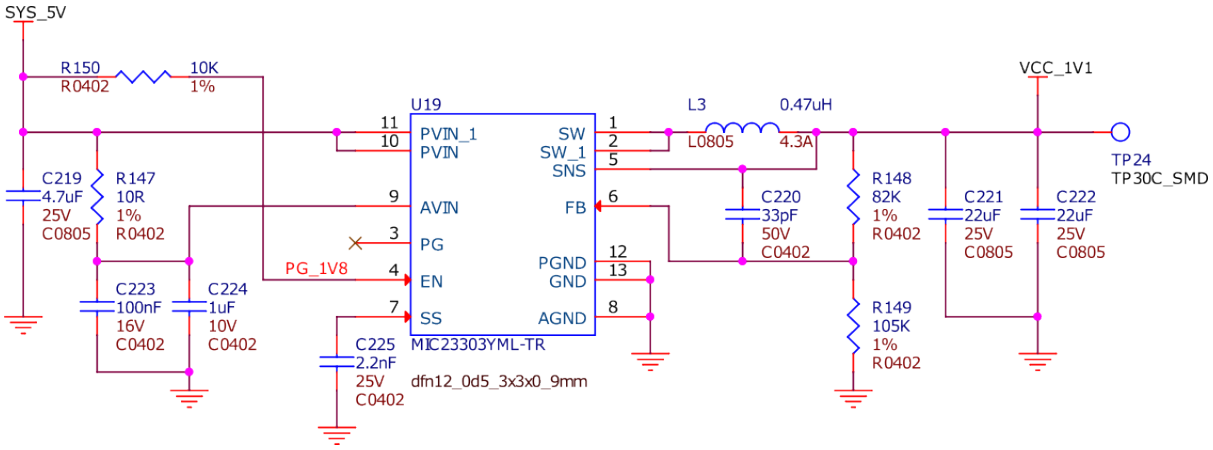


Fig. 9.19: VCC 1V1

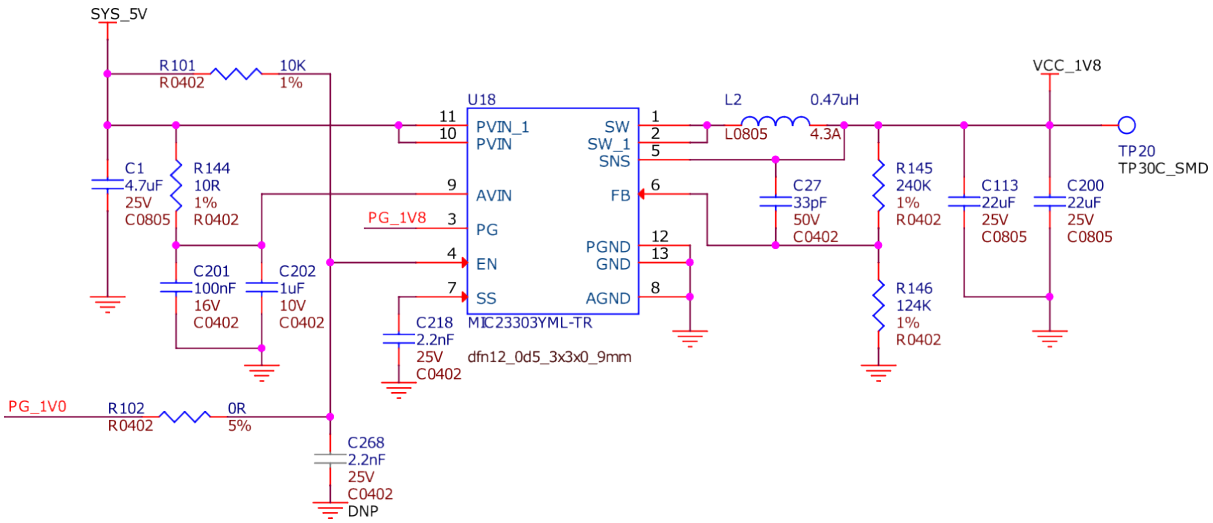


Fig. 9.20: VCC 1V8

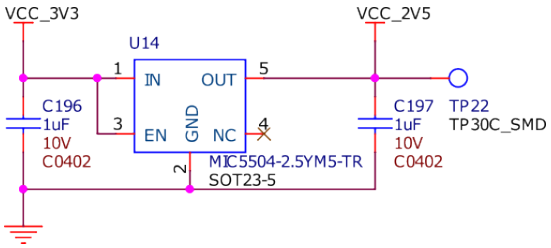


Fig. 9.21: VCC 2V5

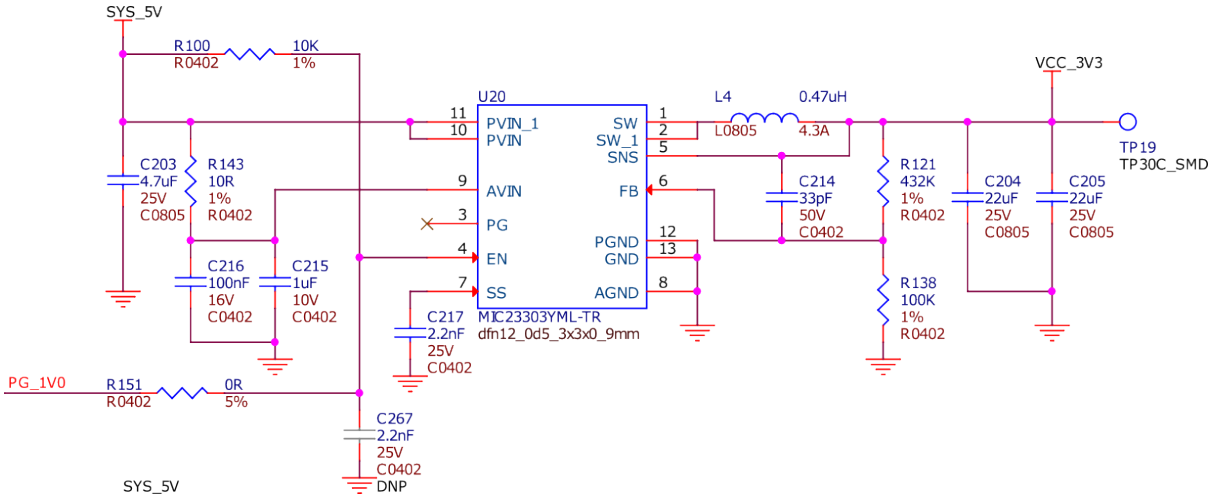


Fig. 9.22: VCC 3V3

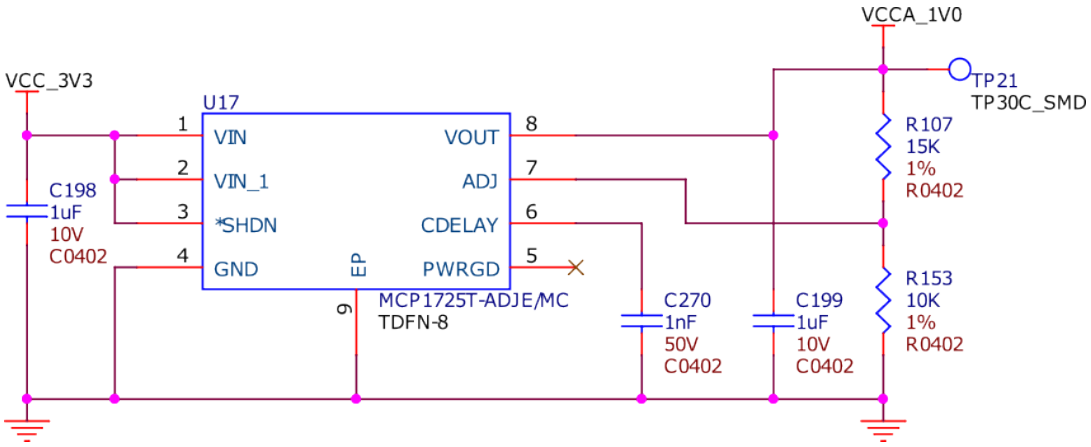


Fig. 9.23: VCCA 1V0

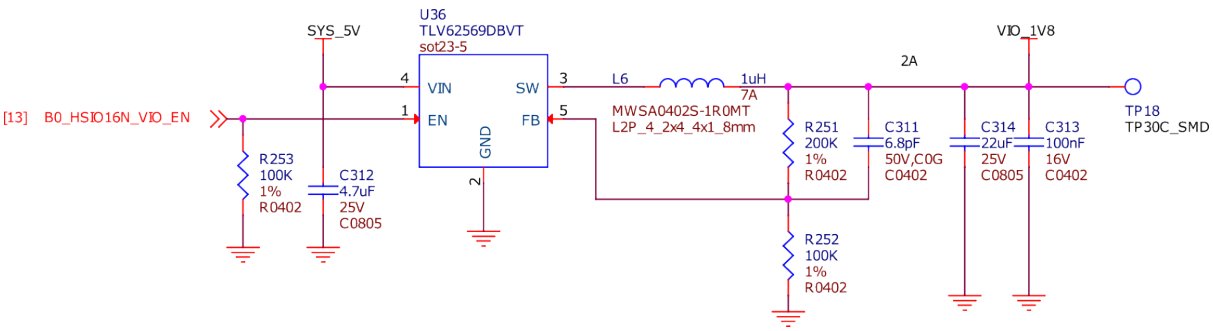


Fig. 9.24: VIO enable

9.3.4 General Connectivity and Expansion

USB-C port

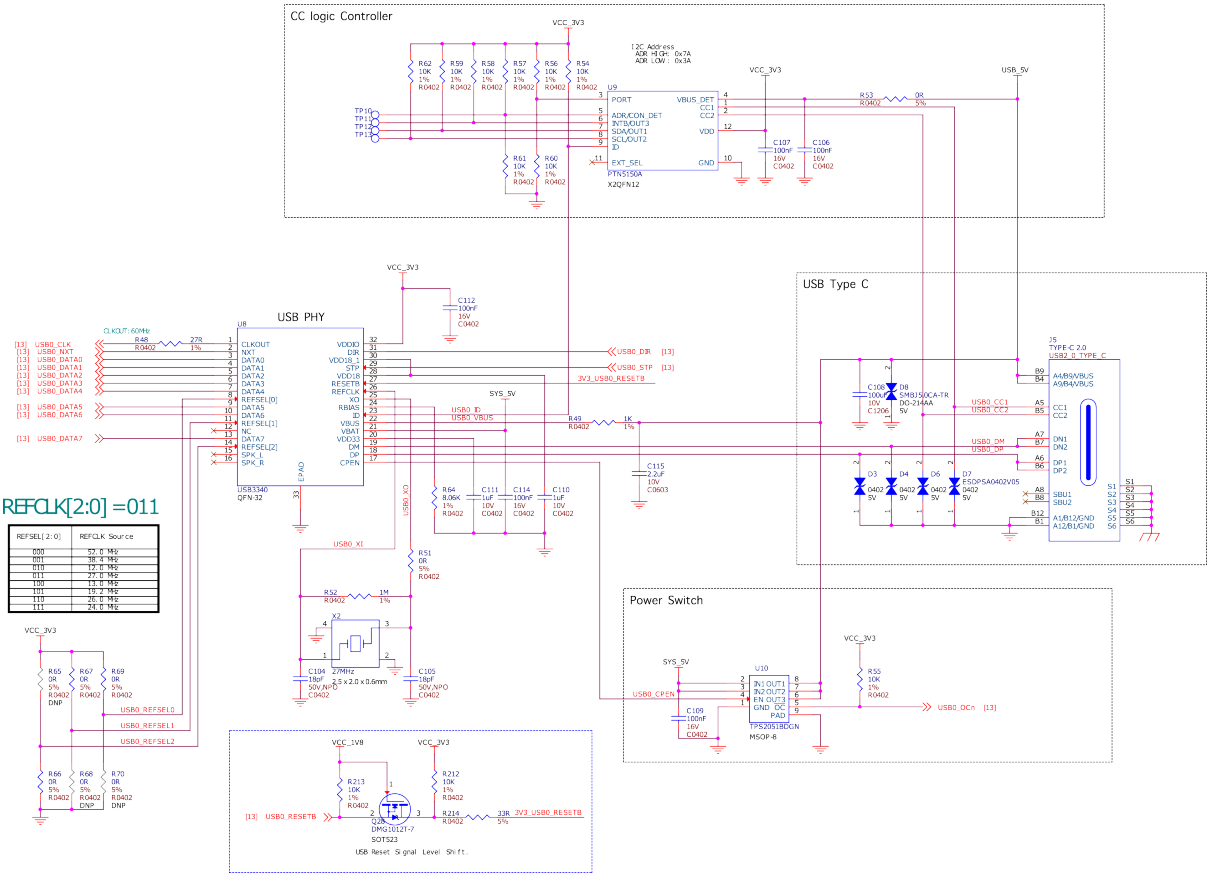


Fig. 9.25: USB C

P8 & P9 cape header pins

ADC

9.3.5 Buttons and LEDs

User LEDs and Power LED

User and reset button

9.3.6 Connectivity

Ethernet

9.3.7 Memory, Media and Data storage

DDR memory

eMMC

microSD

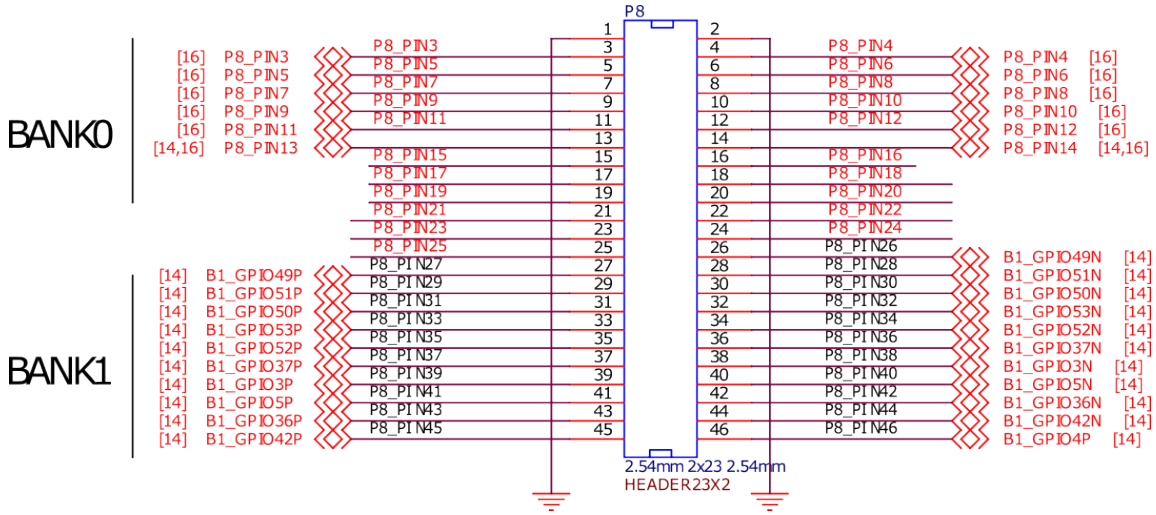


Fig. 9.26: P8 cape header

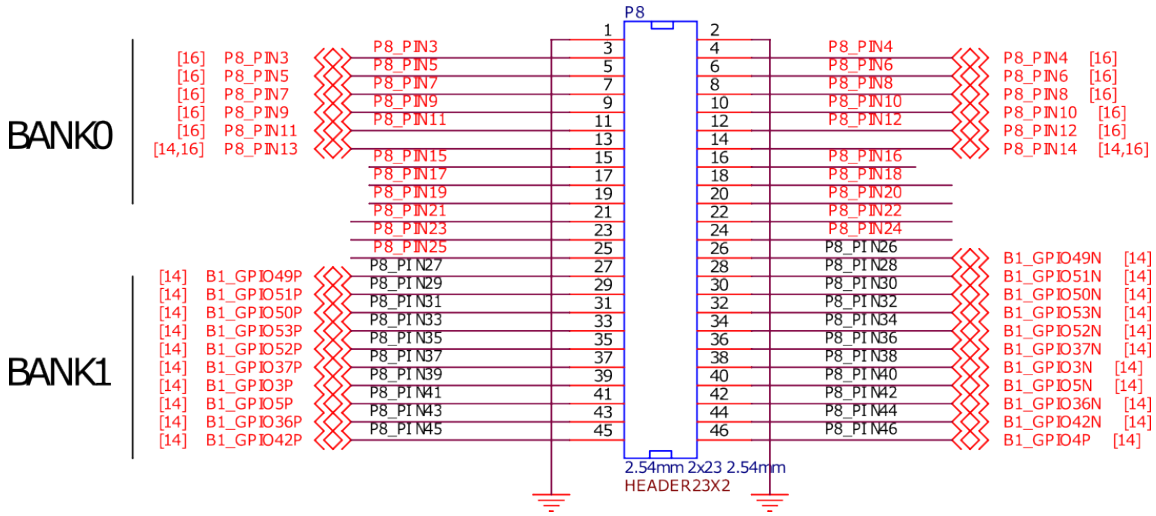


Fig. 9.27: P9 cape header

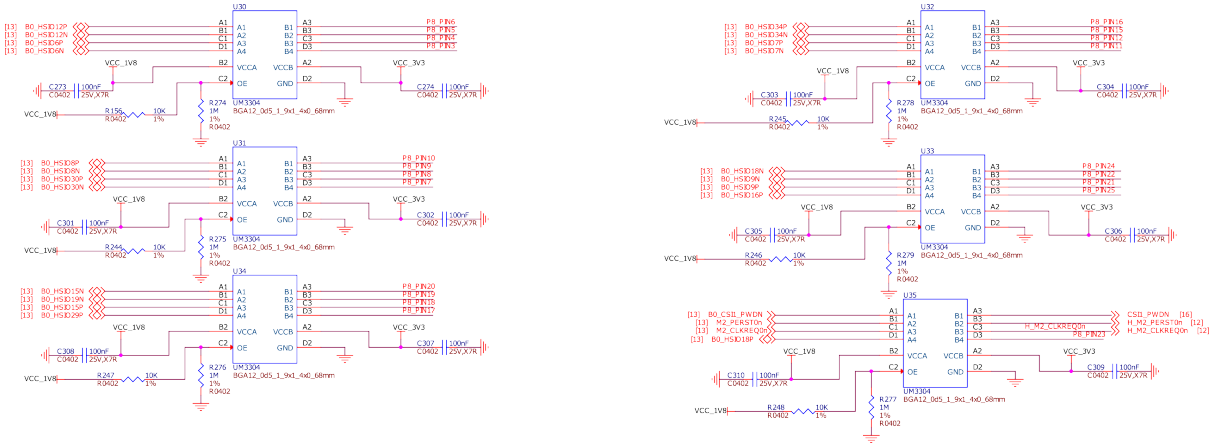


Fig. 9.28: Cape header voltage level translator

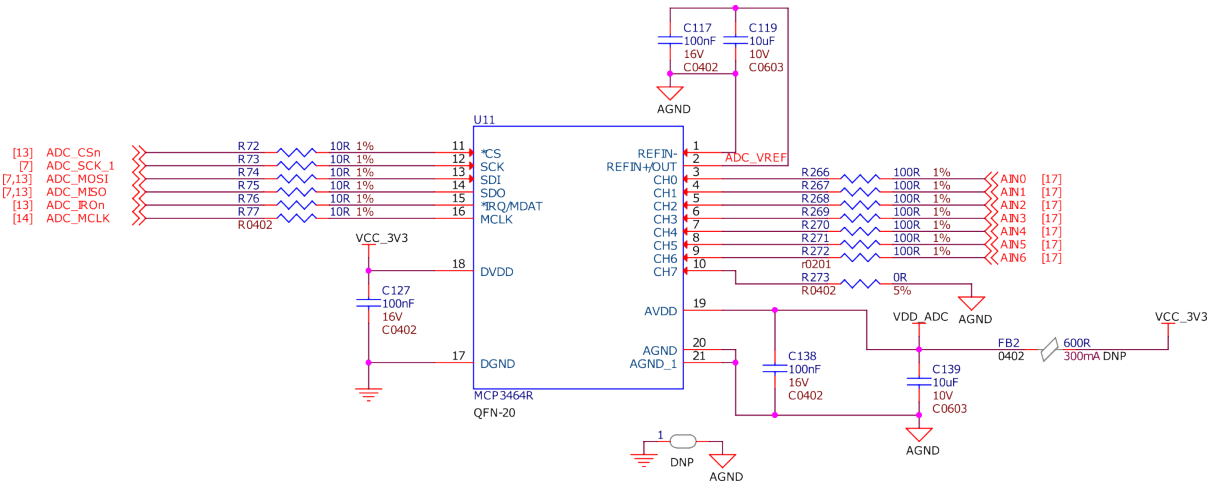


Fig. 9.29: 16bit Delta-Sigma ADC

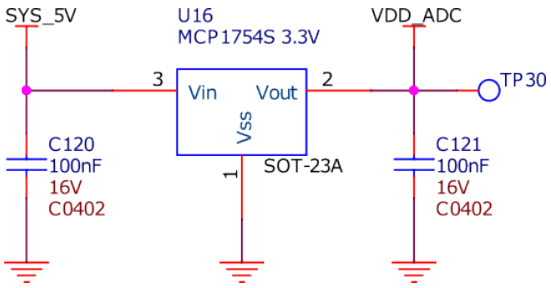


Fig. 9.30: ADC LDO power supply

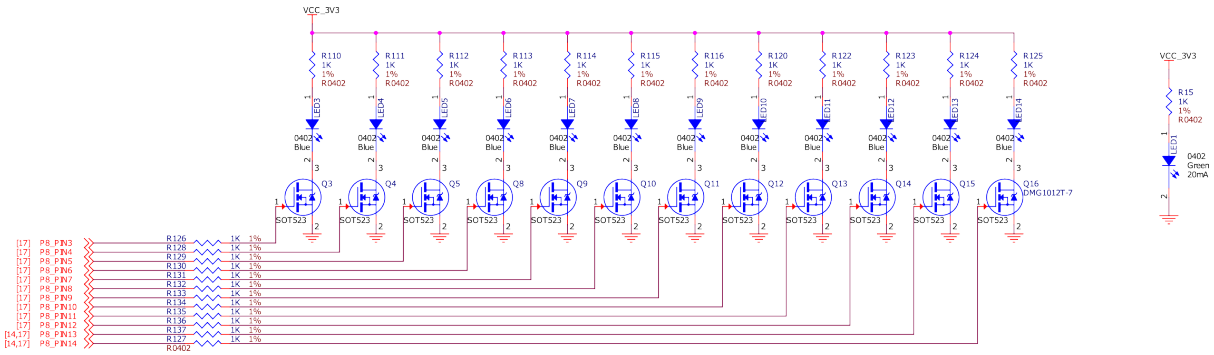


Fig. 9.31: User LEDs and power LED

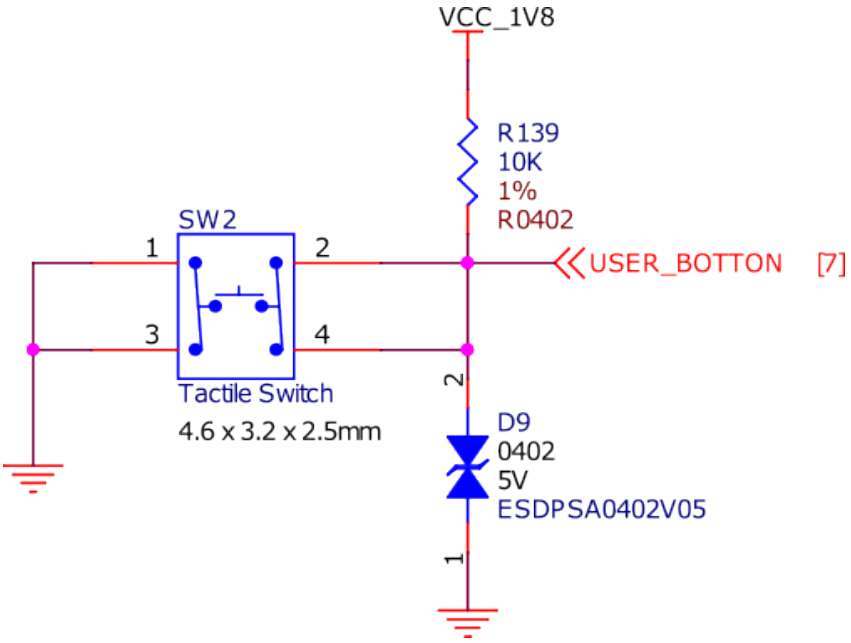


Fig. 9.32: User button

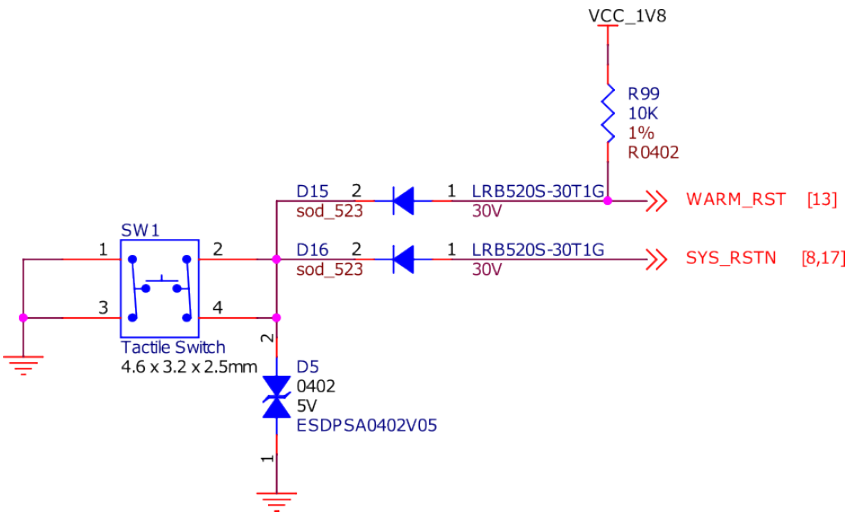


Fig. 9.33: Reset button

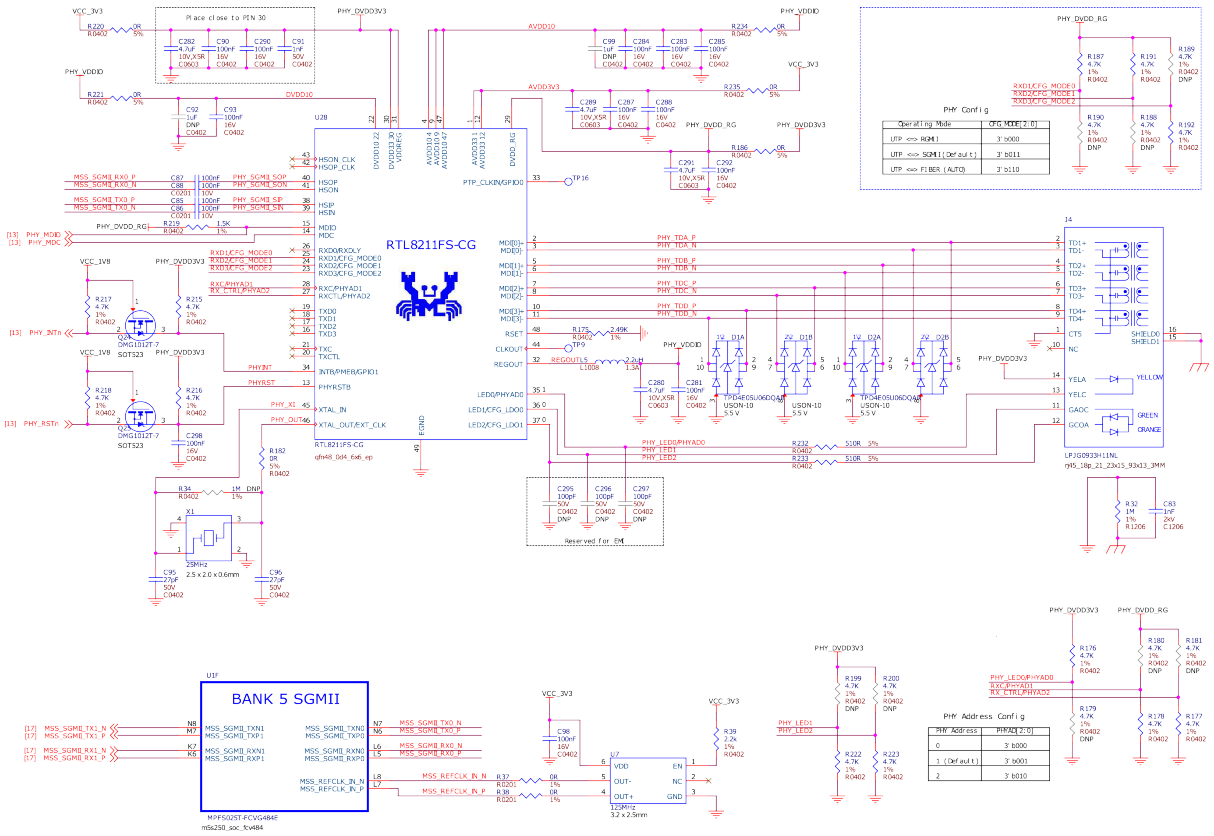


Fig. 9.34: Gigabit ethernet

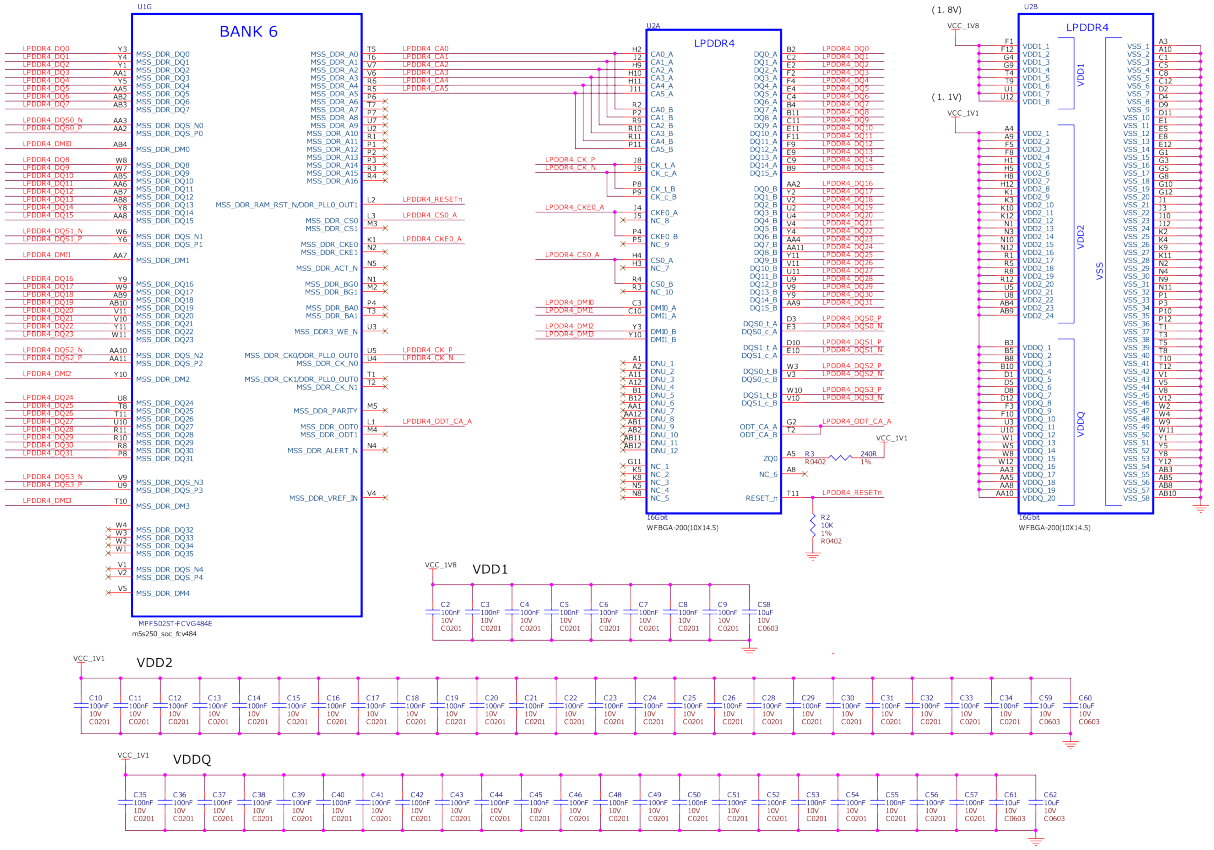


Fig. 9.35: LPDDR memory

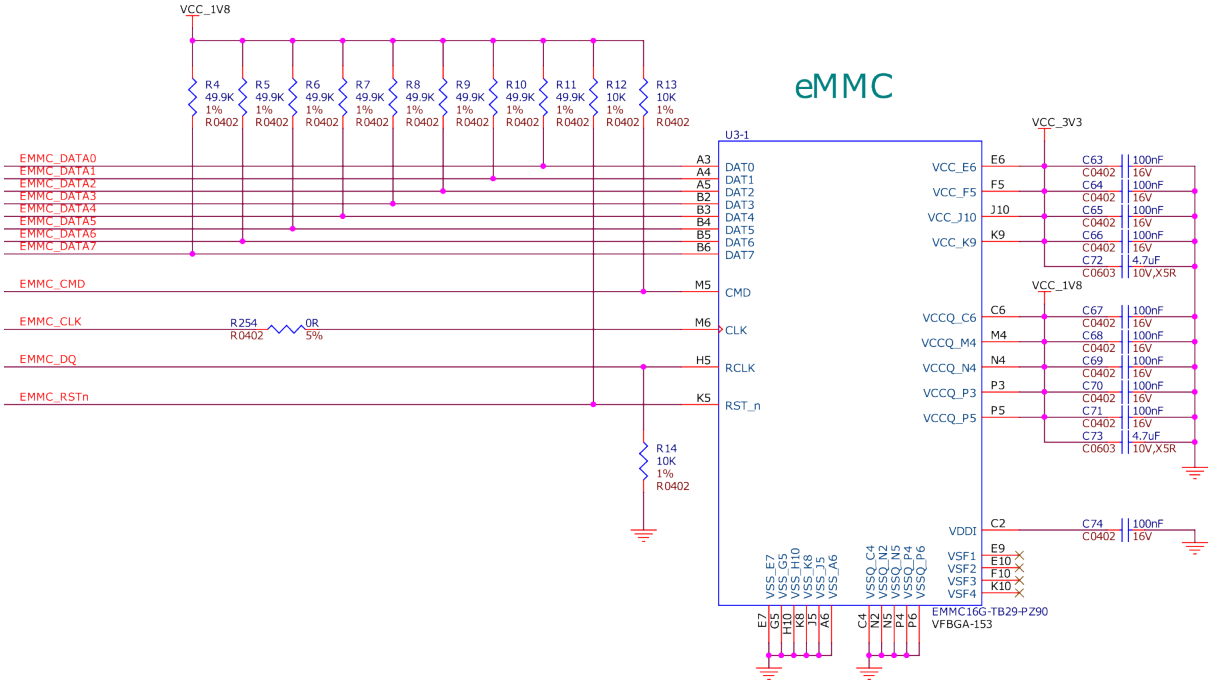


Fig. 9.36: EMMC flash storage

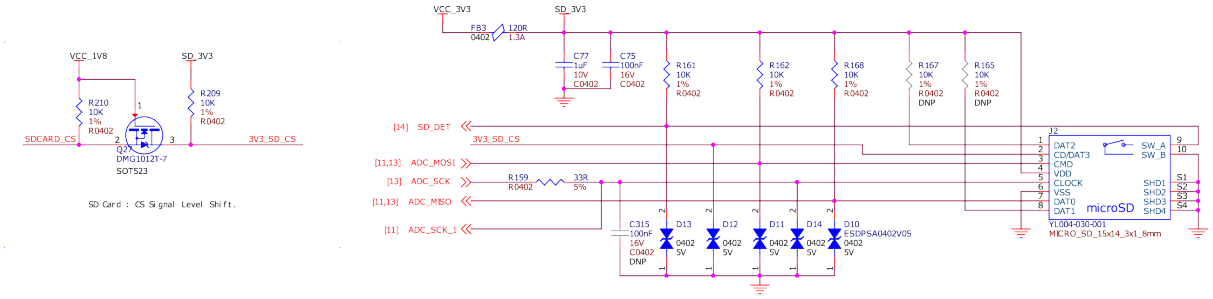


Fig. 9.37: SD Card socket

EEPROM

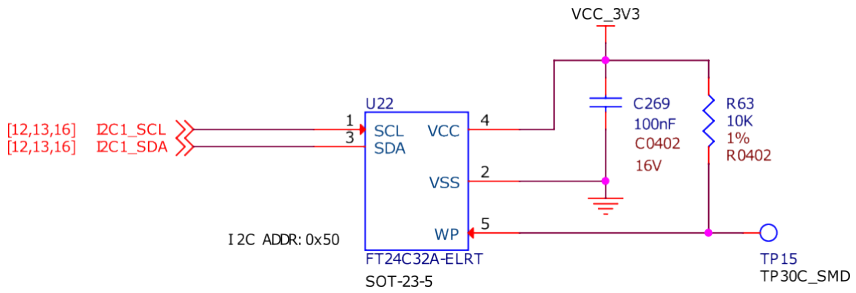


Fig. 9.38: EEPROM

SPI flash

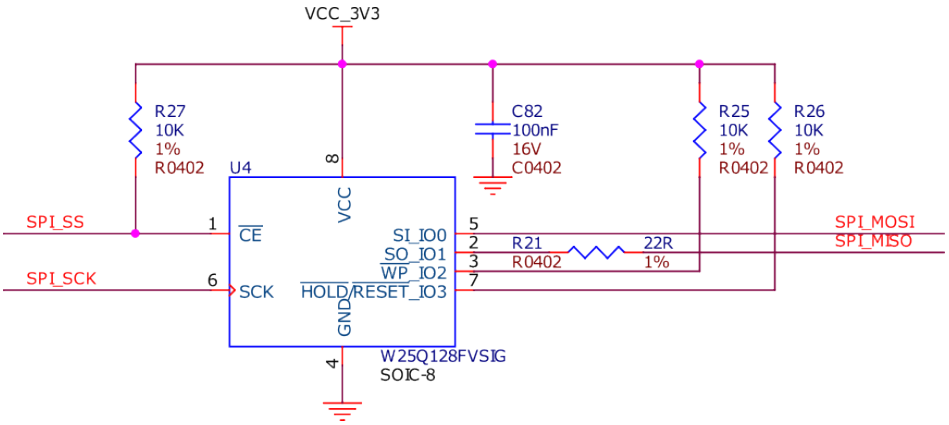


Fig. 9.39: SPI Flash

9.3.8 Multimedia I/O

CSI

9.3.9 Debug

UART debug port

JTAG debug port

9.3.10 Mechanical Specifications

Table 9.4: Dimensions & weight

Parameter	Values
Size	86.38 * 54.61 * 18.8 mm
Max heigh	18.8 mm
PCB Size	86.38 * 54.6 mm
PCB Layers	12 Layers
PCB Thickness	1.6 mm
RoHS compliant	Yes
Gross Weight	106 g
Net weight	45.8 g

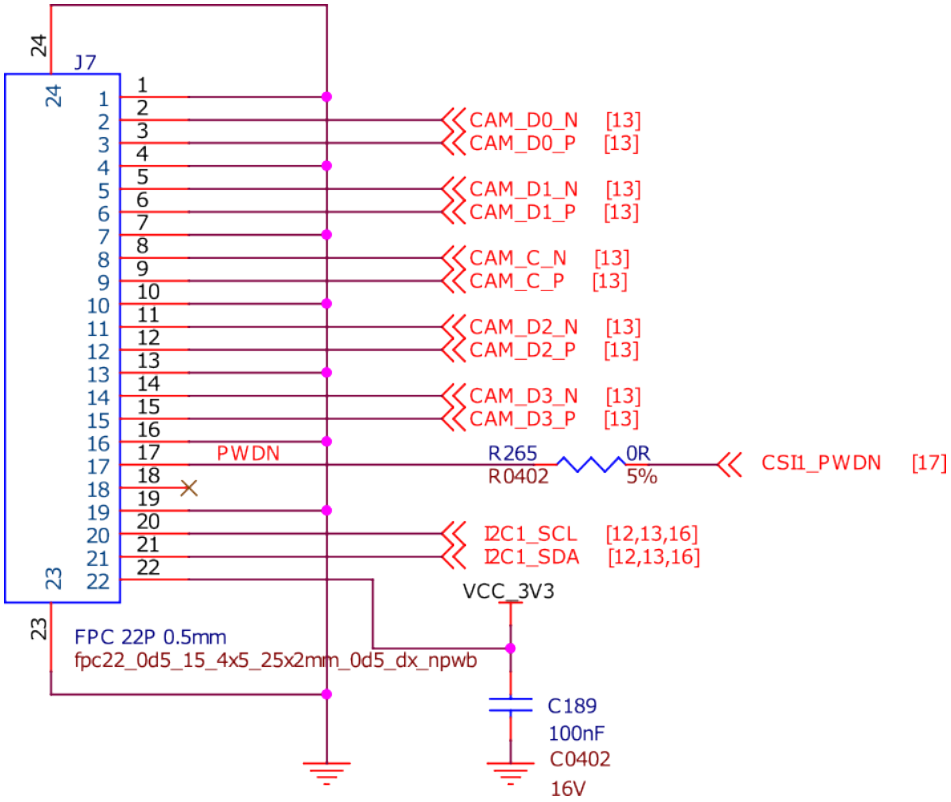


Fig. 9.40: CSI

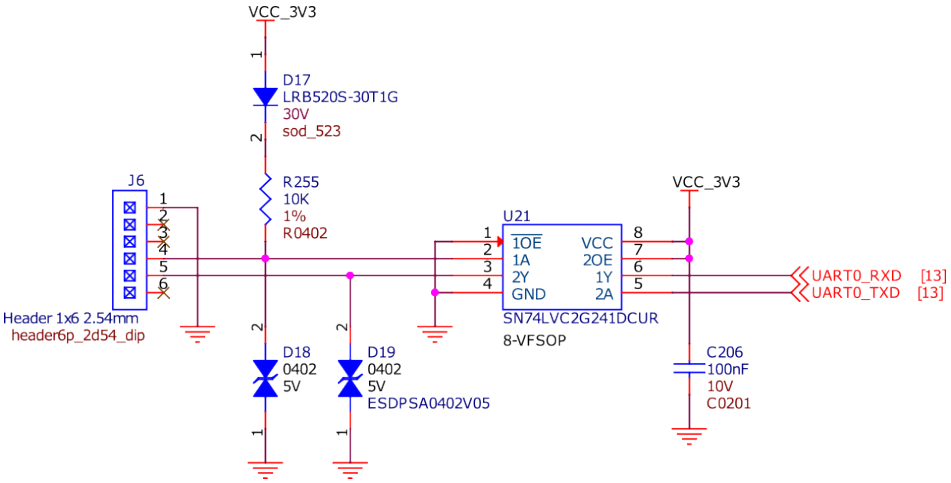


Fig. 9.41: UART debug header

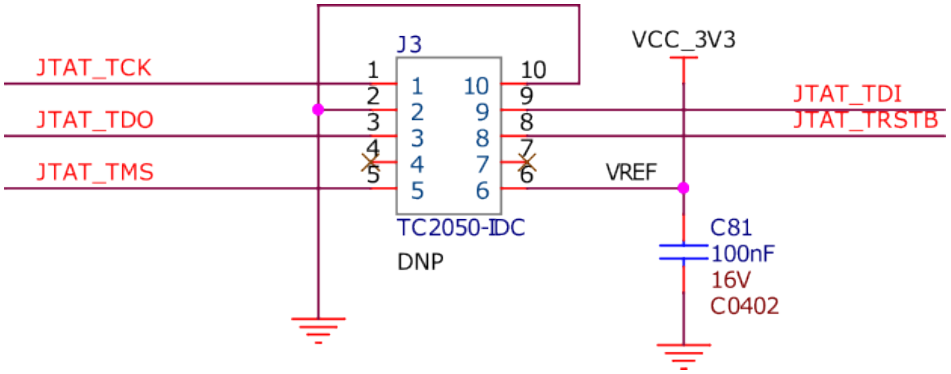


Fig. 9.42: JTAG debug header

9.4 Expansion

9.4.1 Cape Headers

Important: This page is a work in progress, don't use if for debugging.

The expansion interface on the board is comprised of two headers P8 (46 pin) & P9 (46 pin). All signals on the expansion headers are **3.3V** unless otherwise indicated.

Note: Do not connect 5V logic level signals to these pins or the board will be damaged.

Note: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

Connector P8

The following tables show the pinout of the **P8** expansion header. The gateware image is responsible for setting the function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **Name** row is the pin name on the processor.

The **BALL** row is the pin number on the processor.

The rows below **BALL** are the gateware setting for each pin.

NOTES:

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

P8.01-P8.02

P8.01	P8.02
GND	GND

P8.03-P8.05

Pin	P8.03	P8.04	P8.05
Name	HSIO6NB0	HSIO6PB0/CCC_NE_CLKIN_N_11	HSIO12NB0
BALL	V22	W22	V19
DEFAULT	MSS GPIO_2[0]	MSS GPIO_2[1]	MSS GPIO_2[2]
	User LED 0	User LED 1	User LED 2
GPIOS	MSS GPIO_2[0]	MSS GPIO_2[1]	MSS GPIO_2[2]
	User LED 0	User LED 1	User LED 2
ROBOTICS	MSS GPIO_2[0]	MSS GPIO_2[1]	MSS GPIO_2[2]
	User LED 0	User LED 1	User LED 2

P8.06-P8.09

Pin	P8.06	P8.07	P8.08	P8.09
Name	HSIO12PB0/CLKIN_N_9/CCC_NE_CLKIN_N_9	HSIO30NB0	HSIO30PB0/CLKIN_N_3/CCC_NW_CLKIN_N_3	HSIO8NB0
BALL	V20	V15	V14	V21
DEFAULT	MSS GPIO_2[3]	MSS GPIO_2[4]	MSS GPIO_2[5]	MSS GPIO_2[6]
	User LED 3	User LED 4	User LED 5	User LED 6
GPIOS	MSS GPIO_2[3]	MSS GPIO_2[4]	MSS GPIO_2[5]	MSS GPIO_2[6]
	User LED 3	User LED 4	User LED 5	User LED 6
ROBOTICS	MSS GPIO_2[3]	MSS GPIO_2[4]	MSS GPIO_2[5]	MSS GPIO_2[6]
	User LED 3	User LED 4	User LED 5	User LED 6

P8.10-P8.13

Pin	P8.10	P8.11	P8.12	P8.13
Name	HSIO8PB0/CCC_NE_CLKIN_N_10/CCC_NE_PLL0_OUT1	HSIO7NB0	HSIO7PB0/CCC_NE_PLL0_OUT1	GPIO47PB1
BALL	W21	Y21	Y20	B10
DEFAULT	MSS GPIO_2[7]	MSS GPIO_2[8]	MSS GPIO_2[9]	core_pwm[1] @ 0x41500000
	User LED 7	User LED 8	User LED 9	PWM_2:1
GPIOS	MSS GPIO_2[7]	MSS GPIO_2[8]	MSS GPIO_2[9]	MSS GPIO_2[10]
	User LED 7	User LED 8	User LED 9	User LED 10
ROBOTICS	MSS GPIO_2[7]	MSS GPIO_2[8]	MSS GPIO_2[9]	core_pwm[1] @ 0x41500000
	User LED 7	User LED 8	User LED 9	PWM_2:1

P8.14-P8.16

Pin	P8.14	P8.15	P8.16
Name	GPIO47NB1	HSIO34NB0	HSIO34PB0/CCC_NW_CLKIN_N_1
BALL	B9	T12	U12
DEFAULT	MSS GPIO_2[11]	MSS GPIO_2[12]	MSS GPIO_2[13]
	User LED 11	GPIO	GPIO
GPIOS	MSS GPIO_2[11]	MSS GPIO_2[12]	MSS GPIO_2[13]
	User LED 11	GPIO	GPIO
ROBOTICS	MSS GPIO_2[11]	MSS GPIO_2[12]	MSS GPIO_2[13]
	User LED 11	GPIO	GPIO

P8.17-P8.19

Pin	P8.17	P8.18	P8.19
Name	HSIO29PB0	HSIO15PB0/DQS/CCC_NE_PLL1_OUT0	HSIO19NB0
BALL	W13	T16	W18
DEFAULT	MSS GPIO_2[14]	MSS GPIO_2[15]	core_pwm[0] @ 0x41500000
	GPIO	GPIO	PWM_2:0
GPIOS	MSS GPIO_2[14]	MSS GPIO_2[15]	MSS GPIO_2[16]
	GPIO	GPIO	GPIO
ROBOTICS	MSS GPIO_2[14]	MSS GPIO_2[15]	core_pwm[0] @ 0x41500000
	GPIO	GPIO	PWM_2:0

P8.20-P8.22

Pin	P8.20	P8.21	P8.22
Name	HSIO15NB0/DQS	HSIO9PB0/DQS/CCC_NE_PLL0_OUT0	HSIO9NB0/DQS
BALL	R16	AA21	AA22
DEFAULT	MSS GPIO_2[17]	MSS GPIO_2[18]	MSS GPIO_2[19]
	GPIO	GPIO	GPIO
GPIOS	MSS GPIO_2[17]	MSS GPIO_2[18]	MSS GPIO_2[19]
	GPIO	GPIO	GPIO
ROBOTICS	MSS GPIO_2[17]	MSS GPIO_2[18]	MSS GPIO_2[19]
	GPIO	GPIO	GPIO

P8.23-P8.26

Pin	P8.23	P8.24	P8.25	P8.26
Name	HSIO18PB0/CLKIN_N_7	HSIO18NB0	HSIO16PB0	GPIO49NB1
BALL	AB18	AA18	V17	A12
DEFAULT	MSS GPIO_2[20]	MSS GPIO_2[21]	MSS GPIO_2[22]	MSS GPIO_2[23]
	GPIO	GPIO	GPIO	GPIO
GOIOS	MSS GPIO_2[20]	MSS GPIO_2[21]	MSS GPIO_2[22]	MSS GPIO_2[23]
	GPIO	GPIO	GPIO	GPIO
ROBOTICS	MSS GPIO_2[20]	MSS GPIO_2[21]	MSS GPIO_2[22]	MSS GPIO_2[23]
	GPIO	GPIO	GPIO	GPIO

P8.27-P8.29

Pin	P8.27	P8.28	P8.29
Name	GPIO49PB1/CLKIN_S_5	GPIO51NB1	GPIO51PB1/CLKIN_S_6
BALL	A13	B14	B13
DEFAULT	MSS GPIO_2[24]	MSS GPIO_2[25]	MSS GPIO_2[26]
	GPIO	GPIO	GPIO
GPIOS	MSS GPIO_2[24]	MSS GPIO_2[25]	MSS GPIO_2[26]
	GPIO	GPIO	GPIO
ROBOTICS	MSS GPIO_2[24]	MSS GPIO_2[25]	MSS GPIO_2[26]
	GPIO	GPIO	GPIO

P8.30-P8.32

Pin	P8.30	P8.31	P8.32
Name	GPIO50NB1/DQS	GPIO50PB1/DQS	GPIO53NB1
BALL	D14	D13	B15
DEFAULT	MSS GPIO_2[27]	core_gpio[0] @ 0x41100000	core_gpio[1] @ 0x41100000
	GPIO	GPIO	GPIO
GPIOS	MSS GPIO_2[27]	core_gpio[0] @ 0x41100000	core_gpio[1] @ 0x41100000
	GPIO	GPIO	GPIO
ROBOTICS	MSS GPIO_2[27]	core_gpio[0] @ 0x41100000	core_gpio[1] @ 0x41100000
	GPIO	GPIO	GPIO

P8.33-P8.35

Pin	P8.33	P8.34	P8.35
Name	GPIO53PB1/CLKIN_S_7	GPIO52NB1/LPRB_B	GPIO52PB1/LPRB_A
BALL	A15	C15	C14
DEFAULT	core_gpio[2] @ 0x41100000	core_gpio[3] @ 0x41100000	core_gpio[4] @ 0x41100000
	GPIO	GPIO	GPIO
GPIOS	core_gpio[2] @ 0x41100000	core_gpio[3] @ 0x41100000	core_gpio[4] @ 0x41100000
	GPIO	GPIO	GPIO
ROBOTICS	core_gpio[2] @ 0x41100000	core_gpio[3] @ 0x41100000	core_gpio[4] @ 0x41100000
	GPIO	GPIO	GPIO

P8.36-P8.38

Pin	P8.36	P8.37	P8.38
Name	GPIO37NB1	GPIO37PB1/CCC_SW_CLKIN_S_1	GPIO3NB1
BALL	B4	C4	C17
DEFAULT	core_gpio[5] @ 0x41100000	core_gpio[6] @ 0x41100000	core_gpio[7] @ 0x41100000
	GPIO	GPIO	GPIO
GPIOS	core_gpio[5] @ 0x41100000	core_gpio[6] @ 0x41100000	core_gpio[7] @ 0x41100000
	GPIO	GPIO	GPIO
ROBOTICS	core_gpio[5] @ 0x41100000	core_gpio[6] @ 0x41100000	core_gpio[7] @ 0x41100000
	GPIO	GPIO	GPIO

P8.39-P8.41

Pin	P8.39	P8.40	P8.41
Name	GPIO3PB1/CCC_SE_CLKIN_S_10/CCC_SE_PLL1_OUT0	GPIO5NB1	GPIO5PB1/CCC_SE_CLKIN_S_11
BALL	B17	B18	A18
DEFAULT	core_gpio[8] @ 0x41100000	core_gpio[9] @ 0x41100000	core_gpio[10] @ 0x41100000
	GPIO	GPIO	GPIO
GPIOS	core_gpio[8] @ 0x41100000	core_gpio[9] @ 0x41100000	core_gpio[10] @ 0x41100000
	GPIO	GPIO	GPIO
ROBOTICS	core_gpio[8] @ 0x41100000	core_gpio[9] @ 0x41100000	core_gpio[10] @ 0x41100000
	GPIO	GPIO	GPIO

P8.42-P8.44

Pin	P8.42	P8.43	P8.44
Name	GPIO36NB1	GPIO36PB1/CCC_SW_CLKIN_S_0	GPIO42NB1
BALL	D6	D7	D8
DEFAULT	core_gpio[11] @ 0x41100000	core_gpio[12] @ 0x41100000	core_gpio[13] @ 0x41100000
	GPIO	GPIO	GPIO
GPIOS	core_gpio[11] @ 0x41100000	core_gpio[12] @ 0x41100000	core_gpio[13] @ 0x41100000
	GPIO	GPIO	GPIO
ROBOTICS	core_gpio[11] @ 0x41100000	core_gpio[12] @ 0x41100000	core_gpio[13] @ 0x41100000
	GPIO	GPIO	GPIO

P8.45-P8.46

Pin	P8.45	P8.46
Name	GPIO42PB1	GPIO4PB1/CCC_SE_PLL1_OUT1
BALL	D9	D18
DEFAULT	core_gpio[14] @ 0x41100000	core_gpio[15] @ 0x41100000
	GPIO	GPIO
GPIOS	core_gpio[14] @ 0x41100000	core_gpio[15] @ 0x41100000
	GPIO	GPIO
ROBOTICS	core_gpio[14] @ 0x41100000	core_gpio[15] @ 0x41100000
	GPIO	GPIO

Connector P9

The following tables show the pinout of the **P9** expansion header. The gateware image is responsible for setting the function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **Name** row is the pin name on the processor.

The **BALL** row is the pin number on the processor.

The rows below **BALL** are the gateware setting for each pin.

NOTES:

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

P9.01-P9.05

P9.01	P9.02	P9.03	P9.04	P9.05
GND	GND	VCC_3V3	VCC_3V3	VDD_5V

P9.06-P9.10

P9.06	P9.07	P9.08	P9.10
VDD_5V	SYS_5V	SYS_5V	SYS_RSTN

Pin	P9.09
Name	HSIO19PB0
BALL	W19

P9.11-P9.13

Pin	P9.11	P9.12	P9.13
Name	GPIO38NB1/DQS	GPIO38PB1/DQS/CCC_SW_PLL1_OUT0	GPIO2NB1/DQS
BALL	B5	C5	D19
DEFAULT	MMUART4	core_gpio[1] @ 0x41200000	MMUART4
	UART4 RX	GPIO	UART4 TX
GPIOS	core_gpio[0] @ 0x41200000	core_gpio[1] @ 0x41200000	core_gpio[2] @ 0x41200000
	GPIO	GPIO	GPIO
ROBOTICS	~	core_gpio[0] @ 0x41200000	core_gpio[7] @ 0x41200000
	~	GPIO	GPIO

P9.14-P9.16

Pin	P9.14	P9.15	P9.16
Name	GPIO39PB1/CLKIN_S_2/CCC_SW_CLKIN_S_2/CCC_SW_PLL1_OUT0	GPIO40NB1	GPIO40PB1/CCC_SW_PLL1_OUT1
BALL	C6	A5	A6
DEFAULT	core_pwm[0] @ 0x41400000	core_gpio[4] @ 0x41200000	core_pwm[1] @ 0x41400000
	PWM_1:0	GPIO	PWM_1:1
GOIOS	core_gpio[3] @ 0x41200000	core_gpio[4] @ 0x41200000	core_gpio[5] @ 0x41200000
	GPIO	GPIO	GPIO
ROBOTICS	core_pwm[0] @ 0x41400000	core_gpio[1] @ 0x41200000	core_pwm[1] @ 0x41400000
	PWM_1:0	GPIO	PWM_1:1

P9.17-P9.19

Pin	P9.17	P9.18	P9.19
Name	GPIO44NB1/DQS	GPIO44PB1/DQS/CCC_SW_PLL0_OUT0	GPIO45PB1/CCC_SW_PLL0_OUT0
BALL	C9	C10	A10
DEFAULT	~	~	MSS I2C0
	~	~	I2C0 SCL
GPIOS	core_gpio[6] @ 0x41200000	core_gpio[7] @ 0x41200000	MSS I2C0
	GPIO	GPIO	I2C0 SCL
ROBOTICS	~	~	MSS I2C0
	~	~	I2C0 SCL

P9.20-P9.22

Pin	P9.20	P9.21	P9.22
Name	GPIO45NB1	GPIO43NB1	GPIO43PB1
BALL	A11	B8	A8
DEFAULT	MSS I2C0	~	~
	I2C0 SDA	~	~
GPIOS	MSS I2C0	core_gpio[8] @ 0x41200000	core_gpio[8] @ 0x41200000
	I2C0 SDA	GPIO	GPIO
ROBOTICS	MSS I2C0	~	~
	I2C0 SDA	~	~

P9.23-P9.25

Pin	P9.23	P9.24	P9.25
Name	GPIO48NB1	GPIO48PB1/CLKIN_S_4	GPIO41NB1
BALL	C12	B12	B7
DEFAULT	core_gpio[10] @ 0x41200000	~	core_gpio[12] @ 0x41200000
	GPIO	~	GPIO
GPIOS	core_gpio[10] @ 0x41200000	core_gpio[11] @ 0x41200000	core_gpio[12] @ 0x41200000
	GPIO	GPIO	GPIO
ROBOTICS	core_gpio[2] @ 0x41200000	~	core_gpio[3] @ 0x41200000
	GPIO	~	GPIO

P9.26-P9.28

Pin	P9.26	P9.27	P9.28
Name	GPIO41PB1/CLKIN_S_3/CCC_SW_CLKIN_S_3	GPIO46NB1	GPIO46PB1/CCC_SW_PLL0_OUT1
BALL	A7	D11	C11
DEFAULT	~	core_gpio[14] @ 0x41200000	~
	~	GPIO	~
GPIOS	core_gpio[13] @ 0x41200000	core_gpio[14] @ 0x41200000	core_gpio[15] @ 0x41200000
	GPIO	GPIO	GPIO
ROBOTICS	~	~	~
	~	~	~

P9.29-P9.31

Pin	P9.29	P9.30	P9.31
Name	GPIO1PB1/CLKIN_S_9/CCC_SE_CLKIN_S_9	GPIO1NB1	GPIO4NB1
BALL	F17	F16	E18
DEFAULT	~	core_gpio[17] @ 0x41200000	~
	~	GPIO	~
GPIOS	core_gpio[16] @ 0x41200000	core_gpio[17] @ 0x41200000	core_gpio[18] @ 0x41200000
	GPIO	GPIO	GPIO
ROBOTICS	~	core_gpio[5] @ 0x41200000	~
	~	GPIO	~

P9.32-P9.40

P9.32	P9.34
VDD_ADC	GND

P9.33	P9.35	P9.36	P9.37	P9.38	P9.39	P9.40
AIN4	AIN6	AIN5	AIN2	AIN3	AIN0	AIN1

P9.41-P9.42

Pin	P9.41	P9.42
Name	GPIO0PB1/CLKIN_S_8/CCC_SE_CLKIN_S_8/CCC_SE_PLL0_OUT0	GPIO0NB1
BALL	E15	E14
DEFAULT	core_gpio[19] @ 0x41200000	core_pwm[0] @ 0x41000000
	GPIO	PWM_0:0
GPIOS	core_gpio[19] @ 0x41200000	core_gpio[20] @ 0x41200000
	GPIO	GPIO
ROBOTICS	core_gpio[19] @ 0x41200000	~
	GPIO	~

P9.43-P9.46

P9.43	P9.44	P9.45	P9.46
GND	GND	GND	GND

9.5 Demos

Todo: We need a CSI capture demos

Todo: We need a cape compatibility layer demo

9.5.1 Flashing gateware and Linux image

Todo: This is the *hard* way! Special cables and FlashPros are not required when using the firmware we initially ship on the board. This tutorial should be rescripted as how to _unbrick_ your board. Also, we have other workarounds using software and GPIOs rather than FlashPros. Let's not put this in user's face as *the* experience when it is far more painful than using the *change-gateware.sh* script and "hold BOOT button when applying power" solutions we've created!

In this tutorial we are going to learn to flash the gateware image to FPGA and `sdcard.image` to eMMC storage.

Important: Additional hardware required:

1. FlashPro5/6 programmer
2. Tag connect TC2050-IDC-NL 10-Pin No-Legs Cable with Ribbon connector
3. TC2050-CLIP-3PACK Retaining CLIP board for TC2050-NL cables

Programming & Debug tools installation

To flash a gateware image to your BeagleV-Fire board you will require a FlashPro5/6 and FlashPro Express (FPExpress) tool which comes pre-installed as part of Libero SoC Design Suite. A standalone FlashPro Express tool is also available with MicroChip's Programming and Debug Tools package, which we are going to use for this tutorial. Below are the steps to install the software:

1. Download the zip for your operating system from Programming and Debug Tools page.
2. Unzip the file and in the unzipped folder you will find `launch_installer.sh` and `Program_Debug_v2023.1.bin`.
3. Execute the `launch_installer.sh` script to start the installation procedure.

```
[lorforlinux@fedora Program_Debug_v2023.1_lin] $ ./launch_installer.sh

No additional packages to install for installer usage

Requirement search complete.

See /tmp/check_req_installer608695.log for information.

Launch of installer
Preparing to install
Extracting the JRE from the installer archive...
Unpacking the JRE...
```

Note: It's recommended to install under *home/user/microchip* for linux users.

Enabling non-root user to access FlashPro

1. Download `60-openocd.rules`
2. Copy udev rule `sudo cp 60-openocd.rules /etc/udev/rules.d`
3. Trigger udevadm using `sudo udevadm trigger` or reboot the PC for the changes to take effect

Flashing gateware image

Note: content below is valid for beta testers only.

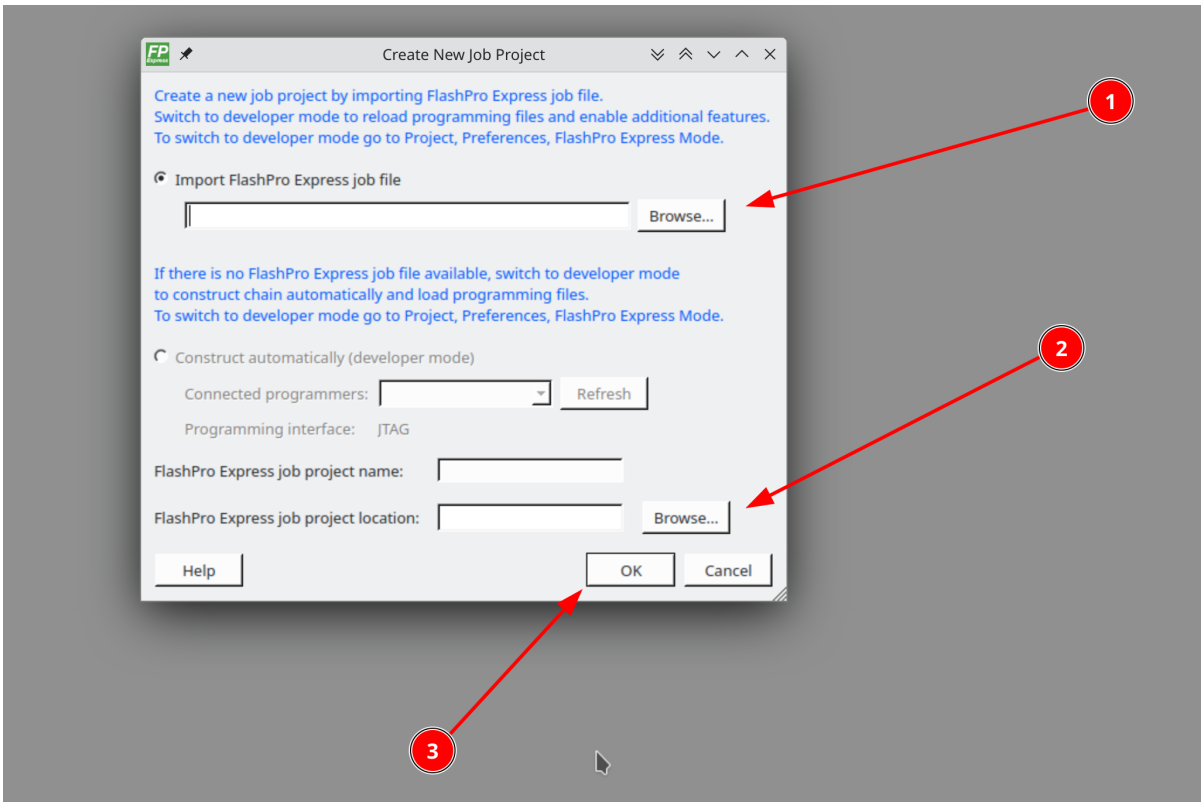
Launch FPExpress

1. Download `FlashProExpress.zip` file and unzip, it contains the `*.job` file which we need to create a new project in FPExpress.
2. Open up a terminal and go to `/home/user/microchip/Program_Debug_v202X.Y/Program_Debug_Tool/bin` which includes FPExpress tool.
3. Execute `./FPExpress` in terminal to start FlashPro Express software.

Create new project

Important: Make sure you have your FlashPro5/6 connected before you create a new project.

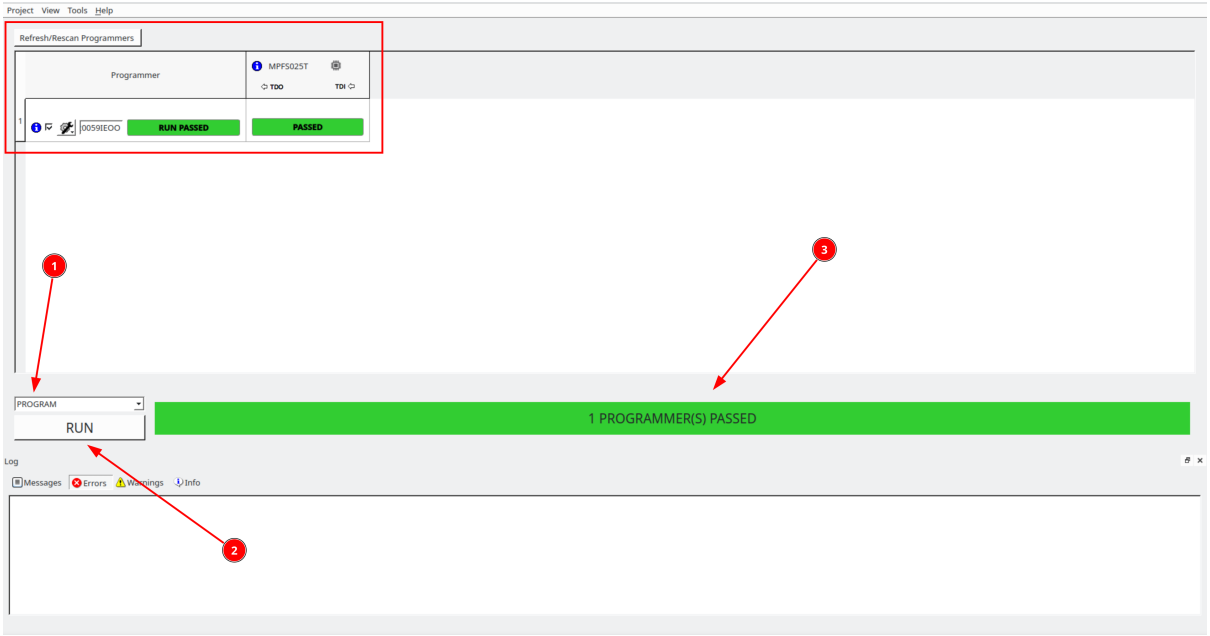
Press `CTRL+N` to create a file and you will see a pop-up window like shown below,



Follow the steps below as annotated in the image above:

1. Click on browse (1) button to select the job file.
2. Click on browse (2) button to select the project location.
3. Click ok button to finish.

If your FlashPro5/6 is connected properly you'll see the window shown below:



Following the annotation in the image above:

1. From drop-down select `Program` action
2. Click on `RUN` button
3. Shows the progress

If you see a lot of green color and progess bar says *PASSED*' then well done you have successfully flashed the gateware image on your BeagleV-Fire board.

Flashing eMMC

Connect to BeagleV-Fire UART debug port using a 3.3v USB to UART bridge.

Now you can run `tio <port>` in a terminal window to access the UART debug port connection. Once you are connected properly you can press the Reset button which will show you a progress bar like in the

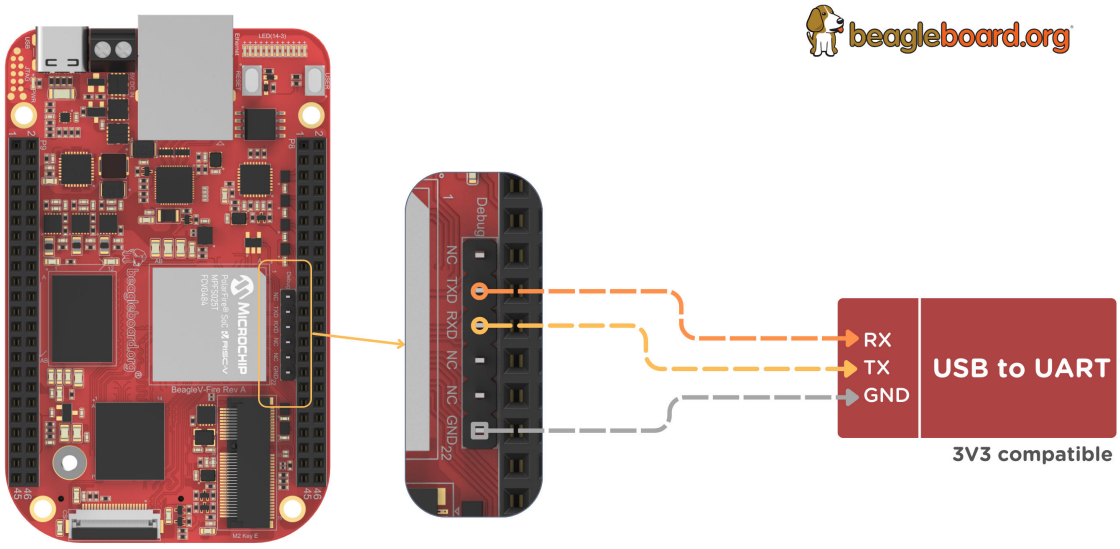
Once you see that progress bar on your screem you can start pressing any button (0-9/a-z) which whill stop the board from fully botting and you'll be able to access Hart Software Services (HSS) prompt. BeagleV-Fire's eMMC content is written by the Hart Software Services (HSS) using the `usbdmsc` command. The HSS `usbdmsc` command exposes the eMMC as a USB mass storage device USB type C connector.

Once you see >> you can execute the commands below:

1. `>> mmc`
2. `>> usbdmsc`

After executing the commands above your BeagleV-Fire's eMMC will be exposed as a mass storage device like shown in the image below:

Once your board is exposed as a mass storage device you can use Balena Etcher to flash the `sdcard.img` on your BeagleV-Fire's eMMC.




```
HSS: decompressing from eNVM to L2 Scratch ... Passed
DDR training ...
  60% [                                .....................]
```

```
HSS: decompressing from eNVM to L2 Scratch ... Passed
DDR training ... Passed ( 5306 ms)

---------------------------------
--        BeagleV-Fire         --
---------------------------------

[5.368717] PolarFire(R) SoC Hart Software Services (HSS) - version 0.99.36-BVF-250T-ES_0.0.1-229-g6d298cc
MPFS HAL version 2.2.104 / DDR Driver version 0.4.023 / Mi-V IHC version 0.1.1 / BOARD=bvf
(c) Copyright 2017-2022 Microchip FPGA Embedded Systems Solutions.

incorporating OpenSBI - version 1.2
(c) Copyright 2019-2022 Western Digital Corporation.

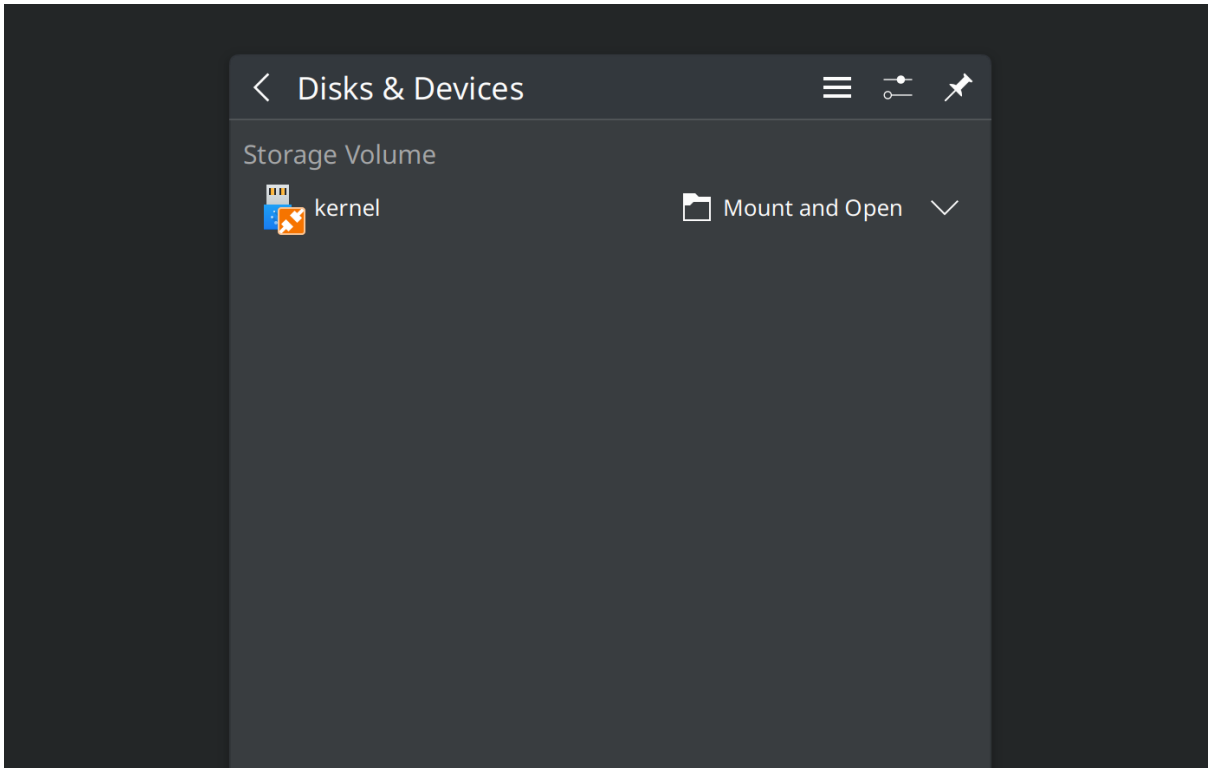
[5.401527] Build ID: cd5a966116e51ce2fe76385cf5d0d2d3531d7401
[5.408384] Built with the following tools:
 - riscv64-unknown-elf-gcc (xPack GNU RISC-V Embedded GCC (Microsemi SoftConsole build), 64-bit) 8.3.0
 - GNU ld (xPack GNU RISC-V Embedded GCC (Microsemi SoftConsole build), 64-bit) 2.32

[5.430258] NOTICE: Running from L2 Scratchpad

[5.435900] Serial Number:
72d51339518cdf7078fcde0ac038e871000000000000000000000000000000000000000000000000000000000000000000000000
[5.448573] Segment Configuration:
       Cached: SEG0_0: offset 0x0080000000, physical DDR 0x00000000
       Cached: SEG0_1: offset 0x1000000000, physical DDR 0x00000000
   Non-cached: SEG1_2: offset 0x00c0000000, physical DDR 0x00000000
   Non-cached: SEG1_3: offset 0x1400000000, physical DDR 0x00000000
Non-cached WCB: SEG1_4: offset 0x00d0000000, physical DDR 0x00000000
Non-cached WCB: SEG1_5: offset 0x1800000000, physical DDR 0x00000000
[5.489456] L2 Cache Configuration:
    L2-Scratchpad:  4 ways (512 KiB)
         L2-Cache:  8 ways (1024 KiB)
           L2-LIM:  4 ways (512 KiB)
[5.503951] DDR-Lo size is   32 MiB
[5.508204] UART error
[5.511590] DDR-Hi size is 1888 MiB
[5.535425] Design Info:
    Design Name: DEV_BOARD_TESTS_6CECBF8DE4F8CE
    Design Version: 00.00.1
[5.545841] Attempting to select eMMC ... Passed
Press a key to enter CLI, ESC to skip
Timeout in 1 second
.[5.806250] Character 49 pressed

[5.810416] Type HELP for list of commands
[5.815277] >>
[9.427945] >>
[9.595339] >>
```

```
[9.595339] >> mmc
[8647.667675] Selecting SDCARD/MMC (fallback) as boot source ...
[8647.674532] Attempting to select eMMC ... Passed
[8647.929915] >> usbdmsc
[8655.594694] initialize MMC
[8655.598426] Attempting to select eMMC ... Passed
[8655.853809] MMC - 512 byte pages, 512 byte blocks, 30621696 pages
Waiting for USB Host to connect... (CTRL-C to quit)
 . 0 bytes written, 0 bytes readUSB Host connected. Waiting for disconnect... (CTRL-C to quit)
 . 0 bytes written, 2300416 bytes read
```

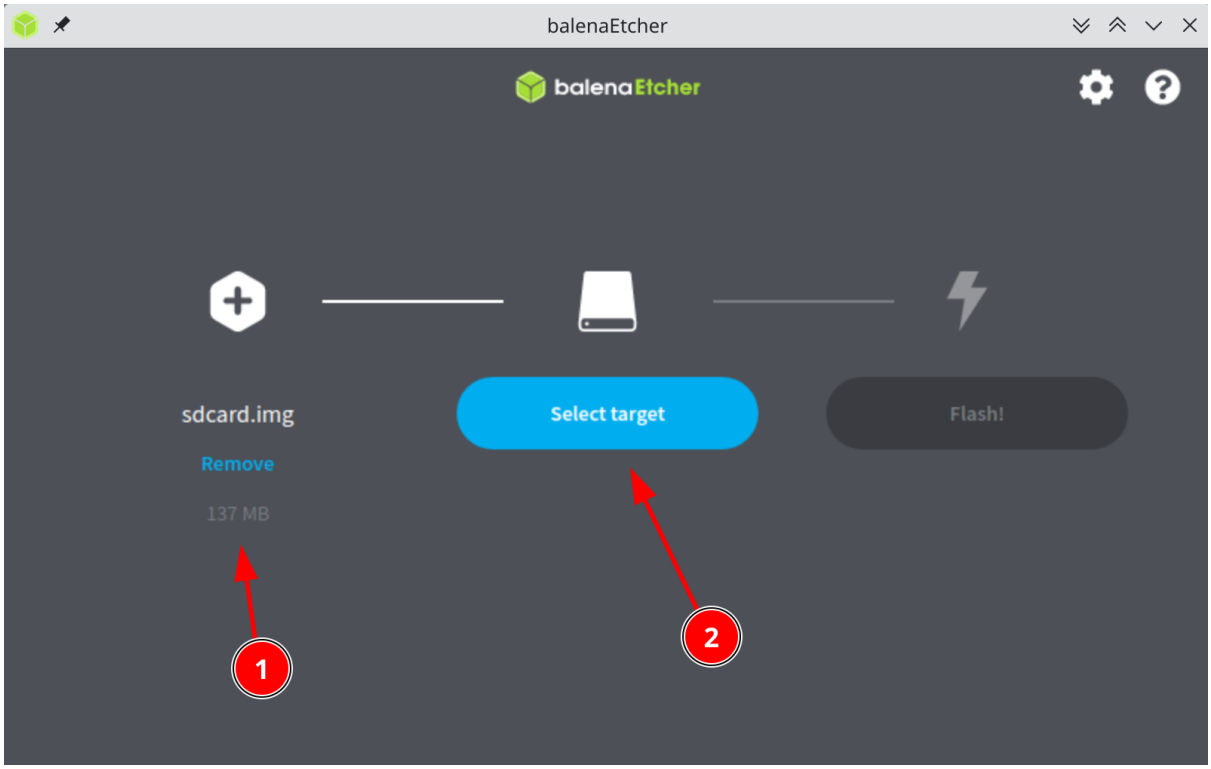


Select image

Select Target

Flash image

1. Select the `sdcard.img` file from your local drive storage.
2. Click on select target.



1. Select `MCC PolarFireSoC_msd` as target.
2. Click `Select (1)` to proceed.
1. Click on `Flash!` to flash the `sdcard.img` on BeagleV-Fire eMMC storage.

Congratulations! with that done you have fully updated BeagleV-Fire board with up to date gateware image on it's PolarFire SoC's FPGA Fabric and linux image on it's eMMC storage.

9.5.2 Microchip FPGA Tools Installation Guide

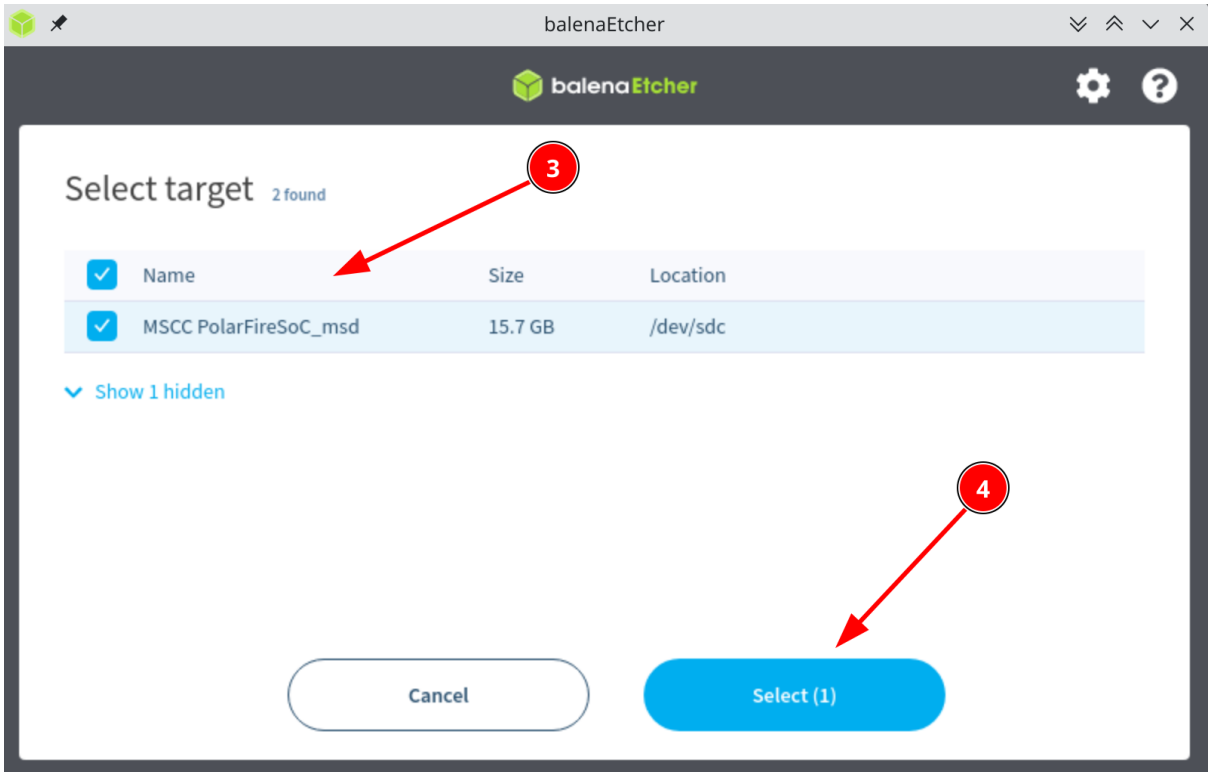
Instructions for installing the Microchip FPGA tools on a Ubuntu 20.04 desktop.

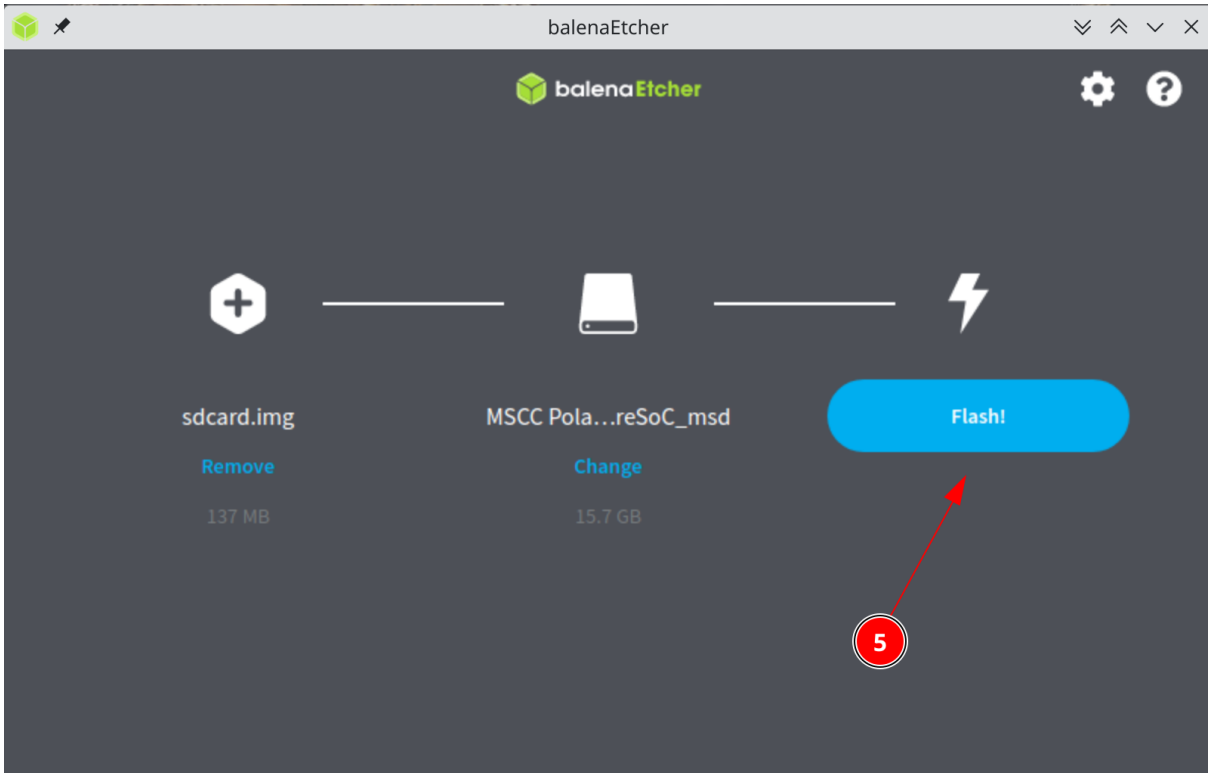
Important: We will be providing instances of Libero that you can run from git.beagleboard.org's gitlab-runners such that you do not need to install the tools on your local machine.

Todo: Make sure people know about the alternative and we provide links to details on that before we send them down this process.

Install Libero 2022.3

- Download installer from the Microchip's fpga and soc design tools section.


balenaEtcher
balenaEtcher
Select target
2 found
3
Name
Size
Location
MSCC PolarFireSoC_msd
15.7 GB
/dev/sdc
Show 1 hidden
4
Cancel
Select (1)


balenaEtcher
balenaEtcher
sdcard.img
Remove
137 MB
MSCC Pola…reSoC_msd
Change
15.7 GB
Flash!
5

- Install Libero

```
unzip Libero_SoC_v2022.3_lin.zip

cd Libero_SoC_v2022.3_lin/

./launch_installer.sh
```

Important: Do not use the default location suggested by the Libero installer. Instead of /usr/local/Microchip/Libero_SoC_v2022.3 install into ~/Microchip/Libero_SoC_v2022.3

Run the post installation script which will install missing packages:

```
sudo /home/<USER-NAME>/Microchip/Libero_SoC_v2022.3/Logs/req_to_install.sh
```

No need to run the FlashPro hardware installation scripts. This will be taken care of as part of the SoftConsole installation.

Install SoftConsole 2022.2

- Download intaller from Microchip website.

```
sudo chmod +x Microchip-SoftConsole-v2022.2-RISC-V-747-linux-x64-installer.
↪run

./Microchip-SoftConsole-v2022.2-RISC-V-747-linux-x64-installer.run
```

Accept the license, Click Forward, Finish.

Perform the post installation steps as described in the html file opened when you click Finish.

Important: Please pay special attention to the "Enabling non-root user to access FlashPro" section of the post-installation instructions. This will actually allow you to program the board using Libero.

Install the Libero licensing daemon

Download the 64 bit Licensing Daemons from the Microchip's daemons section

- Linux_Licensing_Daemon_11.16.1_64-bit.tar.gz
- Windows_Licensing_Daemon_11.16.1_64-bit.zip

Copy the downloaded file to the Microchip directory within your home directory and untar it.

```
cd ~/Microchip

tar -xvf Linux_Licensing_Daemon_11.16.1_64-bit.tar.gz
```

Install the Linux Standard Base:

```
sudo apt-get update

sudo apt-get -y install lsb
```

Request a Libero Silver license

- Visit microchip's fpga and soc design tool licensing page

- Click on Register a free license button and Register or login.
- Click “Request Free License” and choose “Libero Silver 1Yr Floating License for Windows/Linux Server” from the list.
- Enter you MAC address and click register.

Note: A MAC address looks something like 12:34:56::78:ab:cd when you use the “ip address” command to find out its value on your Linux machine. However, you need to enter it as 123456abcd in this dialog box.

You will get an email with a license.dat file. Copy it into the ~/Microchip/license directory. Edit the License.dat file to replace the <put.hostname.here> string with... localhost.

Execute tool setup script

Download the script:

Listing 9.1: Libero environment and license setup script

```
#!/bin/bash


#=============================================================================
# Edit the following section with the location where the following tools are
# installed:
#   - SoftConsole (SC_INSTALL_DIR)
#   - Libero (LIBERO_INSTALL_DIR)
#   - Licensing daemon for Libero (LICENSE_DAEMON_DIR)

#=============================================================================
export SC_INSTALL_DIR=/home/$USER/Microchip/SoftConsole-v2022.2-RISC-V-747
export LIBERO_INSTALL_DIR=/home/$USER/Microchip/Libero_SoC_v2023.2
export LICENSE_DAEMON_DIR=/home/$USER/Microchip/Linux_Licensing_Daemon
export LICENSE_FILE_DIR=/home/$USER/Microchip/license


#=============================================================================
# The following was tested on Ubuntu 20.04 with:
#   - Libero 2023.2
#   - SoftConsole 2022.2

#=============================================================================

#
# SoftConsole
#
export PATH=$PATH:$SC_INSTALL_DIR/riscv-unknown-elf-gcc/bin
export FPGENPROG=$LIBERO_INSTALL_DIR/Libero/bin64/fpgenprog

#
# Libero
#
export PATH=$PATH:$LIBERO_INSTALL_DIR/Libero/bin:$LIBERO_INSTALL_DIR/Libero/
bin64
export PATH=$PATH:$LIBERO_INSTALL_DIR/Synplify/bin
export PATH=$PATH:$LIBERO_INSTALL_DIR/Model/modeltech/linuxacoem
export LOCALE=C
export LD_LIBRARY_PATH=/usr/lib/i386-linux-gnu:$LD_LIBRARY_PATH

#
# Libero License daemon
```

(continues on next page)

(continued from previous page)

```
#
export LM_LICENSE_FILE=1702@localhost
export SNPSLMD_LICENSE_FILE=1702@localhost

$LICENSE_DAEMON_DIR/lmgrd -c $LICENSE_FILE_DIR/License.dat -l $LICENSE_FILE_
↪DIR/license.log
```

`setup-microchip-tools.sh`

Source the script:

```
. ./setup-microchip-tools.sh
```

Important: Do not forget the leading dot. It matters. You will need to run this every time you restart your machine.

You can then start Libero to open an existing Libero project.

```
libero
```

However you will more than likely want to use Libero to run a TCL script that will build a design for you.

```
libero SCRIPT:BUILD_A_DESIGN.tcl
```

9.5.3 Gateware Design Introduction

The PolarFire SoC device used on BeagleV-Fire is an SoC FPGA which includes a RISC-V processors subsystem and a PolarFire FPGA on the same die. The gateware configures the Microcprosessor subsystem's hardware and programs the FPGA with digital logic allowing customization of the use of BeagleV-Fire connectors.

Gateware Architecture

The diagram below is a simplified overview of the gateware's structure.

The overall gateware is made-up of several blocks, some of them interchangeable. These blocks are all clocked and reset by another "Clock and Resets" block not showed in the diagram for clarity.

Each gateware block is associated with one of BeagleV-Fire's connectors.

All gateware blocks have an AMBA APB target interface for software to access control and status registers defined by the gateware to operate digital logic defined by the gateware block. This is the software's control path into the gateware block.

Some gateware blocks also have an AMBA AXI target and/or source interfaces. The AXI interfaces are typically used to move high volume of data at high throughput in and out of DDR memory. For example, the M.2 gateware uses these interfaces to transfer data in and out of its PCIe root port.

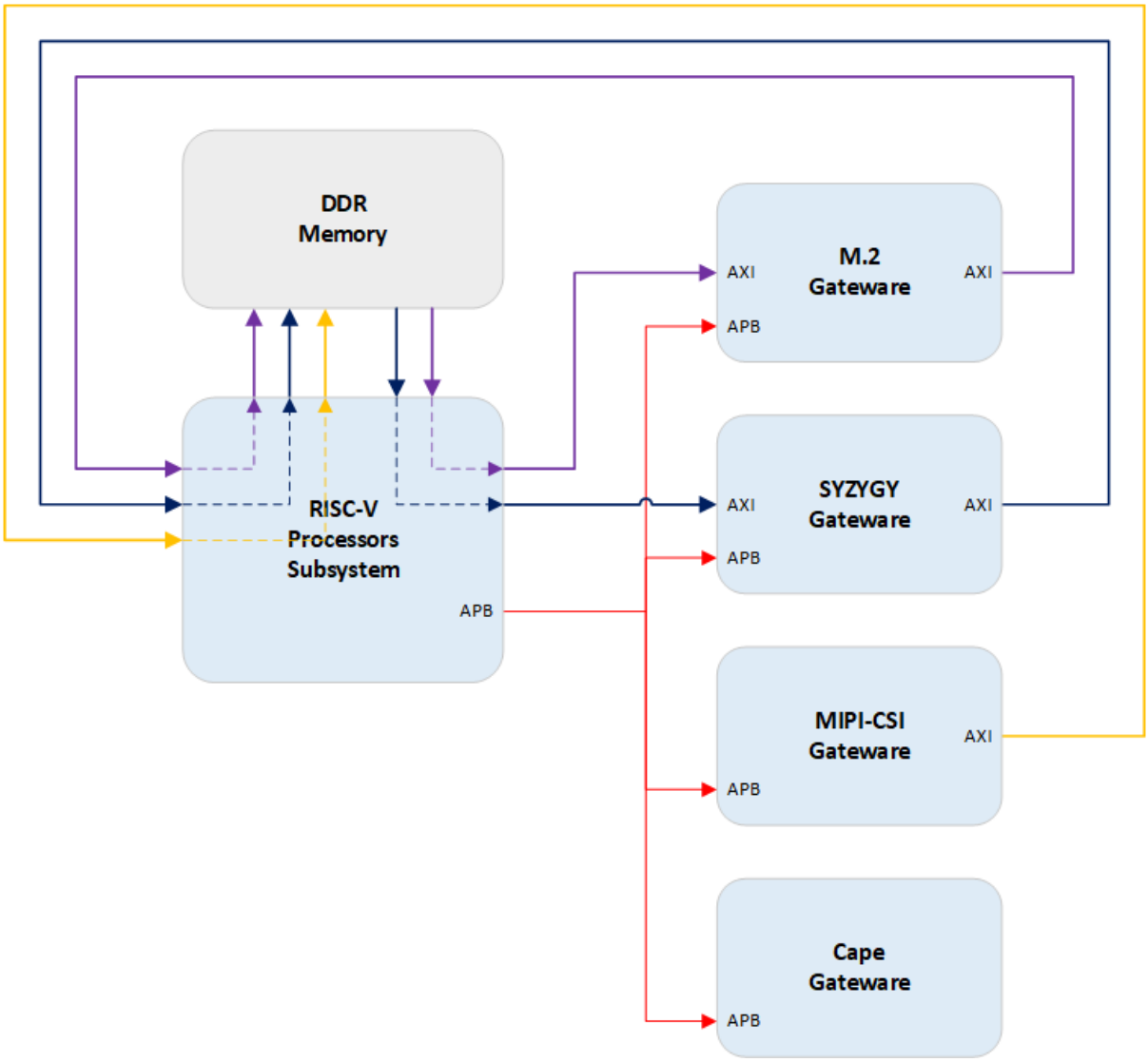
Cape Gateware The cape gateware handles the P8 and P9 connectors signals. This is where support for specific capes is implemented.

This is a very good place to start learning about FPGA and how to customize gateware.

SYZYGY Gateware

The SYZYGY gateware handles the high-speed connector signals. This connector includes:

- up to three transceivers capable of 12.7Gbps communications


DDR
Memory
RISC-V
Processors
Subsystem
APB
M.2
Gateware
AXI
APB
AXI
SYZYGY
Gateware
AXI
APB
AXI
MIPI-CSI
Gateware
APB
AXI
Cape
Gateware
APB

- One SGMII interface
- 10 high-speed I/Os
- Clock inputs

There is a lot of fun that can be had with this interface given its high-speed capabilities.

Please note that only two tranceivers can be used when the M.2 interface is enabled.

MIPI-CSI Gateware The MIPI gateware handles the signals coming from the camera interface.

Gateware for the MIPI-CSI interface is Work-In-Progress.

M.2 Gateware The M.2 gateware implements the PCIe interface used for Wi-Fi modules. It connects the processor subsystem to the PCIe controller associated with the tranceivers bank.

There is limited fun to have here. You either include this block or not in your bitstream.

The M.2 gateware uses one of the four available 12.7 Gbps transceivers. Only two out of the three SYZYGY tranceivers can be used when the M.2 is included in the bitstream. This gateware needs to omited from the bitstream if you want to use all three 12.7Gbps transceivers on the SYZYGY high-speed connector.

RISC-V Processors subsystem The RISC-V Processors Subsystem also includes some gateware mostly dealing with exposing AMBA bus interfaces for the other gateware blocks to attach to. It also handles immutable aspects of the gateware related to how some PolarFire-SoC signals are used to connect BeagleV-Fire peripherals such as the ADC and EEPROM. As such the RISC-V Processors Subsystem gateware is not intended to be customized.

9.5.4 How to retrieve BeagleV-Fire's gateware version

There are two methods to find out what gateware is programmed on a board.

Device Tree

The device tree overlays contains the list of gateware blocks included in the overall gateware design. You can retrieve that information using the following command:

```
tree /proc/device-tree/chosen/overlays/
```

This should give an output similar to the one below.

```
beagle@BeagleV:~$ tree /proc/device-tree/chosen/overlays/
/proc/device-tree/chosen/overlays/
├── name
├── PCIE-M2-GATEWARE
└── ROBOTICS-CAPE-GATEWARE

1 directory, 3 files
```

The gateware version can be retrieve by reading one of the overlay files. For example, the command:

```
cat /proc/device-tree/chosen/overlays/ROBOTICS-CAPE-GATEWARE
```

```
beagle@BeagleV:~$ cat /proc/device-tree/chosen/overlays/ROBOTICS-CAPE-GATEWARE
BVF-0.3.0-5-g3e0d3380beagle@BeagleV:~$
```

should result in:

where the result of a “git describe” command on the gateware repository is displayed. This provides the most recent tag on the gateware repository followed by information about additionanl commits if some exist. In the example above, the gateware was created from a gateware repository hash 3e0d338 which is 5 commits more recent than tag BVF-0.3.0.

Bootloader messages

The Hart Software Services display the gateware design name and design version retrieve from the FPGA at system start-up.

```
[5.528316] Design Info:
    Design Name: DEV_ROBOTICS_3E0D338F3C2574145
    Design Version: 02.00.1
```

The design name is the name of the build option selected when using the bitstream-builder to generate the bitstream. The number at the end of the design name is the hash of the gateware repository used to build the bitstream.

The design version is specified as part of the bitstream-builder build configuration option.

Please note that design name “BVF_GATEWARE” indicates that the bitstream used to program the board was generated directly from the gateware repositories scripts and not the bitstream-builder. You might see this when customizing the gateware. Seeing “BVF_GATEWARE” as the design name should be a warning sign that there is a disconnect between the hardware and software on your board.

9.6 Support

9.6.1 Certifications and export control

Export designations

Todo: update details

- HS: 8471504090
- US HS: 8543708800
- EU HS: 8471707000

Size and weight

Todo: update details

- Bare board dimensions: 86.38*54.61*18.8mm
- Bare board weight: 45.8g

- Full package dimensions: 140 x 100 x 40 mm
- Full package weight: 106g

9.6.2 Additional documentation

Hardware docs

For any hardware document like schematic diagram PDF, EDA files, issue tracker, and more you can checkout the BeagleV-Fire design repository.

Software docs

For BeagleV-Fire specific software projects you can checkout all the BeagleV-Fire project repositories group.

Support forum

For any additional support you can submit your queries on our forum, https://forum.beagleboard.org/tags/c/beaglev/15/fire

Pictures

9.6.3 Change History

Note: This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

Document Changes

For all changes, see https://git.beagleboard.org/docs/docs.beagleboard.io. Frozen releases tested against specific hardware and software revisions are noted below.

Table 9.5: BeagleV-Fire document change history

Rev	Changes	Date	By

Board Changes

For all changes, see https://git.beagleboard.org/beaglev-fire/beaglev-fire/. Versions released into production are noted below.

Table 9.6: BeagleV-Fire board change history

Rev	Changes	Date	By
A	Initial production version	2023-11-02	JK

Chapter 10

Capes

Note: This page is under development.

Todo: Get OSHWA certification for all of our capes and update the documentation to reflect that

Contributors

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

Note: Make sure to read and accept all the terms & condition provided in the *Terms & Conditions* page.

Use of either the boards or the design materials constitutes agreement to the T&C including any modifications done to the hardware or software solutions provided by beagleboard.org foundation.

Capes are add-on boards for BeagleBone or PocketBeagle families of boards. Using a Cape add-on board, you can easily add sensors, communication peripherals, and more.

Please visit BeagleBoard.org - Cape for the list of currently available Cape add-on boards.

In the BeagleBone board family, there are many variants, such as *BeagleBone Black*, *BeagleBone AI*, *BeagleBone AI-64* and compatibles such as SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SeeedStudio BeagleBone Green Gateway and more.

The *BeagleBone cape interface spec* enables a common set of device tree overlays and software to be utilized on each of these different BeagleBone boards.

Each hardware has different internal pin assignments and the number of peripherals in the SoC, but the device tree overlay absorbs these differences.

The user of the Cape add-on boards are essentially able to use it across the corresponding Boards without changing any code at all.

Find the instructions below on using each cape:

- *BeagleBoard.org BeagleBone Relay Cape*

10.1 BeagleBone cape interface spec

This page is a fork of BeagleBone cape interface spec page on elinux. This is the new official home.

10.1.1 Background and overview

Important: Resources

- See Device Tree: Supporting Similar Boards - The BeagleBone Example blog post on BeagleBoard.org
- See spreadsheet with pin header details
- See elinux.org Cape Expansion Headers for BeagleBone page
- See *BeagleBone Black System Reference Manual Connectors section*
- See *BeagleBone AI System Reference Manual Connectors section*
- See BeagleBone AI-64 System Reference Manual Connectors section

Note: Below, when mentioning "Black", this is true for all AM3358-based BeagleBone boards. "AI" is AM5729-based. "AI-64" is TDA4VM-based.

The device tree symbols for the BeagleBone Cape Compatibility Layer are provided in BeagleBoard-DeviceTrees at:

- Black: bbb-bone-buses.dtsi
- AI: bbai-bone-buses.dtsi
- AI-64: k3-j721e-beagleboneai-64-bone-buses.dtsi

The udev rules used to create the userspace symlinks for the BeagleBone Cape Compatibility Layer are provided in usr-customizations at:

More details can be found in *Methodology*.

Summary of cape interfaces

Note: Legend

- *D* : Digital general purpose input and output (GPIO)
- *I* : Inter-integrated circuit bus (I^2C) ports
- *S* : Serial peripheral interface (SPI) ports
- *U* : Universal asynchronous reciever/transmitter (UART) serial ports
- *C* : CAN
- *A* : Analog inputs
- *E* : PWM
- *Q* : Capture/EQEP
- *M* : MMC/SD/SDIO
- *B* : I2S/audio serial ports
- *L* : LCD
- *P* : PRU
- *Y* : ECAP

Table 10.1: Overall

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
USB D+	E1	E2	USB D-					
5V OUT	E3	E4	GND					
GND	1	2	GND		GND	1	2	GND
3V3 OUT	3	4	3V3 OUT		*D M*	3	4	*D M*
5V IN	5	6	5V IN		*D M C*	5	6	*D M C*
5V OUT	7	8	5V OUT		*D C*	7	8	*D C*
PWR BUT	9	10	RESET		*D C*	9	10	*D C*
D U	11	12	*D*		*D Q P*	11	12	*D Q P*
D U	13	14	*D E*		*D E*	13	14	*D*
D	15	16	*D E*		*D P*	15	16	*D P*
D I S	17	18	*D I S*		*D*	17	18	*D*
D I C	19	20	*D I C*		*D E*	19	20	*D M P*
D E S U	21	22	*D E S U*		*D M P*	21	22	*D M Q*
D S	23	24	*D I U C*		*D M*	23	24	*D M*
D P	25	26	*D I U C*		*D M*	25	26	*D*
D P Q	27	28	*D S P*		*D L P*	27	28	*D L P U*
D E S P	29	30	*D S P*		*D L P U*	29	30	*D L P*
D E S P	31	32	ADC VDD REF OUT		*D L*	31	32	*D L*
A	33	34	ADC GND		*D L Q*	33	34	*D E L*
A	35	36	*A*		*D L Q*	35	36	*D E L*
A	37	38	*A*		*D L U*	37	38	*D L U*
A	39	40	*A*		*D L P*	39	40	*D L P*
D P	41	42	*D Q S U P*		*D L P Q*	41	42	*D L P Q*
GND	43	44	GND		*D L P*	43	44	*D L P*
GND	45	46	GND		*D E L P*	45	46	*D E L P*

This is an alternate starting point template that should be fully encompasing with indexes. It is a bit overwhelming, so I'm not making it visible.

P9					P8			
Func-tions	odd	even	Func-tions		Func-tions	odd	even	Func-tions
USB D+	E1	E2	USB D-					
5V OUT	E3	E4	GND					
GND	1	2	GND		GND	1	2	GND
3V3 OUT	3	4	3V3 OUT		D M	3	4	D M
5V IN	5	6	5V IN		D M C4t	5	6	D M C4r
5V OUT	7	8	5V OUT		D C2r	7	8	D C2t
PWR BUT	9	10	RESET		D C3r	9	10	D C3t
D U4r	11	12	D		D P0o Q2b	11	12	D Q2a P0o
D U4t	13	14	D E1a		D E2b	13	14	D
D	15	16	D E1b		D P0i	15	16	D P0i
D I1c S00	17	18	D I1d S0o		D	17	18	D
C0r D I2c	19	20	C0t D I2d		D E2a	19	20	D M P1
D E0b S0i U2t	21	22	D E0a S0c U2r		D M P1	21	22	D M Q2b
D S01	23	24	C1r D I3c U1t		D M	23	24	D M
D P0	25	26	C1t D I3d U1r		D M	25	26	D
D P0 Q0b	27	28	D P0 S10		D L P1	27	28	D L P1 U6r
D E S1i P0	29	30	D P0 S1o		D L P1 U6t	29	30	D L P1
D E S1c P0	31	32	ADC VDD		D L	31	32	D L
A 4	33	34	ADC GND		D L Q1b	33	34	D E L
A 6	35	36	*A* 5		D L Q1a	35	36	D E L
A 2	37	38	*A* 3		D L U5t	37	38	D L U5r
A 0	39	40	*A* 1		D L P1	39	40	D L P1
D P0	41	42	D Q0a S11 U3t P0		D L P1 Q4a	41	42	D L P1 Q4b
GND	43	44	GND		D L P1	43	44	D L P1
GND	45	46	GND		D E L P1	45	46	D E L P1

10.1.2 Digital GPIO

The compatibility layer comes with simple reference nodes for attaching the Linux gpio-leds or gpio-keys to any cape header GPIO pin. This provides simple userspace general purpose input or output with various trigger modes.

The format followed for the gpio-leds nodes is **bone_led_P8_## / bone_led_P9_##**. The **gpio-leds** driver is used by these reference nodes internally and allows users to easily create compatible led nodes in overlays for Black, AI and AI-64.

Listing 10.1: Example device tree overlay to enable LED driver on header P8 pin 3

```
/dts-v1/;
/plugin/;

&bone_led_P8_03 {
    status = "okay";
}
```

In *Example device tree overlay to enable LED driver on header P8 pin 3*, it is possible to redefine the default label and other properties defined in the gpio-leds schema.

Table 10.2: GPIO pins

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
	3	4			P8_03	3	4	P8_04
	5	6			P8_05	5	6	P8_06
	7	8			P8_07	7	8	P8_08
	9	10			P8_09	9	10	P8_10
P9_11	11	12	P9_12		P8_11	11	12	P8_12
P9_13	13	14	P9_14		P8_13	13	14	P8_14
P9_15	15	16	P9_15		P8_15	15	16	P8_16
P9_17	17	18	P9_18		P8_17	17	18	P8_18
P9_19	19	20	P9_20		P8_19	19	20	P8_20
P9_21	21	22	P9_22		P8_21	21	22	P8_22
P9_23	23	24	P9_24		P8_23	23	24	P8_24
P9_25	25	26	P9_26		P8_25	25	26	P8_26
P9_27	27	28	P9_28		P8_27	27	28	P8_28
P9_29	29	30	P9_30		P8_29	29	30	P8_30
P9_31	31	32			P8_31	31	32	P8_32
	33	34			P8_33	33	34	P8_34
	35	36			P8_35	35	36	P8_36
	37	38			P8_37	37	38	P8_38
	39	40			P8_39	39	40	P8_40
P9_41	41	42	P9_42		P8_41	41	42	P8_42
	43	44			P8_43	43	44	P8_44
	45	46			P8_45	45	46	P8_46

Table 10.3: Bone GPIO LEDs interface

SYSFS link	Header pin	Black	AI	AI-64
/sys/class/leds/P8_03	P8_03	gpio1_6	gpio1_24	gpio0_20
/sys/class/leds/P8_04	P8_04	gpio1_7	gpio1_25	gpio0_48
/sys/class/leds/P8_05	P8_05	gpio1_2	gpio7_1	gpio0_33
/sys/class/leds/P8_06	P8_06	gpio1_3	gpio7_2	gpio0_34
/sys/class/leds/P8_07	P8_07	gpio2_2	gpio6_5	gpio0_15
/sys/class/leds/P8_08	P8_08	gpio2_3	gpio6_6	gpio0_14
/sys/class/leds/P8_09	P8_09	gpio2_5	gpio6_18	gpio0_17
/sys/class/leds/P8_10	P8_10	gpio2_4	gpio6_4	gpio0_16
/sys/class/leds/P8_11	P8_11	gpio1_13	gpio3_11	gpio0_60
/sys/class/leds/P8_12	P8_12	gpio1_12	gpio3_10	gpio0_59
/sys/class/leds/P8_13	P8_13	gpio0_23	gpio4_11	gpio0_89
/sys/class/leds/P8_14	P8_14	gpio0_26	gpio4_13	gpio0_75
/sys/class/leds/P8_15	P8_15	gpio1_15	gpio4_3	gpio0_61
/sys/class/leds/P8_16	P8_16	gpio1_14	gpio4_29	gpio0_62
/sys/class/leds/P8_17	P8_17	gpio0_27	gpio8_18	gpio0_3
/sys/class/leds/P8_18	P8_18	gpio2_1	gpio4_9	gpio0_4
/sys/class/leds/P8_19	P8_19	gpio0_22	gpio4_10	gpio0_88
/sys/class/leds/P8_20	P8_20	gpio1_31	gpio6_30	gpio0_76
/sys/class/leds/P8_21	P8_21	gpio1_30	gpio6_29	gpio0_30
/sys/class/leds/P8_22	P8_22	gpio1_5	gpio1_23	gpio0_5
/sys/class/leds/P8_23	P8_23	gpio1_4	gpio1_22	gpio0_31
/sys/class/leds/P8_24	P8_24	gpio1_1	gpio7_0	gpio0_6
/sys/class/leds/P8_25	P8_25	gpio1_0	gpio6_31	gpio0_35
/sys/class/leds/P8_26	P8_26	gpio1_29	gpio4_28	gpio0_51
/sys/class/leds/P8_27	P8_27	gpio2_22	gpio4_23	gpio0_71
/sys/class/leds/P8_28	P8_28	gpio2_24	gpio4_19	gpio0_72
/sys/class/leds/P8_29	P8_29	gpio2_23	gpio4_22	gpio0_73
/sys/class/leds/P8_30	P8_30	gpio2_25	gpio4_20	gpio0_74
/sys/class/leds/P8_31	P8_31	gpio0_10	gpio8_14	gpio0_32
/sys/class/leds/P8_32	P8_32	gpio0_11	gpio8_15	gpio0_26

continues on next page

Table 10.3 – continued from previous page

SYSFS link	Header pin	Black	AI	AI-64
/sys/class/leds/P8_33	P8_33	gpio0_9	gpio8_13	gpio0_25
/sys/class/leds/P8_34	P8_34	gpio2_17	gpio8_11	gpio0_7
/sys/class/leds/P8_35	P8_35	gpio0_8	gpio8_12	gpio0_24
/sys/class/leds/P8_36	P8_36	gpio2_16	gpio8_10	gpio0_8
/sys/class/leds/P8_37	P8_37	gpio2_14	gpio8_8	gpio0_106
/sys/class/leds/P8_38	P8_38	gpio2_15	gpio8_9	gpio0_105
/sys/class/leds/P8_39	P8_39	gpio2_12	gpio8_6	gpio0_69
/sys/class/leds/P8_40	P8_40	gpio2_13	gpio8_7	gpio0_70
/sys/class/leds/P8_41	P8_41	gpio2_10	gpio8_4	gpio0_67
/sys/class/leds/P8_42	P8_42	gpio2_11	gpio8_5	gpio0_68
/sys/class/leds/P8_43	P8_43	gpio2_8	gpio8_2	gpio0_65
/sys/class/leds/P8_44	P8_44	gpio2_9	gpio8_3	gpio0_66
/sys/class/leds/P8_45	P8_45	gpio2_6	gpio8_0	gpio0_79
/sys/class/leds/P8_46	P8_46	gpio2_7	gpio8_1	gpio0_80
/sys/class/leds/P9_11	P9_11	gpio0_30	gpio8_17	gpio0_1
/sys/class/leds/P9_12	P9_12	gpio1_28	gpio5_0	gpio0_45
/sys/class/leds/P9_13	P9_13	gpio0_31	gpio6_12	gpio0_2
/sys/class/leds/P9_14	P9_14	gpio1_18	gpio4_25	gpio0_93
/sys/class/leds/P9_15	P9_15	gpio1_16	gpio3_12	gpio0_47
/sys/class/leds/P9_16	P9_16	gpio1_19	gpio4_26	gpio0_94
/sys/class/leds/P9_17	P9_17	gpio0_5	gpio7_17	gpio0_28
/sys/class/leds/P9_18	P9_18	gpio0_4	gpio7_16	gpio0_40
/sys/class/leds/P9_19	P9_19	gpio0_13	gpio7_3	gpio0_78
/sys/class/leds/P9_20	P9_20	gpio0_12	gpio7_4	gpio0_77
/sys/class/leds/P9_21	P9_21	gpio0_3	gpio3_3	gpio0_39
/sys/class/leds/P9_22	P9_22	gpio0_2	gpio6_19	gpio0_38
/sys/class/leds/P9_23	P9_23	gpio1_17	gpio7_11	gpio0_10
/sys/class/leds/P9_24	P9_24	gpio0_15	gpio6_15	gpio0_13
/sys/class/leds/P9_25	P9_25	gpio3_21	gpio6_17	gpio0_127
/sys/class/leds/P9_26	P9_26	gpio0_14	gpio6_14	gpio0_12
/sys/class/leds/P9_27	P9_27	gpio3_19	gpio4_15	gpio0_46
/sys/class/leds/P9_28	P9_28	gpio3_17	gpio4_17	gpio1_11
/sys/class/leds/P9_29	P9_29	gpio3_15	gpio5_11	gpio0_53
/sys/class/leds/P9_30	P9_30	gpio3_16	gpio5_12	gpio0_44
/sys/class/leds/P9_31	P9_31	gpio3_14	gpio5_10	gpio0_52
/sys/class/leds/P9_33	P9_33	*n/a*	*n/a*	gpio0_50
/sys/class/leds/P9_35	P9_35	*n/a*	*n/a*	gpio0_55
/sys/class/leds/P9_36	P9_36	*n/a*	*n/a*	gpio0_56
/sys/class/leds/P9_37	P9_37	*n/a*	*n/a*	gpio0_57
/sys/class/leds/P9_38	P9_38	*n/a*	*n/a*	gpio0_58
/sys/class/leds/P9_39	P9_39	*n/a*	*n/a*	gpio0_54
/sys/class/leds/P9_40	P9_40	*n/a*	*n/a*	gpio0_81
/sys/class/leds/P9_41	P9_41	gpio0_20	gpio6_20	gpio1_0
/sys/class/leds/P9_42	P9_42	gpio0_7	gpio4_18	gpio0_123
/sys/class/leds/A15	A15[7]	gpio0_19	NA	NA

10.1.3 I²C

Cape interface specification provides I²C controller device links for userspace interaction and defined device tree entries for creating compatible overlays for any compliant board and OS image. The device tree node are named as **bone_i2c_#**.

[7] On BeagleBone Black, A15 is used to enable or disable the external CEC clock generation for the HDMI framer.

I²C pins

Table 10.4: I2C pins

P9			
Functions	odd	even	Functions
1 SCL	17	18	1 SDA
2 SCL	19	20	2 SDA
4 SCL[4][5]	21	22	4 SDA?[5]
	23	24	3 SCL[3]
	25	26	3 SDA[3]

I²C port mapping

Table 10.5: I2C port mapping

Links	DT symbol	Black	AI	AI-64	SCL	SDA	Overlay
/dev/bone/i2c/0	bone_i2c_0	I2C0	I2C1	WKUP_I2C0	On-board		BONE-I2C0
/dev/bone/i2c/1	bone_i2c_1	I2C1	I2C5	MAIN_I2C6	P9.17	P9.18	BONE-I2C1
/dev/bone/i2c/2	bone_i2c_2	I2C2	I2C4	MAIN_I2C3	P9.19	P9.20	BONE-I2C2
/dev/bone/i2c/3	bone_i2c_3	I2C1	I2C3	MAIN_I2C4	P9.24	P9.26	BONE-I2C3
/dev/bone/i2c/4	bone_i2c_4	I2C2	*n/a*	MAIN_I2C3	P9.21	P9.22	BONE-I2C4

Important: In the case the same controller is used for 2 different bone bus nodes, usage of those nodes is mutually-exclusive.

Note: The provided pre-compiled overlays enable the I²C bus driver only, not a specific device driver. Either a custom overlay is required to load the device driver or usermode device driver loading can be performed, depending on the driver. See *Using I2C with Linux drivers* for information on loading I²C drivers from userspace.

I²C overlay example

Listing 10.2: Example device tree overlay to enable I2C driver

```
/dts-v1/;
/plugin/;

&bone_i2c_1 {
    status = "okay";
    accel@1c {
        compatible = "fsl,mma8453";
        reg = <0x1c>;
    };
}
```

In *Example device tree overlay to enable I2C driver*, you can specify what driver you want to load and provide any properties it might need.

- https://www.kernel.org/doc/html/v5.10/i2c/summary.html
- https://www.kernel.org/doc/html/v5.10/i2c/instantiating-devices.html#method-1-declare-the-i2c-devices-statically
- https://www.kernel.org/doc/Documentation/devicetree/bindings/i2c/

[4] Port 4 is mutually exclusive with port 2 on Black
[5] On Black and AI-64 only, not AI
[3] Port 3 is mutually exclusive with port 1 on Black

I²C userspace example

See *Using I2C with Linux drivers* for examples on using the userspace links to load appropriate Linux kernel drivers.

10.1.4 SPI

SPI bone bus nodes allow creating compatible overlays for Black, AI and AI-64.

Table 10.6: SPI pins

P9			
Functions	odd	even	Functions
0 CS0	17	18	0 SDO
	19	20	
0 SDI	21	22	0 CLK
0 CS1	23	24	
	25	26	
	27	28	1 CS0
1 SDI	29	30	1 SDO
1 CLK	31	32	
	33	34	
	35	36	
	37	38	
	39	40	
	41	42	1 CS1

Table 10.7: SPI port mapping

Bone bus	DT symbol	Black	AI	AI-64	SDO	SDI	CLK	CS	Overlay
/dev/bone/spi/0.0	bone_spi_0	SPI0	SPI2	MAIN_SPI6	P9.18	P9.21	P9.22	P9.17 (CS0)	BONE-SPI0_0
/dev/bone/spi/0.1								P9.23 (CS1)	BONE-SPI0_1
/dev/bone/spi/1.0	bone_spi_1	SPI1	SPI3	MAIN_SPI7	P9.30	P9.29	P9.31	P9.28 (CS0)	BONE-SPI1_0
/dev/bone/spi/1.1								P9.42 (CS1)	BONE-SPI1_1

Note: The provided pre-compiled overlays enable the "spidev" driver using the "rohm,dh2228fv" compatible string. See https://stackoverflow.com/questions/53634892/linux-spidev-why-it-shouldnt-be-directly-in-devicetree for more background. A custom overlay is required to overload the compatible string to load a non-spidev driver.

Todo: figure out if BONE-SPI0_0 and BONE-SPI0_1 can be loaded at the same time

Note: Some boards may implement CS using a GPIO.

Listing 10.3: Example device tree overlay to enable SPI driver

```
/dts-v1/;
/plugin/;

&bone_spi_0 {
    status = "okay";
     pressure@0 {
         compatible = "bosch,bmp280";
         reg = <0>;       /* CS0 */
         spi-max-frequency = <5000000>;
     };
}
```

In *Example device tree overlay to enable SPI driver*, you can specify what driver you want to load and provide any properties it might need.

- https://www.kernel.org/doc/html/v5.10/spi/spi-summary.html
- https://www.kernel.org/doc/Documentation/devicetree/bindings/spi/

10.1.5 UART

UART bone bus nodes allow creating compatible overlays for Black, AI and AI-64.

Table 10.8: UART pins

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
4 RX[6]	11	12				11	12	
4 TX[6]	13	14				13	14	
	15	16				15	16	
	17	18				17	18	
	19	20				19	20	
2 TX	21	22	2 RX			21	22	7 RX
	23	24	1 TX			23	24	
	25	26	1 RX			25	26	
	27	28				27	28	6 RX
	29	30			6 TX	29	30	
	31	32				31	32	
	33	34				33	34	7 TX
	35	36				35	36	
	37	38			5 TX	37	38	5 RX
	39	40				39	40	
	41	42	3 TX			41	42	

Important: RTSn and CTSn mappings are not compatible across boards in the family and are therefore not part of the cape specification.

Table 10.9: UART port mapping

Bone bus	DT symbol	Black	AI	AI-64	TX	RX	Overlay
/dev/bone/uart/0	bone_uart_0	UART0	UART1	MAIN_UART0	Console debug header pins		
/dev/bone/uart/1	bone_uart_1	UART1	UART10	MAIN_UART2	P9.24	P9.26	BONE-UART1
/dev/bone/uart/2	bone_uart_2	UART2	UART3	*n/a*	P9.21	P9.22	BONE-UART2
/dev/bone/uart/3	bone_uart_3	UART3	*n/a*	*n/a*	P9.42	*n/a*	BONE-UART3
/dev/bone/uart/4	bone_uart_4	UART4	UART5	MAIN_UART0[6]	P9.13	P9.11	BONE-UART4
/dev/bone/uart/5	bone_uart_5	UART5	UART8	MAIN_UART5	P8.37	P8.38	BONE-UART5
/dev/bone/uart/6	bone_uart_6	*n/a*	*n/a*	MAIN_UART8	P8.29	P8.28	BONE-UART6
/dev/bone/uart/7	bone_uart_7	*n/a*	*n/a*	MAIN_UART2	P8.34	P8.22	BONE-UART7

Important: In the case the same controller is used for 2 different bone bus nodes, usage of those nodes is mutually-exclusive.

10.1.6 CAN

CAN bone bus nodes allow creating compatible overlays for Black, AI and AI-64.

[6] This port is shared with the console UART on AI-64

Table 10.10: CAN pins

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
	5	6			4 TX	5	6	4 RX
	7	8			2 RX	7	8	2 TX
	9	10			3 RX	9	10	3 TX
	11	12				11	12	
	13	14				13	14	
	15	16				15	16	
	17	18				17	18	
0 RX	19	20	0 TX			19	20	
	21	22				21	22	
	23	24	1 RX			23	24	
	25	26	1 TX			25	26	

Table 10.11: CAN port mapping

Bone bus	Black	AI	AI-64	TX	RX	Overlays
/dev/bone/can/0	CAN0	*n/a*	MAIN_MCAN0	P9.20	P9.19	BONE-CAN0
/dev/bone/can/1	CAN1	CAN2	MAIN_MCAN4	P9.26	P9.24	BONE-CAN1
/dev/bone/can/2	*n/a*	CAN1[1]	MAIN_MCAN5	P8.08	P8.07	BONE-CAN2
/dev/bone/can/3	*n/a*	*n/a*	MAIN_MCAN6	P8.10	P8.09	BONE-CAN3
/dev/bone/can/4	*n/a*	*n/a*	MAIN_MCAN7	P8.05	P8.06	BONE-CAN4

10.1.7 ADC

Todo: We need a udev rule to make sure the ADC shows up at /dev/bone/adc! There's nothing for sure that IIO devices will show up in the same place.

Todo: I think we can also create symlinks for each channel based on which device is there, such that we can do /dev/bone/adc/Px_y

Todo: I believe a multiplexing IIO driver is the future solution

[1] BeagleBone AI rev A2 and later only

Table 10.12: ADC pins

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
USB D+	E1	E2	USB D-					
5V OUT	E3	E4	GND					
GND	1	2	GND		GND	1	2	GND
3V3 OUT	3	4	3V3 OUT		D M	3	4	D M
5V IN	5	6	5V IN		D M C4t	5	6	D M C4r
5V OUT	7	8	5V OUT		D C2r	7	8	D C2t
PWR BUT	9	10	RESET		D C3r	9	10	D C3t
D U4r	11	12	D		D P0o	11	12	D Q2a P0o
D U4t	13	14	D E1a		D E2b	13	14	D
D	15	16	D E1b		D P0i	15	16	D P0i
D I1c S00	17	18	D I1d S0o		D	17	18	D
C0r D I2c	19	20	C0t D I2d		D E2a	19	20	D M P1
D E0b S0i U2t	21	22	D E0a S0c U2r		D M P1	21	22	D M Q2b
D S01	23	24	C1r D I3c U1t		D M	23	24	D M
D P0	25	26	C1t D I3d U1r		D M	25	26	D
D P0 Q0b	27	28	D P0 S10		D L P1	27	28	D L P1 U6r
D E S1i P0	29	30	D P0 S1o		D L P1 U6t	29	30	D L P1
D E S1c P0	31	32	ADC VDD		D L	31	32	D L
A 4	33	34	ADC GND		D L Q1b	33	34	D E L
A 6	35	36	*A* 5		D L Q1a	35	36	D E L
A 2	37	38	*A* 3		D L U5t	37	38	D L U5r
A 0	39	40	*A* 1		D L P1	39	40	D L P1
D P0	41	42	D Q0a S11 U3t P0		D L P1	41	42	D L P1
GND	43	44	GND		D L P1	43	44	D L P1
GND	45	46	GND		D E L P1	45	46	D E L P1

Table 10.13: Bone ADC

Index	Header pin	Black/AI-64	AI
0	P9_39	in_voltage0_raw	in_voltage0_raw
1	P9_40	in_voltage1_raw	in_voltage1_raw
2	P9_37	in_voltage2_raw	in_voltage3_raw
3	P9_38	in_voltage3_raw	in_voltage2_raw
4	P9_33	in_voltage4_raw	in_voltage7_raw
5	P9_36	in_voltage5_raw	in_voltage6_raw
6	P9_35	in_voltage6_raw	in_voltage4_raw

Table 10.14: Bone ADC Overlay

Black	AI	AI-64	overlay
Internal	External (STMPE811)	TBD	BONE-ADC.dts

10.1.8 PWM

Todo: remove deep references to git trees

PWM bone bus nodes allow creating compatible overlays for Black, AI and AI-64. For the definitions, you can see bbai-bone-buses.dtsi#L415 & bbb-bone-buses.dtsi#L432

PWM pins

Table 10.15: PWM pins

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
	13	14	1 A		2 B	13	14	
	15	16	1 B			15	16	
	17	18				17	18	
	19	20			2 A	19	20	
0 B	21	22	0 A			21	22	

PWM port mapping

Table 10.16: Bone bus PWM

SYSFS link	DT symbol	Black	AI	AI-64	A	B	Overlay
/dev/bone/pwm/0	bone_pwm_0	PWM0	•	PWM1	P9.22	P9.21	BONE-PWM0
/dev/bone/pwm/1	bone_pwm_0	PWM1	PWM3	PWM2	P9.14	P9.16	BONE-PWM1
/dev/bone/pwm/2	bone_pwm_0	PWM2	PWM2	PWM0	P8.19	P8.13	BONE-PWM2

PWM overlay example

Listing 10.4: Example device tree overlay to enable PWM driver

```
/dts-v1/;
/plugin/;

&bone_pwm_0 {
    status = "okay";
}
```

In *Example device tree overlay to enable PWM driver*, you can specify what driver you want to load and provide any properties it might need.

- https://www.kernel.org/doc/html/v5.10/driver-api/pwm.html
- https://www.kernel.org/doc/Documentation/devicetree/bindings/pwm/

PWM userspace example

See *Fading an External LED* and *Motors* for examples on using the userspace links to use the PWMs.

10.1.9 TIMER PWM

TIMER PWM bone bus uses ti,omap-dmtimer-pwm driver, and timer nodes that allow creating compatible overlays for Black, AI and AI-64. For the timer node definitions, you can see bbai-bone-buses.dtsi#L449 & bbb-bone-buses.dtsi#L466.

Table 10.17: Bone TIMER PWMs

Bone bus	Header pin	Black	AI	overlay
/sys/bus/platform/devices/bone_pwm_0/	P9.14	timer6	timer10	BONE-TIMER_PWM_0.dts
/sys/bus/platform/devices/bone_pwm_1/	P8.19	timer4	timer11	BONE-TIMER_PWM_1.dts
/sys/bus/platform/devices/bone_pwm_2/	P8.13	timer7	timer12	BONE-TIMER_PWM_2.dts
/sys/bus/platform/devices/bone_pwm_3/	P9.21	•	timer13	BONE-TIMER_PWM_3.dts
/sys/bus/platform/devices/bone_pwm_4/	P9.29	timer5	timer14	BONE-TIMER_PWM_4.dts
/sys/bus/platform/devices/bone_pwm_5/	P9.22	•	timer15	BONE-TIMER_PWM_5.dts

10.1.10 Counter

A counter function for quadrature encoders is implemented with the EQEP peripheral.

Todo: Additional quadrature encoders can be implemented in PRU firmware.

Table 10.18: Counter pins

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
	11	12			2 B	11	12	2 A
	13	14				13	14	
	15	16			3 B	15	16	3 A
	17	18				17	18	
	19	20				19	20	
	21	22				21	22	
	23	24				23	24	
	25	26				25	26	
0 B	27	28				27	28	
	29	30				29	30	
	31	32				31	32	
	33	34			1 B	33	34	
	35	36			1 A	35	36	
	37	38				37	38	
	39	40				39	40	
	41	42	0 A			41	42	

On BeagleBone's without an eQEP on specific pins, consider using the PRU to perform a software counter function.

Table 10.19: Counter port mapping

SYSFS link	DT symbol	Black	AI	AI-64	A	B	STRB	INDX	Overlay
/dev/bone/counter/0	bone_counter_0	eQEP0	eQEP2	eQEP0	P9.42	P9.27	• Black/AI-64: P9.25 • AI: P8.06	• Black/AI-64: P9.41 • AI: P8.05	BONE-COUNTER0
/dev/bone/counter/1	bone_counter_1	eQEP1	eQEP0	eQEP1	P8.35	P8.33	• Black/AI-64: P8.32 • AI: P9.21	• Black/AI-64: P8.31 • AI: –	BONE-COUNTER1
/dev/bone/counter/2	bone_counter_2	eQEP2	eQEP1	–	P8.12	P8.11	• Black: P8.15 • AI: P8.18	• Black: P8.16 • AI: P9.15	BONE-COUNTER2
/dev/bone/counter/3	bone_counter_3	eQEP2	–	–	P8.41	P8.42	• Black: P8.40	• Black: P8.39	BONE-COUNTER3
/dev/bone/counter/4	bone_counter_4	–	–	–	P8.16	P8.15			BONE-COUNTER4

10.1.11 eCAP

Todo: This doesn't include any abstraction yet.

Table 10.20: ECAP pins

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
USB D+	E1	E2	USB D-					
5V OUT	E3	E4	GND					
GND	1	2	GND		GND	1	2	GND
3V3 OUT	3	4	3V3 OUT		D M	3	4	D M
5V IN	5	6	5V IN		D M C4t	5	6	D M C4r
5V OUT	7	8	5V OUT		D C2r	7	8	D C2t
PWR BUT	9	10	RESET		D C3r	9	10	D C3t
D U4r	11	12	D		D P0o	11	12	D Q2a P0o
D U4t	13	14	D E1a		D E2b	13	14	D
D	15	16	D E1b		D P0i	15	16	D P0i
D I1c S00	17	18	D I1d S0o		D	17	18	D
C0r D I2c	19	20	C0t D I2d		D E2a	19	20	D M P1
D E0b S0i U2t	21	22	D E0a S0c U2r		D M P1	21	22	D M Q2b
D S01	23	24	C1r D I3c U1t		D M	23	24	D M
D P0	25	26	C1t D I3d U1r		D M	25	26	D
D P0 Q0b	27	28	D P0 S10		D L P1	27	28	D L P1 U6r
D E S1i P0	29	30	D P0 S1o		D L P1 U6t	29	30	D L P1
D E S1c P0	31	32	ADC VDD		D L	31	32	D L
A 4	33	34	ADC GND		D L Q1b	33	34	D E L
A 6	35	36	*A* 5		D L Q1a	35	36	D E L
A 2	37	38	*A* 3		D L U5t	37	38	D L U5r
A 0	39	40	*A* 1		D L P1	39	40	D L P1
D P0	41	42	D Q0a S11 U3t P0		D L P1	41	42	D L P1
GND	43	44	GND		D L P1	43	44	D L P1
GND	45	46	GND		D E L P1	45	46	D E L P1

Table 10.21: Black eCAP PWMs

Bone bus	Header pin	peripheral	overlay
/sys/bus/platform/drivers/ecap/48302100.ecap	P9.42	eCAP0_in_PWM0_out	BBB-ECAP0.dts
/sys/bus/platform/drivers/ecap/48304100.ecap	P9.28	eCAP2_in_PWM2_out	BBB-ECAP2.dts

Table 10.22: AI eCAP PWMs

Bone bus	Header pin	peripheral	overlay
/sys/bus/platform/drivers/ecap/4843e100.ecap	P8.15	eCAP1_in_PWM1_out	BBAI-ECAP1.dts
/sys/bus/platform/drivers/ecap/48440100.ecap	P8.14	eCAP2_in_PWM2_out	BBAI-ECAP2.dts
/sys/bus/platform/drivers/ecap/48440100.ecap	P8.20	eCAP2_in_PWM2_out	BBAI-ECAP2A.dts
/sys/bus/platform/drivers/ecap/48442100.ecap	P8.04	eCAP3_in_PWM3_out	BBAI-ECAP3.dts
/sys/bus/platform/drivers/ecap/48442100.ecap	P8.26	eCAP3_in_PWM3_out	BBAI-ECAP3A.dts

10.1.12 MMC/SDIO

SDIO is popular to connect various WiFi modules. The interface can also be used to connect MMC cards.

Important: On Black, the SDIO pins are consumed by default by the on-board eMMC which must be disabled to utilize the SDIO capability. On AI and AI-64, the on-board eMMC does not consume these pins.

Table 10.23: Bone MMC/SD/SDIO/eMMC

Header pin	Description
P8.3	DAT6[9]
P8.4	DAT7 Page 472, 9
P8.5	DAT2
P8.6	DAT3
P8.20	CMD
P8.21	CLK
P8.22	DAT5 Page 472, 9
P8.23	DAT4 Page 472, 9
P8.24	DAT1
P8.25	DAT0

Table 10.24: Bone SDIO Overlay

Black	AI	overlay
MMC2	MMC3	BONE-SDIO

10.1.13 LCD

Table 10.25: 16bit LCD interface

Header pin	Black	AI	RGB565
P8_27	lcd_vsync	vout1__vsync	VSYNC
P8_28	lcd_pclk	vout1_pclk	PCLK
P8_29	lcd_hsync	vout1_hsync	HSYNC
P8_30	lcd_ac_bias_en	vout1_de	DE
P8_32	lcd_data15	vout1_d15	R4
P8_31	lcd_data14	vout1_d14	R3
P8_33	lcd_data13	vout1_d13	R2
P8_35	lcd_data12	vout1_d12	R1
P8_34	lcd_data11	vout1_d11	R0
P8_36	lcd_data10	vout1_d10	G5
P8_38	lcd_data9	vout1_d9	G4
P8_37	lcd_data8	vout1_d8	G3
P8_40	lcd_data7	vout1_d7	G2
P8_39	lcd_data6	vout1_d6	G1
P8_42	lcd_data5	vout1_d5	G0
P8_41	lcd_data4	vout1_d4	B4
P8_44	lcd_data3	vout1_d3	B3
P8_43	lcd_data2	vout1_d2	B2
P8_46	lcd_data1	vout1_d1	B1
P8_45	lcd_data0	vout1_d0	B0

Table 10.26: 16bit LCD interface Overlay

Black	AI	overlay
lcdc	dss	

10.1.14 I²S

Todo: Describe I²S and ALSA

[9] 8-bit eMMC support only on Black. AI and AI-64 provide 4-bit support only.

I²S pins

Table 10.27: I²S pins

P9					P8			
Functions	odd	even	Functions		Functions	odd	even	Functions
	11	12	0 SCK_IN			11	12	
	13	14				13	14	
	15	16				15	16	
	17	18				17	18	
	19	20				19	20	
	21	22				21	22	
	23	24				23	24	
0 HCLK	25	26				25	26	
0 WS_IN	27	28	0 SDO0			27	28	
0 WS	29	30	0 SDI0			29	30	
0 SCK	31	32			0 SDO1[8]	31	32	
	33	34			0 SDI1 Page 473, 8	33	34	

Important: I²S 0 is used by HDMI audio on Black.

I²S port mapping

Table 10.28: I²S port mapping

I2S	Header pin	Black	AI	AI-64
HCLK	P9.25	mcasp0_ahclkx	mcasp1_ahclkx	AUDIO_EXT_REFCLK2
SCLK	P9.31	mcasp0_aclkx	mcasp1_aclkx	MCASP0_ACLKX
SCLK_IN	P9.12	mcasp0_aclkr_mux3	mcasp1_aclkr	MCASP0_ACLKR
WS	P9.29	mcasp0_fsx	mcasp1_fsx	MCASP0_AFSX
WS_IN	P9.27	mcasp0_fsr	mcasp1_fsr	MCASP0_AFSR
SDO0	P9.28	mcasp0_axr2	mcasp1_axr11	MCASP0_AXR0
SDI0	P9.30	mcasp0_axr0	mcasp1_axr10	MCASP0_AXR1
SDO1	P8.31	mcasp0_axr1		
SDI1	P8.33	mcasp0_axr3		

Table 10.29: I²S port mapping link and overlay

Link	Overlays
/dev/bone/i2s/0	BONE-I2S0

I²S overlay example

10.1.15 PRU

The overlay situation for PRUs is a bit more complex than with other peripherals. The mechanism for loading, starting and stopping the PRUs can go through either UIO or RemoteProc.

- /dev/remoteproc/prussX-coreY (AM3358 X = “”, other x = “1|2”)

[8] Only available on Black.

Table 10.30: Bone PRU eCAP

Header Pin	Black	AI
P8.15	pr1_ecap0	pr1_ecap0
P8.32	•	pr2_ecap0
P9.42	pr1_ecap0	•

Table 10.31: AI PRU UART

UART	TX	RX	RTSn	CTSn	Overlays
PRU1 UART0	P8_31	P8_33	P8_34	P8_35	
PRU2 UART0	P8_43	P8_44	P8_45	P8_46	

Table 10.32: Bone PRU

Header Pin	Black	AI
P8.03	•	pr2_pru0 10
P8.04	•	pr2_pru0 11
P8.05	•	pr2_pru0 06
P8.06	•	pr2_pru0 07
P8.07	•	pr2_pru1 16
P8.08	•	pr2_pru0 20
P8.09	•	pr2_pru1 06
P8.10	•	pr2_pru1 15
P8.11	pr1_pru0 15 (Out)	pr1_pru0 04
P8.12	pr1_pru0 14 (Out)	pr1_pru0 03
P8.13	•	pr1_pru1 07
P8.14	•	pr1_pru1 09
P8.15	pr1_pru0 15 (In)	pr1_pru1 16
P8.16	pr1_pru0 14 (In)	pr1_pru1 18
P8.17	•	pr2_pru0 15

continues on next page

Table 10.32 – continued from previous page

Header Pin	Black	AI
P8.18	•	pr1_pru1 05
P8.19	•	pr1_pru1 06
P8.20	•	pr2_pru0 03
P8.21	•	pr2_pru0 02
P8.22	•	pr2_pru0 09
P8.23	•	pr2_pru0 08
P8.24	•	pr2_pru0 05
P8.25	•	pr2_pru0 04
P8.26	•	pr1_pru1 17
P8.27	•	pr2_pru1 17
P8.28	•	pr2_pru0 17
P8.29	•	pr2_pru0 18
P8.30	•	pr2_pru0 19
P8.31	•	pr2_pru0 11
P8.32	•	pr2_pru1 00
P8.33	•	pr2_pru0 10
P8.34	•	pr2_pru0 08
P8.35	•	pr2_pru0 09

continues on next page

Table 10.32 – continued from previous page

Header Pin	Black	AI
P8.36	•	pr2_pru0 07
P8.37	•	pr2_pru0 05
P8.38	•	pr2_pru0 06
P8.39	•	pr2_pru0 03
P8.40	•	pr2_pru0 04
P8.41	•	pr2_pru0 01
P8.42	•	pr2_pru0 02
P8.43	•	pr2_pru1 20
P8.44	•	pr2_pru0 00
P8.45	•	pr2_pru1 18
P8.46	•	pr2_pru1 19
P9.11	•	pr2_pru0 14
P9.13	•	pr2_pru0 15
P9.14	•	pr1_pru1 14
P9.15	•	pr1_pru0 5
P9.16	•	pr1_pru1 15
P9.17	•	pr2_pru1 09
P9.18	•	pr2_pru1 08

continues on next page

Table 10.32 – continued from previous page

Header Pin	Black	AI
P9.19	•	pr1_pru1 02
P9.20	•	pr1_pru1 01
P9.24	pr1_pru0 16 (In)	•
P9.25	pr1_pru0 07	pr2_pru1 05
P9.26	pr1_pru1 16 (In)	pr1_pru0 17
P9.27	pr1_pru0 05	pr1_pru1 11
P9.28	pr1_pru0 03	pr2_pru1 13
P9.29	pr1_pru0 01	pr2_pru1 11
P9.30	pr1_pru0 02	pr2_pru1 12
P9.31	pr1_pru0 00	pr2_pru1 10
P9.41	pr1_pru0 06	pr1_pru1 03
P9.42	pr1_pru0 04	pr1_pru1 10

10.1.16 Dynamic overlays

Todo: Document dynamic DT overlays

10.1.17 Dynamic pinmux control

Todo: Document dynamic pinmux control

10.1.18 Methodology

The methodology for applied in the kernel and software images to expose the software interfaces is to be documented here. The most fundamental elements are the device tree entries, including overlays, and udev rules.

Device Trees

Todo: Describe how the Device Trees expose symbols for reuse across boards

For every resource exposed on the cape headers, an indexed symbol should be provided in the base device tree.

udev rules

10-of-symlink.rules

```
#From: https://github.com/mvduin/py-uio/blob/master/etc/udev/rules.d/10-of-
↪symlink.rules
# allow declaring a symlink for a device in DT
ATTR{device/of_node/symlink}!="", \
        ENV{OF_SYMLINK}="%s{device/of_node/symlink}"

ENV{OF_SYMLINK}!="", ENV{DEVNAME}!="", \
        SYMLINK+="%E{OF_SYMLINK}", \
        TAG+="systemd", ENV{SYSTEMD_ALIAS}+="/dev/%E{OF_SYMLINK}"
```

TBD

```
# Also courtesy of mvduin
# create symlinks for gpios exported to sysfs by DT
SUBSYSTEM=="gpio", ACTION=="add", TEST=="value", ATTR{label}!="sysfs", \
                RUN+="/bin/mkdir -p /dev/bone/gpio", \
                RUN+="/bin/ln -sT '/sys/class/gpio/%k' /dev/bone/gpio/%s
↪{label}"
```

Verification

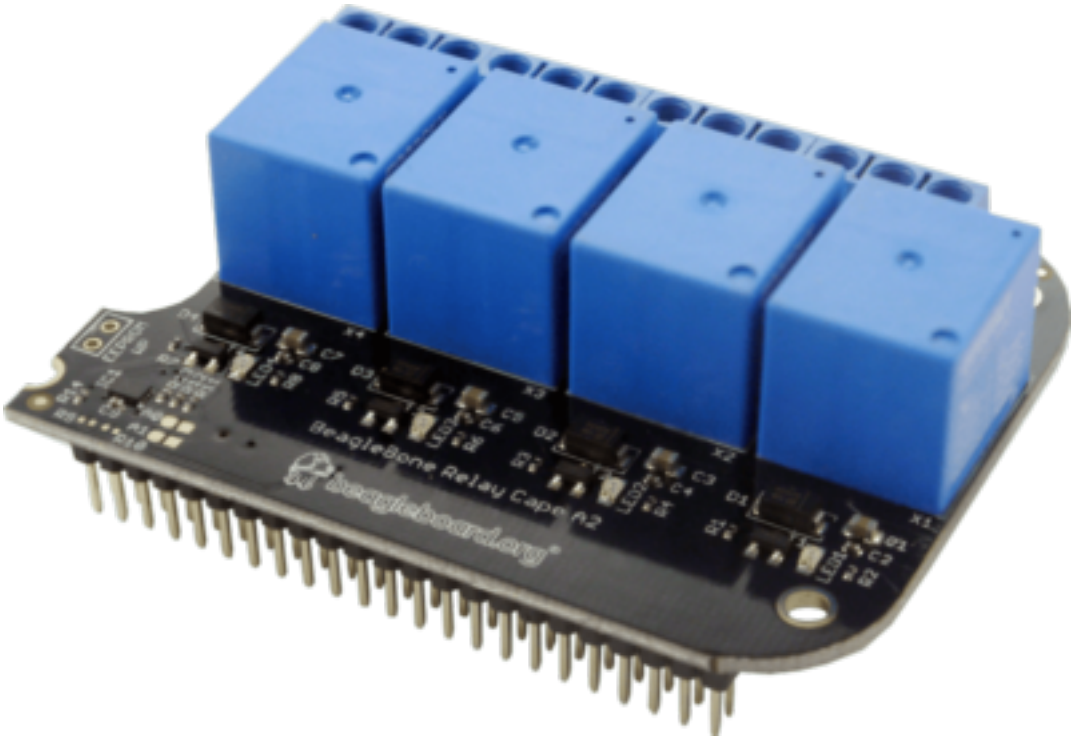
Todo: The steps used to verify all of these configurations is to be documented here. It will serve to document what has been tested, how to reproduce the configurations, and how to verify each major triannual release. All faults will be documented in the issue tracker.

10.1.19 References

- Device Tree: Supporting Similar Boards - The BeagleBone Example
- Google drive with summary of expansion signals on various BeagleBoard.org designs
- Beagleboard:Cape Expansion Headers

10.2 BeagleBoard.org BeagleBone Relay Cape

Relay Cape, as the name suggests, is a simple Cape with relays on it. It contains four relays, each of which can be operated independently from the BeagleBone.



- Order page
- Schematic

Note: The following describes how to use the device tree overlay under development. The description may not be suitable for those using older firmware.

10.2.1 Installation

No special configuration is required. When you plug the Relay Cape into your BeagleBoard, it will automatically be recognized by the Cape Universal function.

You can check to see if the Relay Cape is recognized with the following command.

```
ls /proc/device-tree/chosen/overlay
```

A list of currently loaded device tree overlays is displayed here. If you see *BBORG_RELAY-00A2.kernel* in this list, it has been loaded correctly.

If it is not loaded correctly, you can also load it directly by adding the following to the U-Boot options (which can be reflected by changing the file /boot/uEnv.txt) to reflect the text below.

```
uboot_overlay_addr0=BBORG_RELAY-00A2.dtbo
```

10.2.2 Usage

```
ls /sys/class/leds
```

The directory “relay1”, for instance, exists in the following directory. The LEDs can be controlled by modifying the files in its directory.

```
echo 1 > relay1/brightness
```

This allows you to adjust the brightness; entering 1 for brightness turns it ON while entering 0 turns it OFF.

The four relays can be changed individually by changing the number after “relay” in /sys/class/leds/relay.

10.2.3 Code to Get Started

Currently, using sysfs in .c files, libgpiod-dev/gpiod in .c files, and python3 files with the Relay Cape work well!

- For instance, a kernel that I found to work is kernel: *5.10.140-ti-r52*
- Another idea, an image I found that works is *BeagleBoard.org Debian Bullseye Minimal Image 2022-11-01*

There are newer images and kernels if you want to update and there are older ones in case you would like to go back in time to use older kernels and images for the Relay Cape. Please remember that older firmware will work differently on the BeagleBone Black or other related am335x SBC.

10.2.4 C Source with File Descriptors

You can name this file GPIO.c and use gcc to handle compiling the source into a binary like so:

gcc GPIO.c -o GPIO

```
/*

This is an example of programming GPIO from C using the sysfs interface on
a BeagleBone Black/BeagleBone Black Wireless or other am335x board with the␣
↪Relay Cape.

Use the Relay Cape attached to the BeagleBone Black for a change in seconds␣
↪and then exit with CTRL-C.

The original source can be found here by Mr. Tranter: https://github.com/
↪tranter/blogs/blob/master/gpio/part5/demo1.c

Jeff Tranter <jtranter@ics.com>

and...Seth. I changed the source a bit to fit the BeagleBone Black and Relay␣
↪Cape while using sysfs.

*/

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{

// Export the desired pin by writing to /sys/class/leds/relay1/brightness

    int fd = open("/sys/class/leds/relay1/brightness", O_WRONLY);
    if (fd == -1) {
        perror("Unable to open /sys/class/leds/relay1/brightness");
        exit(1);
    }

    fd = open("/sys/class/leds/relay1/brightness", O_WRONLY);
    if (fd == -1) {
```

(continues on next page)

(continued from previous page)

```
        perror("Unable to open /sys/class/leds/relay1/brightness");
        exit(1);
    }

// Toggle LED 50 ms on, 50ms off, 100 times (10 seconds)

    for (int i = 0; i < 100; i++) {
        if (write(fd, "1", 1) != 1) {
            perror("Error writing to /sys/class/leds/relay1/brightness");
            exit(1);
        }
        usleep(50000);

        if (write(fd, "0", 1) != 1) {
            perror("Error writing to /sys/class/leds/relay1/brightness");
            exit(1);
        }
        usleep(50000);
    }

    close(fd);

    // And exit
    return 0;
}
```

10.2.5 C Source with LibGPIOd-dev and File Descriptors

Also...if you are looking to dive into the new interface, libgpiod-dev/gpiod.h, here is another form of source that can toggle the same GPIO listed from the file descriptor.

One thing to note: *sudo apt install cmake*

1. mkdir GPIOd && cd GPIOd
2. nano LibGPIO.c
3. add the below source into the file LibGPIO.c

```
/*

Simple gpiod example of toggling a LED connected to a gpio line from
the BeagleBone Black Wireless and Relay Cape.
Exits with or without CTRL-C.

*/

// This source can be found here: https://github.com/tranter/blogs/blob/
↪master/gpio/part9/example.c
// It has been changed by me, Seth, to handle the RelayCape and BBBW Linux␣
↪based SiP SBC.

// kernel: 5.10.140-ti-r52
// image : BeagleBoard.org Debian Bullseye Minimal Image 2022-11-01

// type gpioinfo and look for this line: line 20: "P9_41B" "relay1" output␣
↪active-high [used]
// That line shows us the info. we need to make an educated decision on what␣
↪fd we will use, i.e. relay1.
// We will also need to locate which chipname is being utilized. For␣
↪instance: gpiochip0 - 32 lines:
```

(continues on next page)

(continued from previous page)

```
// #include <linux/gpio.h>
#include <gpiod.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    const char *chipname = "gpiochip0";
    struct gpiod_chip *chip;
    struct gpiod_line *lineLED;

int i, ret;

// Open GPIO chip
chip = gpiod_chip_open_by_name(chipname);
if (!chip) {
    perror("Open chip failed\n");
    return 1;
}

// Open GPIO lines
lineLED = gpiod_chip_get_line(chip, 20);
if (!lineLED) {
    perror("Get line failed\n");
    return 1;
}

// Open LED lines for output
ret = gpiod_line_request_output(lineLED, "relay1", 0);
if (ret < 0) {
    perror("Request line as output failed\n");
    return 1;
}

// Blink a LED
i = 0;
while (true) {
    ret = gpiod_line_set_value(lineLED, (i & 1) != 0);
    if (ret < 0) {
        perror("Set line output failed\n");
        return 1;
    }
    usleep(1000000);
    i++;
}

// Release lines and chip
gpiod_line_release(lineLED);
gpiod_chip_close(chip);
return 0;
}
```

4. mkdir build && touch CMakeLists.txt
5. In CMakeLists.txt, add these values and text via nano or your favorite editor!

```
cmake_minimum_required(VERSION 3.22)

project(gpiod LANGUAGES C)

add_executable(LibGPIO LibGPIO.c)
```

(continues on next page)

(continued from previous page)

```
target_link_libraries(LibGPIO gpiod)
```

6. cd build && cmake ..
7. make
8. ./LibGPIO

These are a few examples on how to use the RelayCape and am335x supported BeagleBone Black Wireless/BeagleBone Black SBCs.

Chapter 11

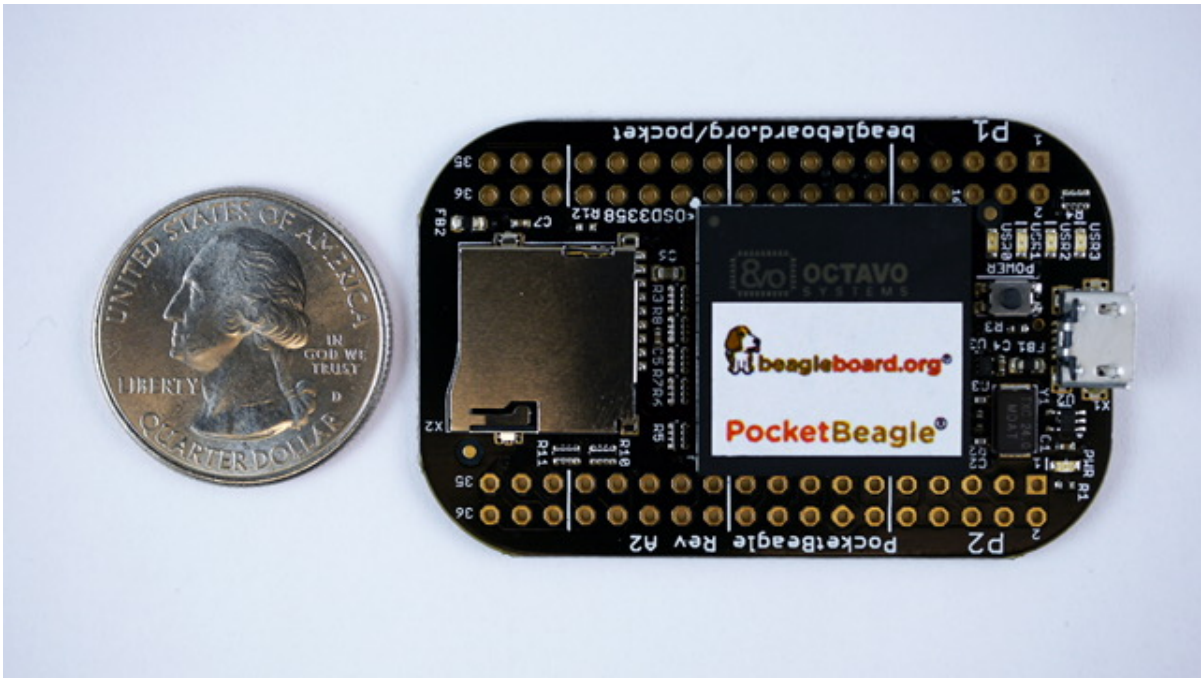
PocketBeagle

PocketBeagle is an ultra-tiny-yet-complete open-source USB-key-fob computer. PocketBeagle features an incredible low cost, slick design and simple usage, making PocketBeagle the ideal development board for beginners and professionals alike.



License Terms

- This documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page
- For export, emissions and other compliance, see *Support Information*



11.1 Introduction

This document is the **System Reference Manual** for PocketBeagle and covers its use and design. PocketBeagle is an ultra-tiny-yet-complete Linux-enabled, community-supported, open-source USB-key-fob-computer. PocketBeagle features an incredible low cost, slick design and simple usage, making it the ideal development board for beginners and professionals alike. Simply develop directly in a web browser providing you with a playground for programming and electronics. Exploring is made easy with several available libraries and tutorials with many more coming.

PocketBeagle will boot directly from a microSD card. Load a Linux distribution onto your card, plug your board into your computer and get started. PocketBeagle runs GNU.Linux, so you can leverage many different high-level programming languages and a large body of drivers that prevent you from needing to write a lot of your own software.

This design will keep improving as the product matures based on feedback and experience. Software updates will be frequent and will be independent of the hardware revisions and as such not result in a change in the revision number of the board. A great place to find out the latest news and projects for PocketBeagle is on the home page beagleboard.org/pocket

Important: Make sure you check the BeagleBoard.org docs repository for the most up to date information.

11.2 Change History

This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.



Fig. 11.1: PocketBeagle Home Page

11.2.1 Document Change History

Table 11.1: Change History

Rev	Changes	Date	By
A.x	Production Document	*December 7, 2017*	JK
0.0.5	Converted to .rst and gitlab hosting	*July 21, 2022*	DK

11.2.2 Board Changes

Table 11.2: Board History

Rev	Changes	Date	By
A1	Preliminary	*February 14, 2017*	JK
A2	Production. Fixed mikroBUS Click reset pins (made GPIO).	*September 22, 2017*	JK
A2a	Fixed label on P2_24. Was labeled GPIO48, should be GPIO44.	*November 7, 2017*	JK
A2b	Because there are 2 TI parts which have long lead-time, we made the following changes: 1. Use ESD discrete devices instead of integrated TVS TI: TPD4S012DRYR. 2. Change Logic IC TI SN74LVC1G07DCKR to Nexperia 74LVC1G07GV	*June 15, 2021*	JK

PocketBone

Upon the creation of the first, 27mm-by-27mm, Octavo Systems OSD3358 SIP, Jason did a hack two-layer board in EAGLE called “PocketBone” to drop the Beagle name as this was a totally unofficial effort not geared at being a BeagleBoard.org Foundation project. The board never worked because the 32kHz and 24MHz crystals were backwards and Michael Welling decided to pick it up and redo the design in KiCad as a four-layer board. Jason paid for some prototypes and this resulted in the first successful “PocketBone”, a fully-open-source 1-GHz Linux computer in a fitting into a mini-mint tin.

Rev A1

The Rev A1 of PocketBeagle was a prototype not released to production. A few lines were wrong to be able to control mikroBUS Click add-on board reset lines and they were adjusted.

Rev A2

The Rev A2 of PocketBeagle was released to production and launched at World MakerFaire 2017.

Known issues in rev A2:

Issue	Link
GPIO44 is incorrectly labelled as GPIO48	Issue #4

Rev A2B

Because 2 TI parts had a long lead time, we made the following changes:

Change # | Modification		Reference Designators	Part Type	Before value	After value)
1	Changed C2,C3 from 18pF to 22pF.	C2,C3	Cap Ceramic	18pF	22pF
2	Changed Y1 from 24MHz_18pF to 24MHz_22pF.	Y1	Crystal	24MHz_18pF	24MHz_22pF
3	Use ESD discrete devices(D1-D4) to replace U3.	U3	ESD Solution	integrated	ESD discrete devices(D1-D4)
4	Changed U2 from SN74LVC1G07DCKR to 74LVC1G07GV,125.	U2	Logic	SN74LVC1G07DCKR	74LVC1G07GV,125
5	The PCB Revision for this board is Rev A2b.	The PCB Revision for this board is Rev A2b.			

11.3 Connecting Up PocketBeagle

This section provides instructions on how to hook up your board. The most common scenario is tethering PocketBeagle to your PC for local development.

11.3.1 What's In the Package

In the package you will find two items as shown in figures below.

- PocketBeagle
- Getting Started instruction card with link to the support URL.

11.3.2 Connecting the board

This section will describe how to connect to the board. Information can also be found on the Quick Start Guide that came in the box. Detailed information is also available at beagleboard.org/getting-started

The board can be configured in several different ways, but we will discuss the most common scenario. Future revisions of this document may include additional configurations.

11.3.3 Tethered to a PC using Debian Images

In this configuration, you will need the following additional items:

- microUSB to USB Type A Cable



Fig. 11.2: PocketBeagle Package



Fig. 11.3: PocketBeagle Package Insert front



Fig. 11.4: PocketBeagle Package Insert back

- microSD card (>=4GB and <128GB)

The board is powered by the PC via the USB cable, no other cables are required. The board is accessed either as a USB storage drive or via a web browser on the PC. You need to use either Firefox or Chrome on the PC, IE will not work properly. Figure below shows this configuration.

In some instances, such as when additional add-on boards, or PocketCapes are connected, the PC may not be able to supply sufficient power for the full system. In that case, review the power requirements for the add-on board/cape; additional power may need to be supplied via the 5v input, but rarely is this the case.

Getting Started

The following steps will guide you to quickly download a PocketBeagle software image onto your microSD card and get started writing code.

1. Navigate to the Getting Started Page beagleboard.org/getting-started Follow along with the instructions and click on the link noted in Figure 5 below www.beagleboard.org/distros. You can also get to this page directly by going to bbb.io/latest

1. Download the latest image onto your computer by following the link to the latest image and click on the Debian image for Stretch IoT (non-GUI) for BeagleBone and PocketBeagle via microSD card. See Figure 6 below. This will download a `.img.xz` file into the downloads folder of your computer.

1. Transfer the image to a microSD card.

Download and install an SD card programming utility if you do not already have one. We like https://etcher.io/ for new users and so we show that one in the steps below. Go to your downloads folder and doubleclick on the .exe file and follow the on-screen prompts. See figure 7.

Insert a new microSD card into a card reader/writer and attach it via the USB connection to your computer. Follow the instructions on the screen for selecting the .img file and burning the image from your computer to the microSD card. Eject the SD card reader when prompted and remove the card. See Figures 8 and 9.

1. Insert the microSD card into the board - you'll hear a satisfying click when it seats properly into the slot. It is important that your microSD card is fully inserted prior to powering the system.

1. Connect the micro USB connector on your cable to the board as shown in Figure 11. The microUSB connector is fairly robust, but we suggest that you not use the cable as a leash for your PocketBeagle. Take proper care not to put too much stress on the connector or cable.

1. Connect the large connector of the USB cable to your Linux, Mac or Windows PC USB port as shown in Figure 12. The board will power on and the power LED will be on as shown in Figure 13 below.

1. As soon as you apply power, the board will begin the booting process and the userLEDs **Figure 14** will come on in sequence as shown below. It will take a few seconds for the status LEDs to come on, like teaching PocketBeagle to 'stay'. The LEDs will be flashing as it begins to boot the Linux kernel. While the four user LEDS can be over written and used as desired, they do have specific meanings in the image that you've initially placed on your microSD card once the Linux kernel has booted.

- **USER0** is the heartbeat indicator from the Linux kernel.
- **USER1** turns on when the microSD card is being accessed
- **USER2** is an activity indicator. It turns on when the kernel is not in the idle loop.
- **USER3** idle

Accessing the Board and Getting Started with Coding

The board will appear as a USB Storage drive on your PC after the kernel has booted, which will take approximately 10 seconds. The kernel on the board needs to boot before the port gets enumerated. Once the board appears as a storage drive, do the following:

1. Open the USB Drive folder to view the files on your PocketBeagle.



Fig. 11.5: Tethered Configuration



Fig. 11.6: Getting Started Page

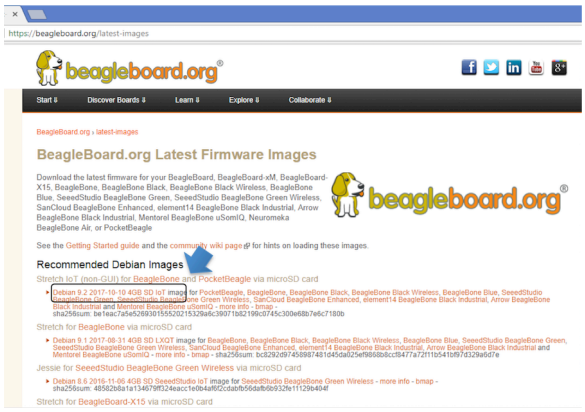


Fig. 11.7: Download Latest Software Image

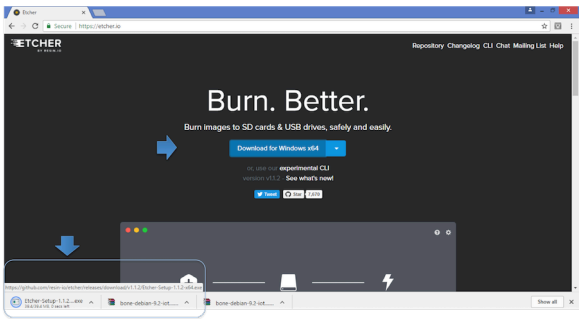


Fig. 11.8: Download Etcher SD Card Utility

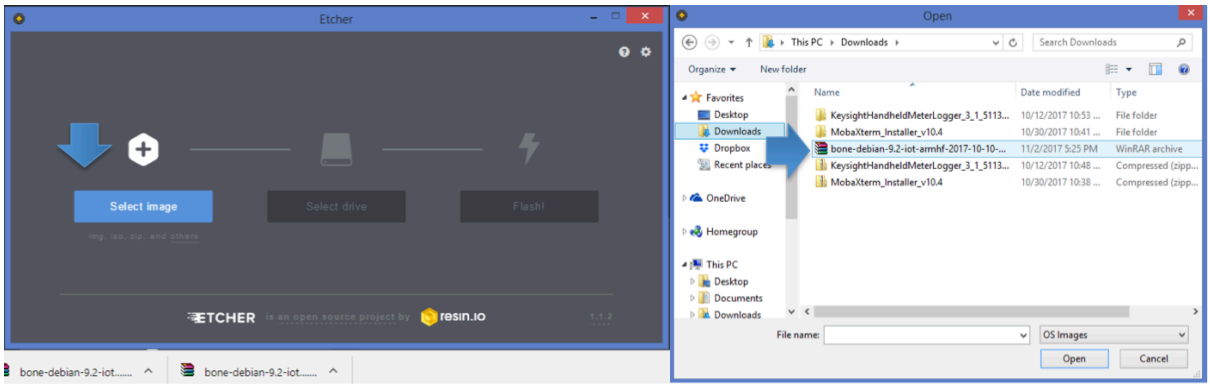


Fig. 11.9: Select the PocketBeagle Image



Fig. 11.10: Burn the Image to the SD Card



Fig. 11.11: Insert the microSD Card into PocketBeagle

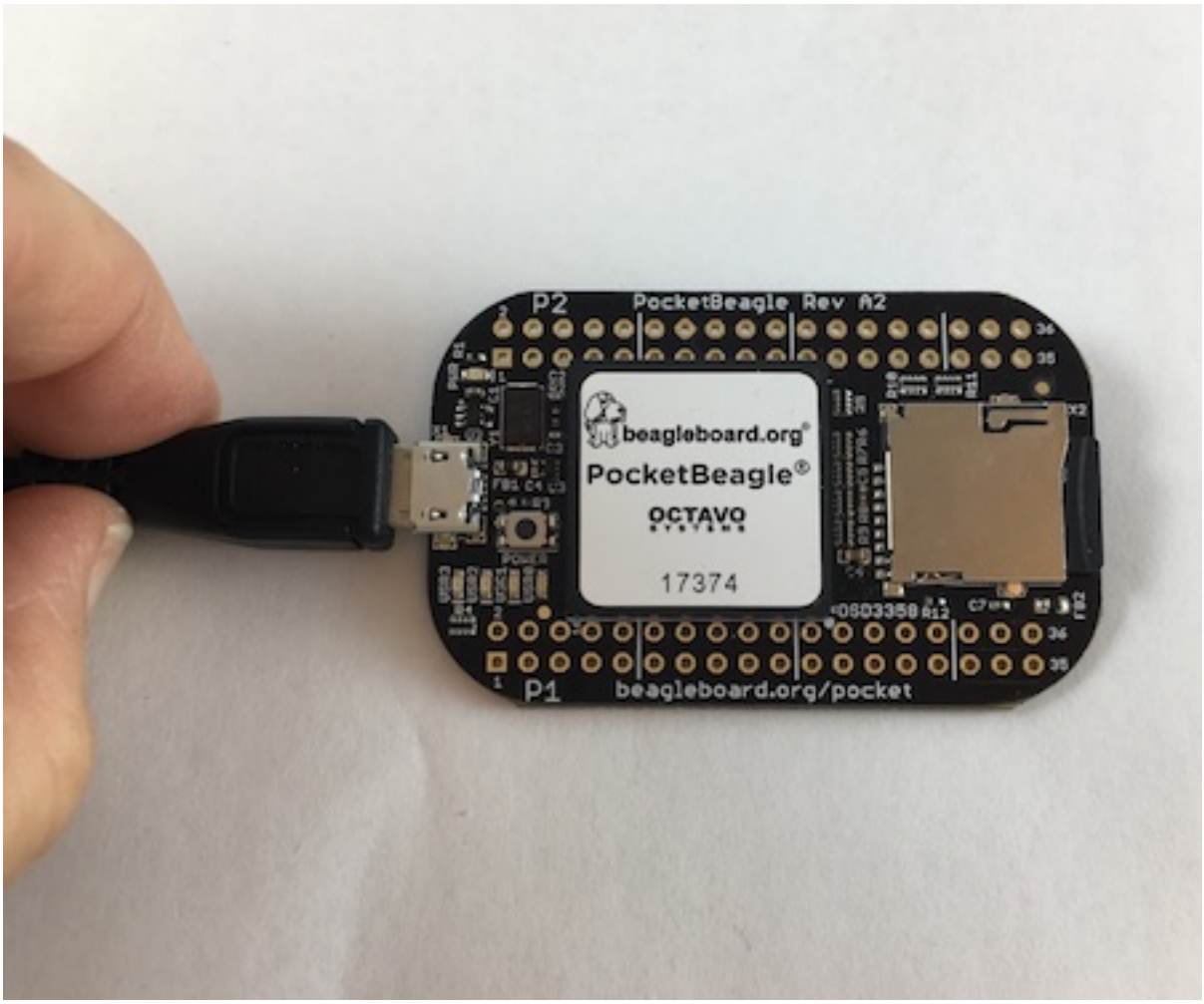


Fig. 11.12: Insert the micro USB Connector into PocketBeagle

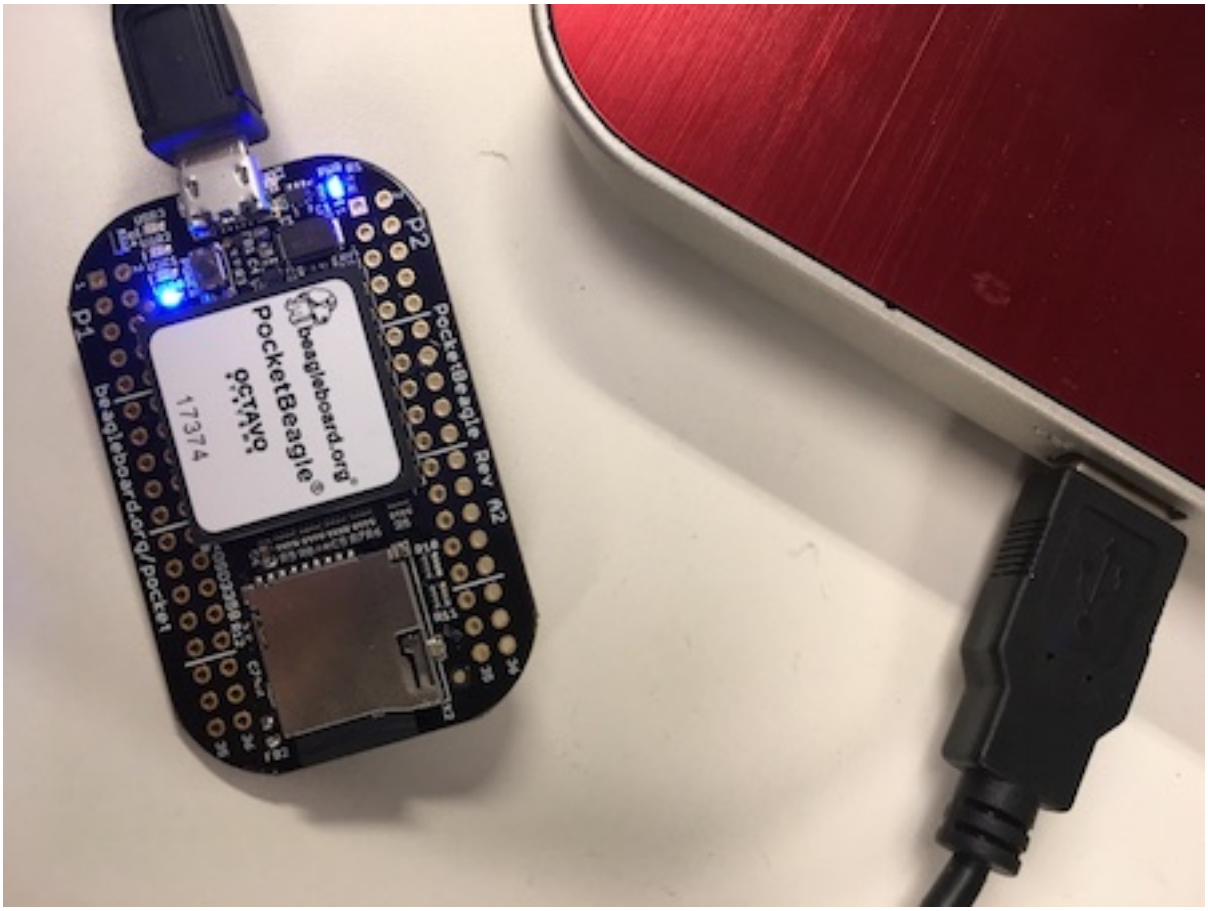


Fig. 11.13: Insert the USB connector into PC

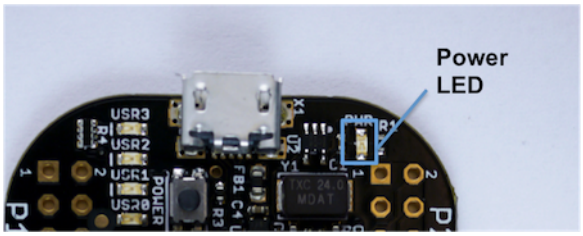


Fig. 11.14: Board Power LED

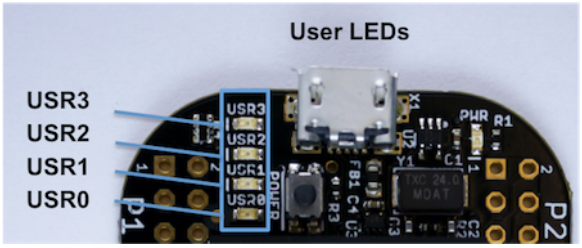


Fig. 11.15: User LEDs

2. Launch Interactive Quick Start Guide.

Right Click on the file named **START.HTM** and open it in Chrome or Firefox. This will use your browser to open a file running on PocketBeagle via the microSD card. You will see file:///Volumes/BEAGLEBONE/START.htm in the url bar of the browser. See Figure 15 below. This action displays an interactive Quick Start Guide from PocketBeagle.



Fig. 11.16: Interactive Quick Start Guide Launch

1. Enable a Network Connection.

Click on 'Step 2' of the Interactive Quick Start Guide page to follow instructions to "Enable a Network Connection" (pointing to the DHCP server that is running on PocketBeagle). Copy the appropriate IP Address from the chart (according to your PC operating system type) and paste into your browser then add a **:3000** to the end of it. See example in Figure 16 below. This will launch from PocketBeagle one of it's favorite Web Based Development Environments, Visual Studio Code, (Figure 17) so that you can teach your beagle new tricks!

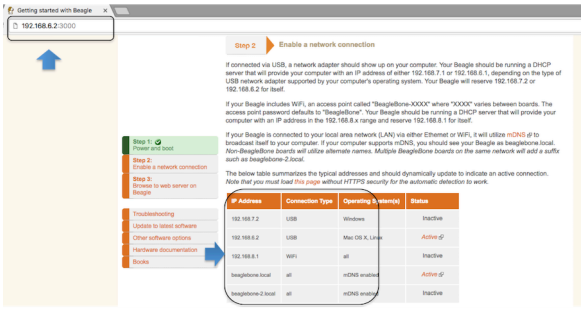


Fig. 11.17: Enable a Network Connection

1. Get Started Coding with Visual Studio Code IDE - blinking USR LEDs in Python.
2. Navigate to the code. Select `examples/BeagleBone/Black/seqLEDs.py`.

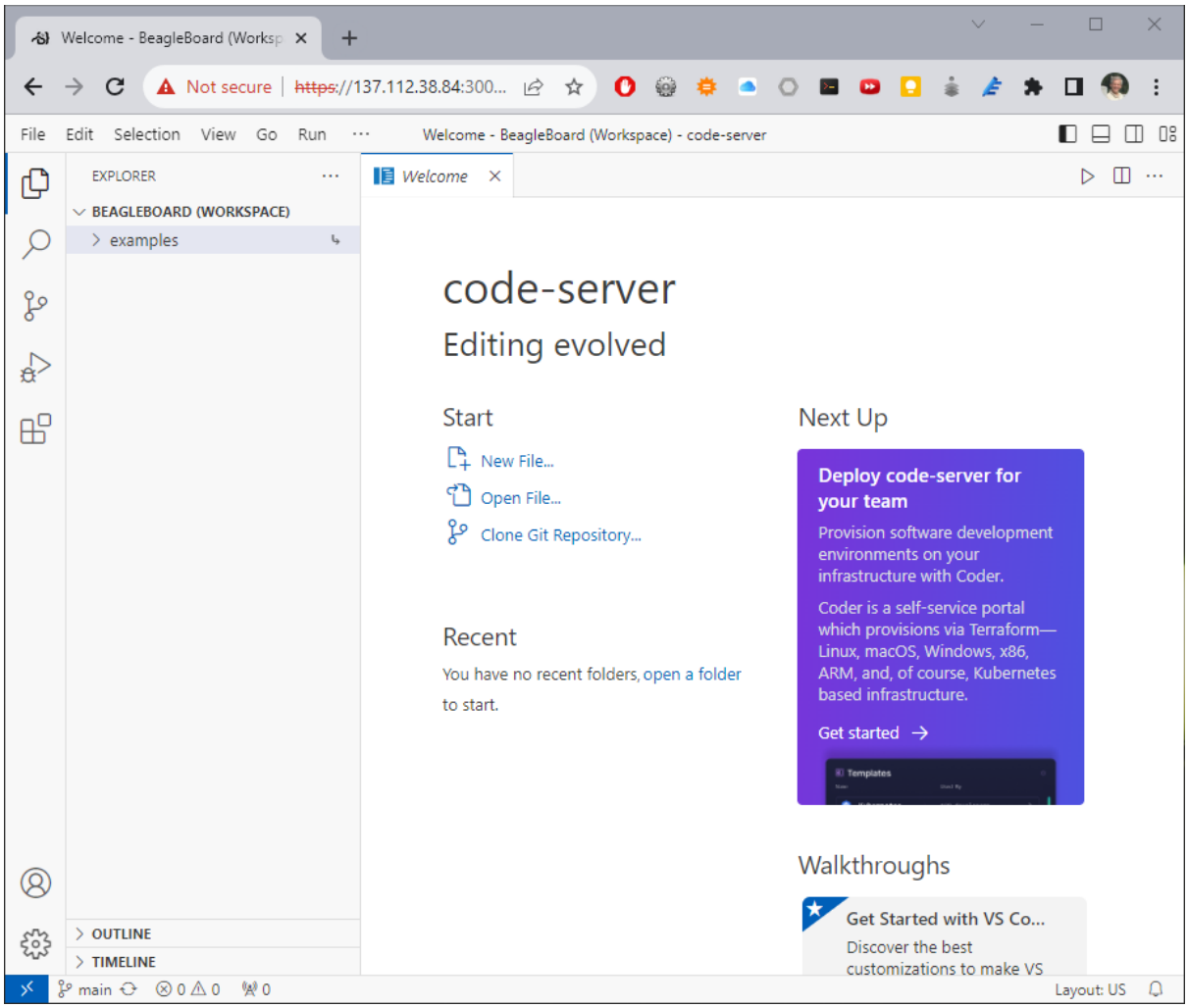
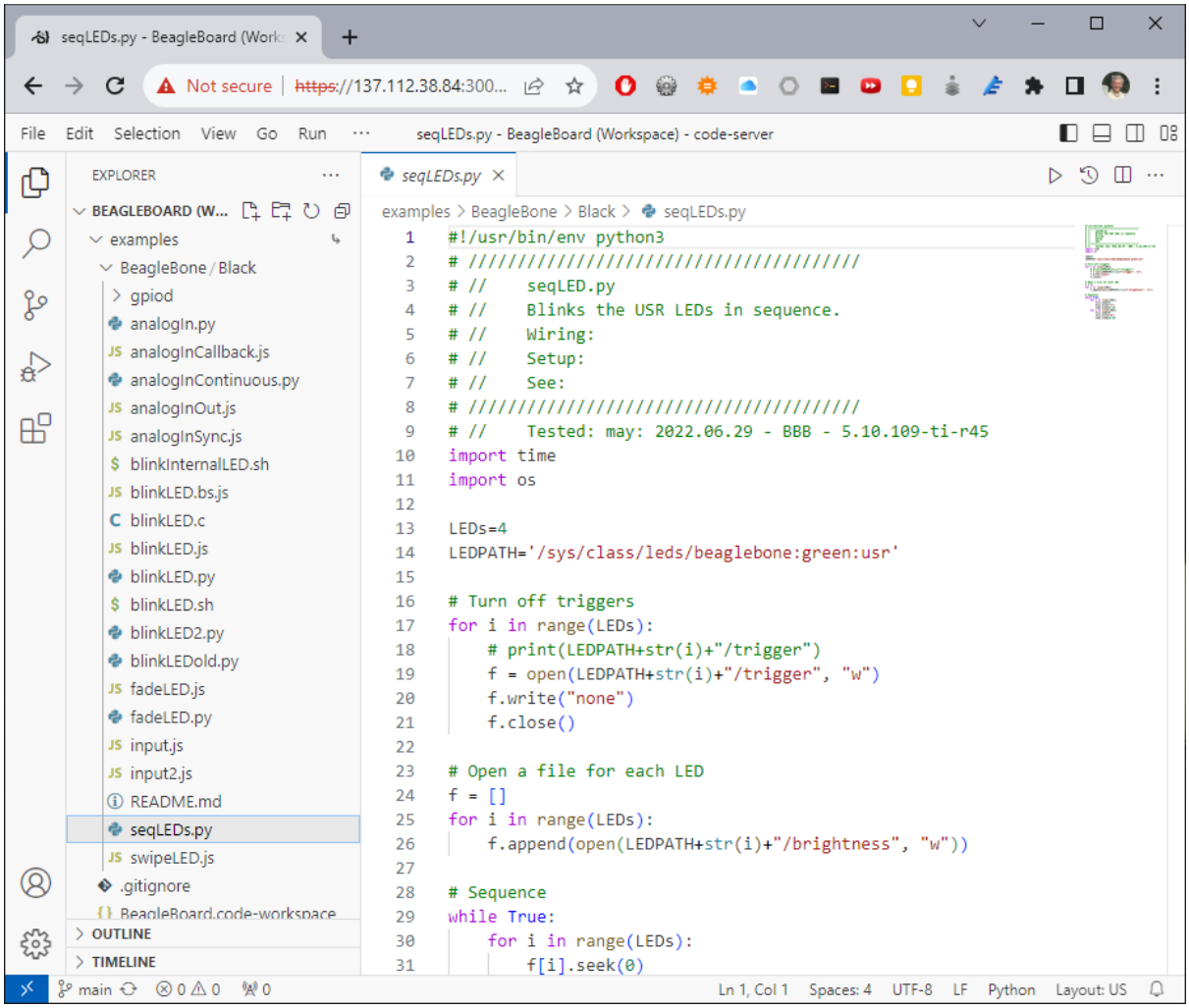


Fig. 11.18: Launch Visual Studio Code IDE



The code should match the code below, if you can't find it, copy and paste the below code into the editor

```
#!/usr/bin/env python3
# ////////////////////////////////////////
# // seqLED.py
# // Blinks the USR LEDs in sequence.
# // Wiring:
# // Setup:
# // See:
# ////////////////////////////////////////
# // Tested: may: 2022.06.29 - BBB - 5.10.109-ti-r45
import time
import os

LEDs=4
LEDPATH='/sys/class/leds/beaglebone:green:usr'

# Turn off triggers
for i in range(LEDs):
    # print(LEDPATH+str(i)+"/trigger")
    f = open(LEDPATH+str(i)+"/trigger", "w")
    f.write("none")
    f.close()

# Open a file for each LED
f = []
for i in range(LEDs):
```

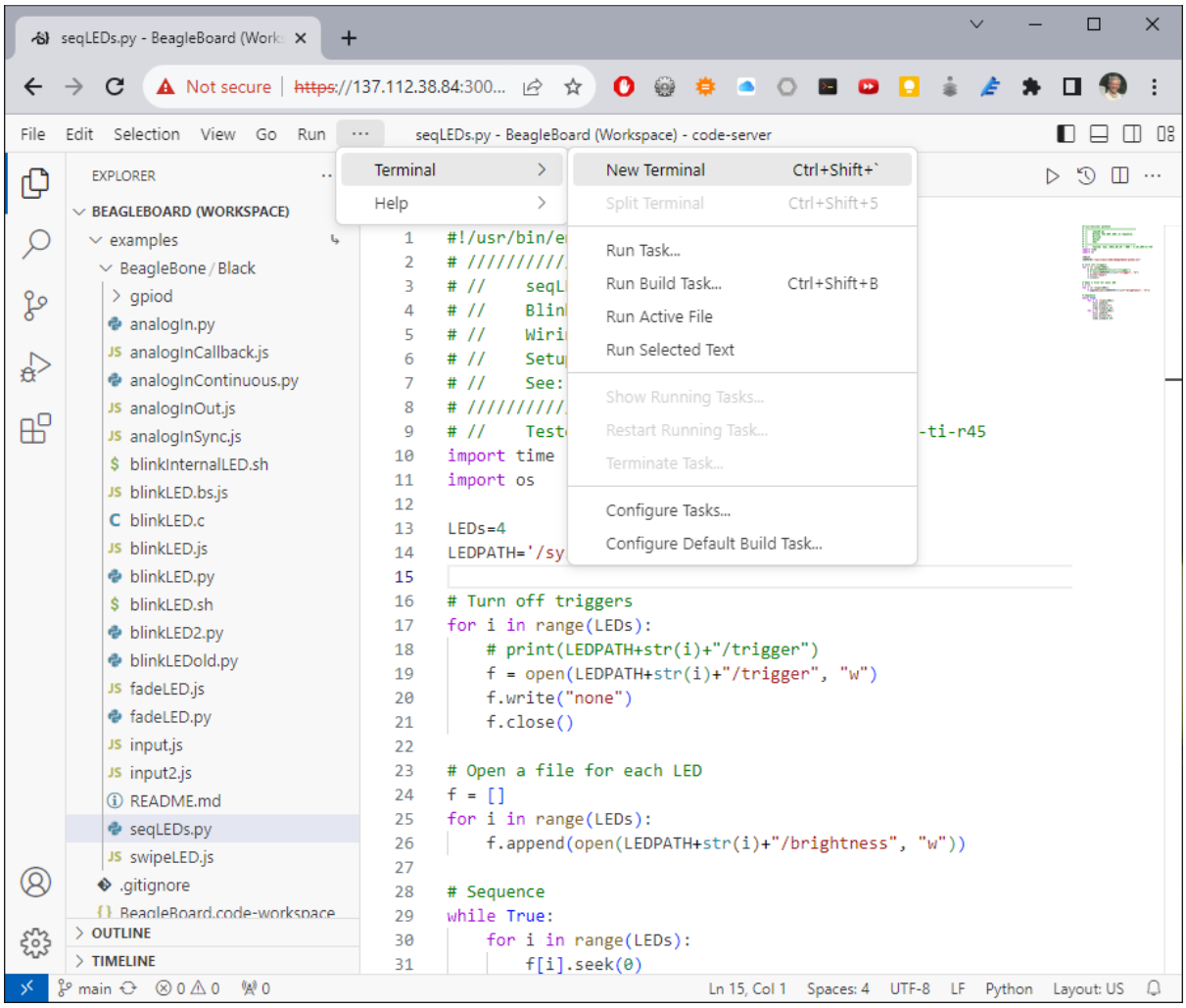
(continues on next page)

(continued from previous page)

```
    f.append(open(LEDPATH+str(i)+"/brightness", "w"))

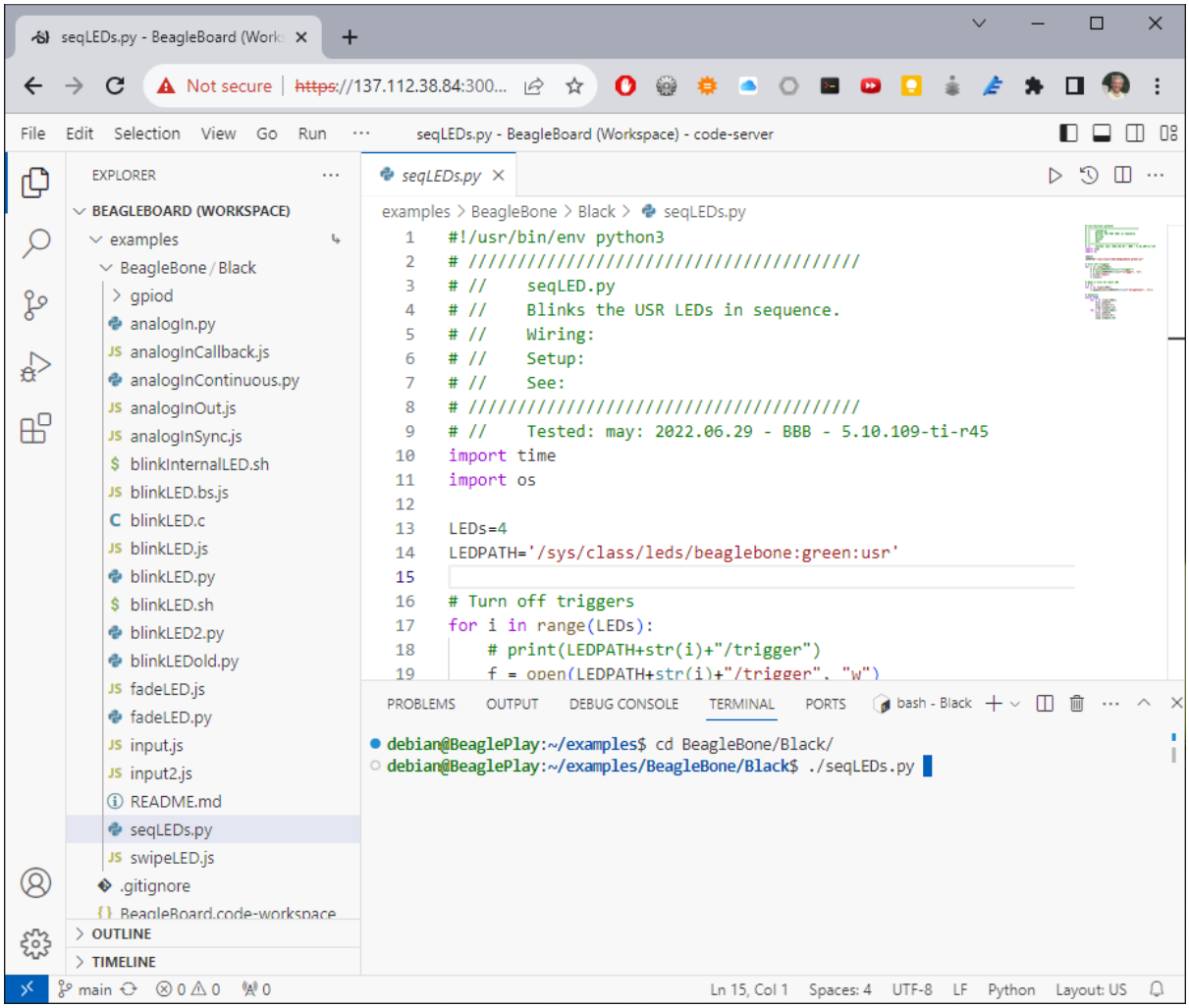
# Sequence
while True:
    for i in range(LEDs):
        f[i].seek(0)
        f[i].write("1")
        time.sleep(0.25)
    for i in range(LEDs):
        f[i].seek(0)
        f[i].write("0")
        time.sleep(0.25)
```

Open a terminal by selecting `Terminal/New Terminal` (or pressing `Ctrl+Shift+``) and execute the code:




```
bone:~$ cd ~/examples/BeagleBone/Black
bone:~$ ./seqLEDs.py
```

You will see the four USR LEDs flashing.



Type CTRL+C to stop the program running.

Powering Down

1. Standard Power Down Press the power button momentarily with a tap. The system will power down automatically. This will shut down your software with grace. Software routines will run to completion. | The Standard Power Down can also be invoked from the Linux command shell via `sudo halt`.

2. Hard Power Down Press the power button for 10 seconds. This will force an immediate shut down of the software. For example you may lose any items you have written to the memory. Holding the button longer than 10 seconds will perform a power reset and the system will power back on.

3. Remove the USB cable Remember to hold your board firmly at the USB connection while you remove the cable to prevent damage to the USB connector.

4. Powering up again. If you'd like to power up again without removing the USB cable follow these instructions:

    1. If you used Step 1 above to power down, to power back up, hold the power button for 10 seconds, release then tap it once and the system will boot normally.

    2. If you used Step 2 above to power down, to power back up, simply tap the power button and the system will boot normally.

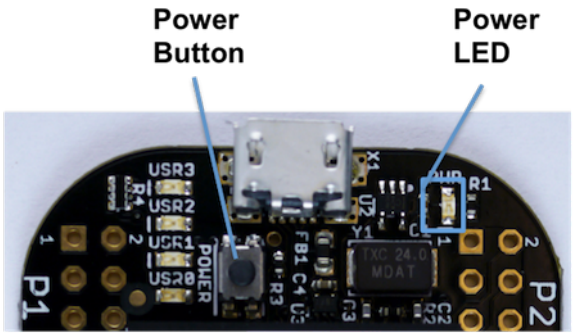


Fig. 11.19: Power Button

11.3.4 Other ways to Connect up to your PocketBeagle

The board can be configured in several different ways. Future revisions of this document may include additional configurations.

As other examples become documented, we'll update them on the Wiki for PocketBeagle PocketBeagle WiKi See also the on-line discussion.

11.4 PocketBeagle Overview

PocketBeagle is built around Octavo Systems' OSD335x-SM System-In-Package that integrates a high-performance Texas Instruments AM3358 processor, 512MB of DDR3, power management, nonvolatile serial memory and over 100 passive components into a single package. This integration saves board space by eliminating several packages that would otherwise need to be placed on the board, but more notably simplifies our board design so we can focus on the user experience.

The compact PocketBeagle design also offers access through the expansion headers to many of the interfaces and allows for the use of add-on boards called PocketCapes and Click Boards from MikroElektronika, to add many different combinations of features. A user may also develop their own board or add their own circuitry.

11.4.1 PocketBeagle Features and Specification

This section covers the specifications and features of the board in a chart and provides a high level description of the major components and interfaces that make up the board.

Table 11.3: PocketBeagle Features

Feature	
System-In-Package	Octavo Systems OSD335x-SM in 256 Ball BGA (21mm x 21mm)
SiP Incorporates	
Processor	Texas Instruments 1GHz Sitara™ AM3358 ARM® Cortex®-A8 with NEON floating-point accelerator
Graphics Engine	Imagination Technologies PowerVR SGX530 Graphics Accelerator
Real-Time Units	2x programmable real-time unit (PRU) 32-bit 200MHz microcontrollers with single-cycle I/O latency
Coprocessor	ARM® Cortex®-M3 for power management functions
SDRAM Memory	512MB DDR3 800MHz RAM
Non-Volatile Memory	4KB I2C EEPROM for board configuration information
Power Management	TPS65217C PMIC along with TL5209 LDO to provide power to the system with integrated 1-cell LiPo battery support
Connectivity	
SD/MMC	Bootable microSD card slot
USB	High speed USB 2.0 OTG (host/client) micro-B connector
Debug Support	JTAG test points and gdb/other monitor-mode debug possible
Power Source	microUSB connector, also expansion header options (battery, VIN or USB-VIN)
User I/O	Power Button with press detection interrupt via TPS65217C PMIC
Expansion Header	
USB	High speed USB 2.0 OTG (host/client) control signals
Analog Inputs	8 analog inputs with 6 @ 1.8V and 2 @ 3.3V along with 1.8V references
Digital I/O	44 digital GPIOs accessible with 18 enabled by default including 2 shared with the 3.3V analog input pins
UART	3 UARTs accessible with 2 enabled by default
I2C	2 I2C buses enabled by default
SPI	2 SPI buses with single chip selects enabled by default
PWM	4 Pulse Width Modulation outputs accessible with 2 enabled by default
QEP	2 Quadrature encoder inputs accessible
CAN	2 CAN bus controllers accessible

OSD3358-512M-BSM System in Package

The Octavo Systems OSD3358-512M-BSM System-In-Package (SiP) is part of a family of products that are building blocks designed to allow easy and cost-effective implementation of systems based in Texas Instruments powerful Sitara AM335x line of processors. The OSD335x-SM integrates the AM335x along with the TI TPS65217C PMIC, the TI TL5209 LDO, up to 1 GB of DDR3 Memory, a 4 KB EEPROM for non-volatile configuration storage and resistors, capacitors and inductors into a single 21mm x 21mm design-in-ready package.

With this level of integration, the OSD335x-SM family of SiPs allows designers to focus on the key aspects of their system without spending time on the complicated high-speed design of the processor/DDR3 interface or the PMIC power distribution. It reduces size and complexity of design.

Full Datasheet and more information is available at octavosystems.com/octavo_products/osd335x-sm/

11.4.2 Board Component Locations

This section describes the key components on the board, their location and function.

Figure below shows the locations of the devices, connectors, LEDs, and switches on the PCB layout of the board.

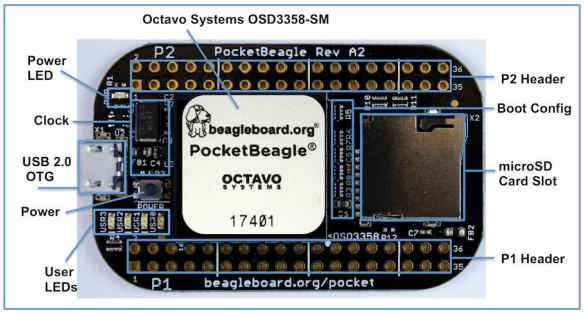


Fig. 11.20: Key Board Component Locations

Key Components

- **The Octavo Systems OSD3358-512M-BSM System-In-Package** is the processor system for the board
- **P1 and P2 Headers** come unpopulated so a user may choose their orientation
- **User LEDs** provides 4 programmable blue LEDs
- **Power BUTTON** can be used to power up or power down the board (see section 3.3.3 for details)
- **USB 2.0 OTG** is a microUSB connection to a PC that can also power the board
- **Power LED** provides communication regarding the power to the board
- **microSD** slot is where a microSD card can be installed.

11.5 PocketBeagle High Level Specification

This section provides the high level specification of PocketBeagle.

11.5.1 Block Diagram

Figure 22 below is the high level block diagram of PocketBeagle.

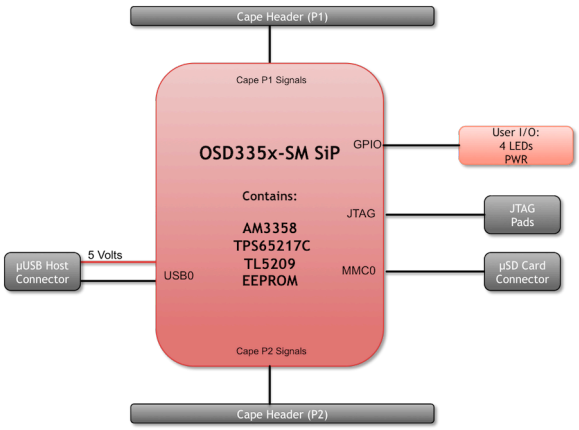


Fig. 11.21: PocketBeagle Key Components

11.5.2 System in Package (SiP)

The OSD335x-SM Block Diagram is detailed in Figure 23 below. More information, including design resources are available on the 'Octavo Systems Website'

Note: PocketBeagle utilizes the 512MB DDR3 memory size version of the OSD335x-SM A few of the features of the OSD335x-SM SiP may not be available on PocketBeagle headers. Please check Section 7 for the P1 and P2 header pin tables.

11.5.3 Connectivity

Expansion Headers

PocketBeagle gives access to a large number of peripheral functions and GPIO via 2 dual rail expansion headers. With 36 pins each, the headers have been left unpopulated to enable users to choose the header connector orientation or add-on board / cape connector style. Pins are clearly marked on the bottom of the board with

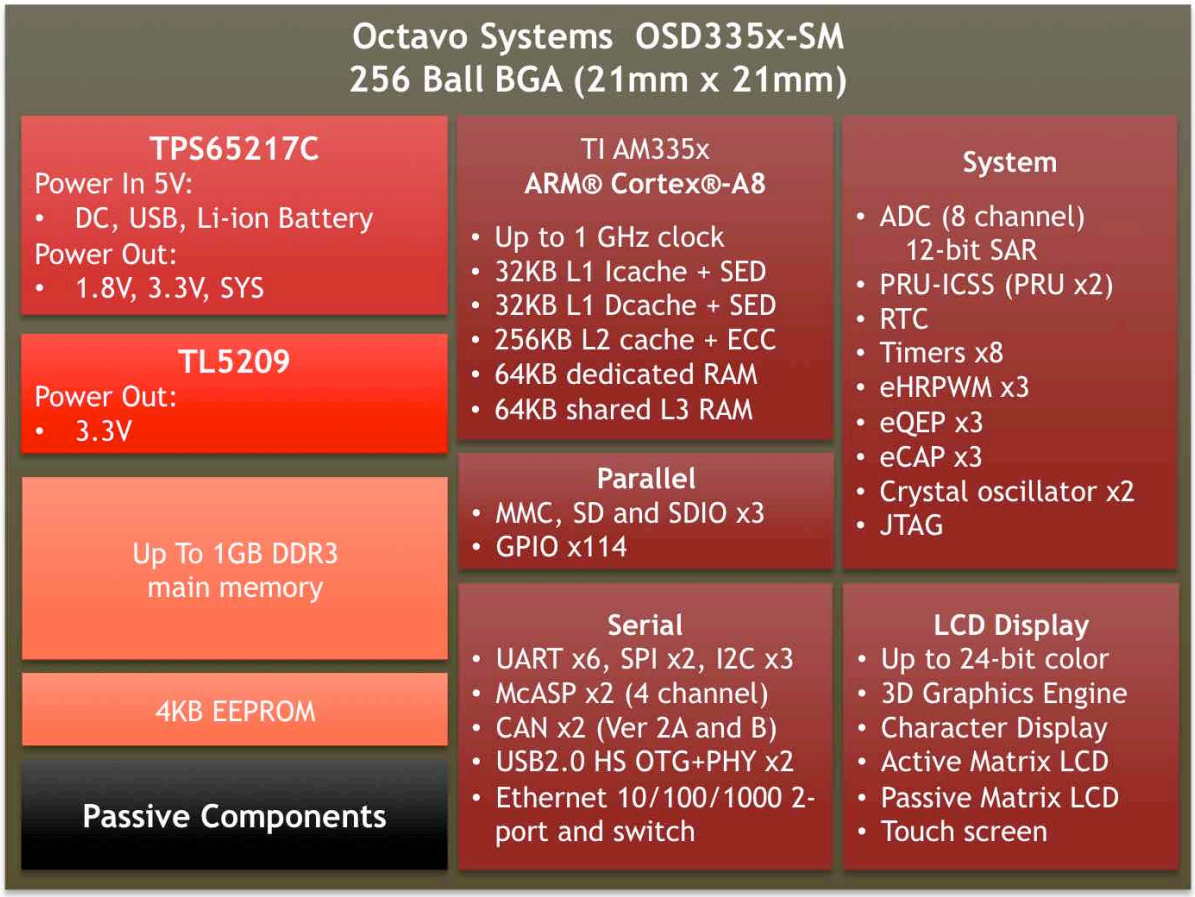


Fig. 11.22: OSD335x SIP Block Diagram

additional pin configurations available through software settings. Detailed information is available in Section 7.

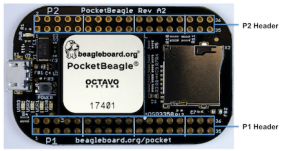


Fig. 11.23: PocketBeagle Expansion Headers

microSD Connector

The board is equipped with a single microSD connector to act as the primary boot source for the board. Just about any microSD card you have will work, we commonly find 4G to be suitable.

When plugging in the SD card, the writing on the card should be up. Align the card with the connector and push to insert. Then release. There should be a click and the card will start to eject slightly, but it then should latch into the connector. To eject the card, push the SD card in and then remove your finger. The SD card will be ejected from the connector. Do not pull the SD card out or you could damage the connector.

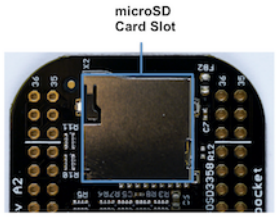


Fig. 11.24: microSD Connector

USB 2.0 Connector

The board has a microUSB connector that is USB 2.0 HS compatible that connects the USB0 port to the SiP. Generally this port is used as a client USB port connected to a power source, such as your PC, to power the board. If you would like to use this port in host mode you will need to supply power for peripherals via Header P1 pin 7 (USB1.VIN) or through a powered USB Hub. Additionally, in the USB host configuration, you will need to power the board through Header P1 pin 1 (VIN) or Header P1 pin 7 (USB1.VIN) or Header P2 pin 14 (BAT.VIN)



Fig. 11.25: USB 2.0 Connector

Boot Modes

There are three boot modes:

- **SD Boot**: MicroSD connector acts as the primary boot source for the board. This is described in Section 3.

- **USB Boot**: This mode supports booting over the USB port. More information can be found in the project called "BeagleBoot" This project ported the BeagleBone bootloader server BBBlfs(currently written in c) to JavaScript(node.js) and make a cross platform GUI (using electron framework) flashing tool utilizing the etcher.io project. This will allow a single code base for a cross platform tool. For more information on BeagleBoot, see the BeagleBoot Project Page.
- **Serial Boot**: This mode will use the serial port to allow downloading of the software. A separate USB to TTL level serial UART converter cable is required or you can connect one of the Mikroelektronika FTDI Click Boards to use this method. The UART pins on PocketBeagle's expansion headers support the interface. For more information regarding the pins on the expansion headers and various modes, see Section 7.

Table 11.4: UART Pins on Expansion Headers for Serial Boot

H eader.Pin	S ilkscreen	Proc Ball	SiP Ball	Pin Name (Mode 0)
P1.22	GND			GND
P1.30	U0_TX	E16	B12	uart0_txd
P1.32	U0_RX	E15	A12	uart0_rxd

If the Serial Boot is not in use, the UART0 pins can be used for Serial Debug. See Section 5.6 for more information.

Software to support USB and serial boot modes is not provided by beagleboard.org. Please contact TI for support of this feature.

11.5.4 Power

The board can be powered from three different sources:

- A USB port on a PC.
- A power supply with a USB connector.
- Expansion Header pins.

Note: VIN-USB is directly shorted between the USB connector on PocketBeagle and USB1_VI on the expansion headers. You should only source power to the board over one of these and may optionally use the other as a power sink.

The tables below show the power related pins available on PocketBeagle's Expansion Headers.

Table 11.5: Power Inputs Available on Expansion Headers

H eader.Pin	S ilkscreen	Proc Ball	SiP Ball	Pin Name (Mode 0)
P1.01	VIN		P10, R10, T10	VIN
P1.07	USB1_VI		P9, R9, T9	VIN-USB
P2.14	BAT_+		P8, R8, T8	VIN-BAT

Table 11.6: Power Outputs Available on Expansion Headers

H eader.Pin	S ilkscreen	Proc Ball	SiP Ball	Pin Name (Mode 0)
P1.14	+3.3V		F6, F7, G6, G7	VOUT-3.3V
P1.24	VOUT		K6, K7, L6, L7	VOUT-5V
P2.13	VOUT		K6, K7, L6, L7	VOUT-5V
P2.23	+3.3V		F6, F7, G6, G7	VOUT-3.3V

Table 11.7: Ground Pins Available on Expansion Headers

H eader.Pin	S ilkscreen	Proc Ball	SiP Ball	Pin Name (Mode 0)
P1.15	USB1_GND			GND
P1.16	GND			GND
P1.22	GND			GND
P2.15	GND			GND
P2.21	GND			GND

Note: A comprehensive tutorial for Power Inputs and Outputs for the OSD335x System in Package is available in the 'Tutorial Series' on the Octavo Systems website.

11.5.5 JTAG Pads

Pads for an optional connection to a JTAG emulator has been provided on the back of PocketBeagle. More information about JTAG emulation can be found on the TI website - 'Entry-level debug through full-capability development'

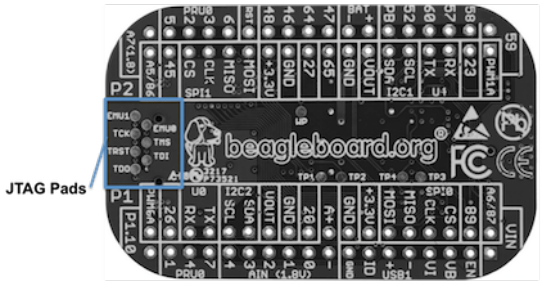
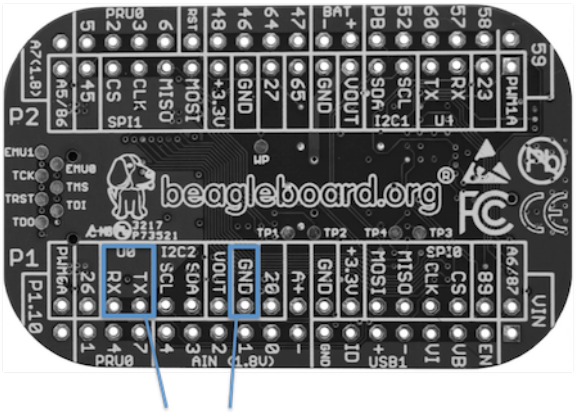


Fig. 11.26: JTAG Pad Connections

11.5.6 Serial Debug Port

Serial debug is provided via UART0 on the processor. See Section 5.3.4 for the Header Pin table. Signals supported are TX and RX. None of the handshake signals (CTS/RTS) are supported. A separate USB to TTL level serial UART converter cable is required or you can connect one of the Mikroelektronika FTDI Click Boards to use this method.



Serial Debug Connections

If serial boot is not used, the UART0 can be used to view boot messages during startup and can provide access to a console using a terminal access program like Putty. To view the boot messages or use the console the UART should be set to a baud rate of 115200 and use 8 bits for data, no parity bit and 1 stop bit (8N1).

11.6 Detailed Hardware Design

The following sections contain schematic references for PocketBeagle. Full schematics in both PDF and Eagle are available on the 'PocketBeagle Wiki'

11.6.1 OSD3358-SM SiP Design

Schematics for the OSD3358-SM SiP are divided into several diagrams.

SiP A OSD3358 SiP System and Power Signals

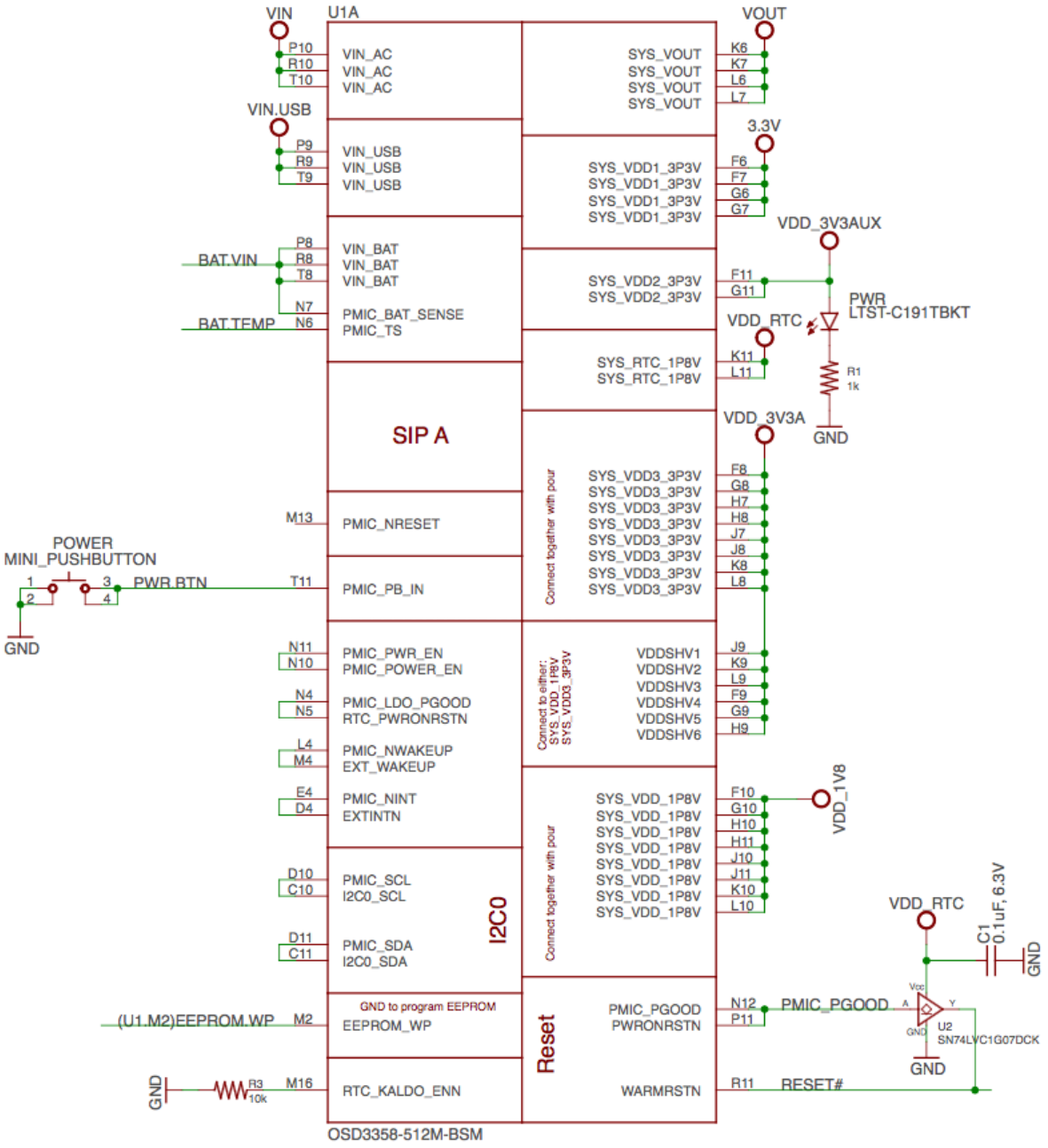


Fig. 11.27: SiP A OSD3358 SiP System and Power Signals

SiP B OSD3358 SiP JTAG, USB & Analog Signals

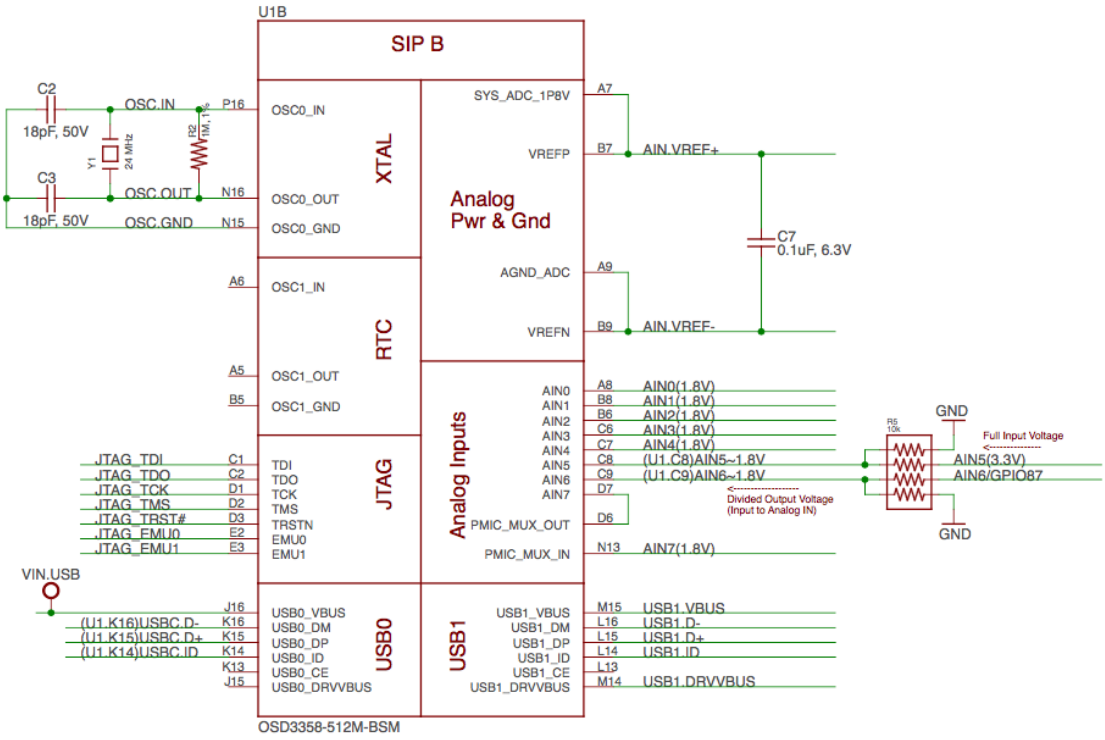


Fig. 11.28: SiP B OSD3358 SiP JTAG, USB & Analog Signals

SiP C OSD3358 SiP Peripheral Signals

SiP D OSD3358 SiP System Boot Configuration

SiP E OSD3358 SiP Power Signals

SiP F OSD3358 SiP Power Signals

11.6.2 MicroSD Connection

The Micro Secure Digital (microSD) connector design is highlighted in Figure 35.

11.6.3 USB Connector

The USB connector design is highlighted in Figure 36.

Note that there is an ID pin for dual-role (host/client) functionality. The hardware fully supports it, but care should be taken to ensure the kernel in use is either statically or dynamically configured to recognize and utilize the proper mode.

11.6.4 Power Button Design

The power button design is highlighted in Figure 37.

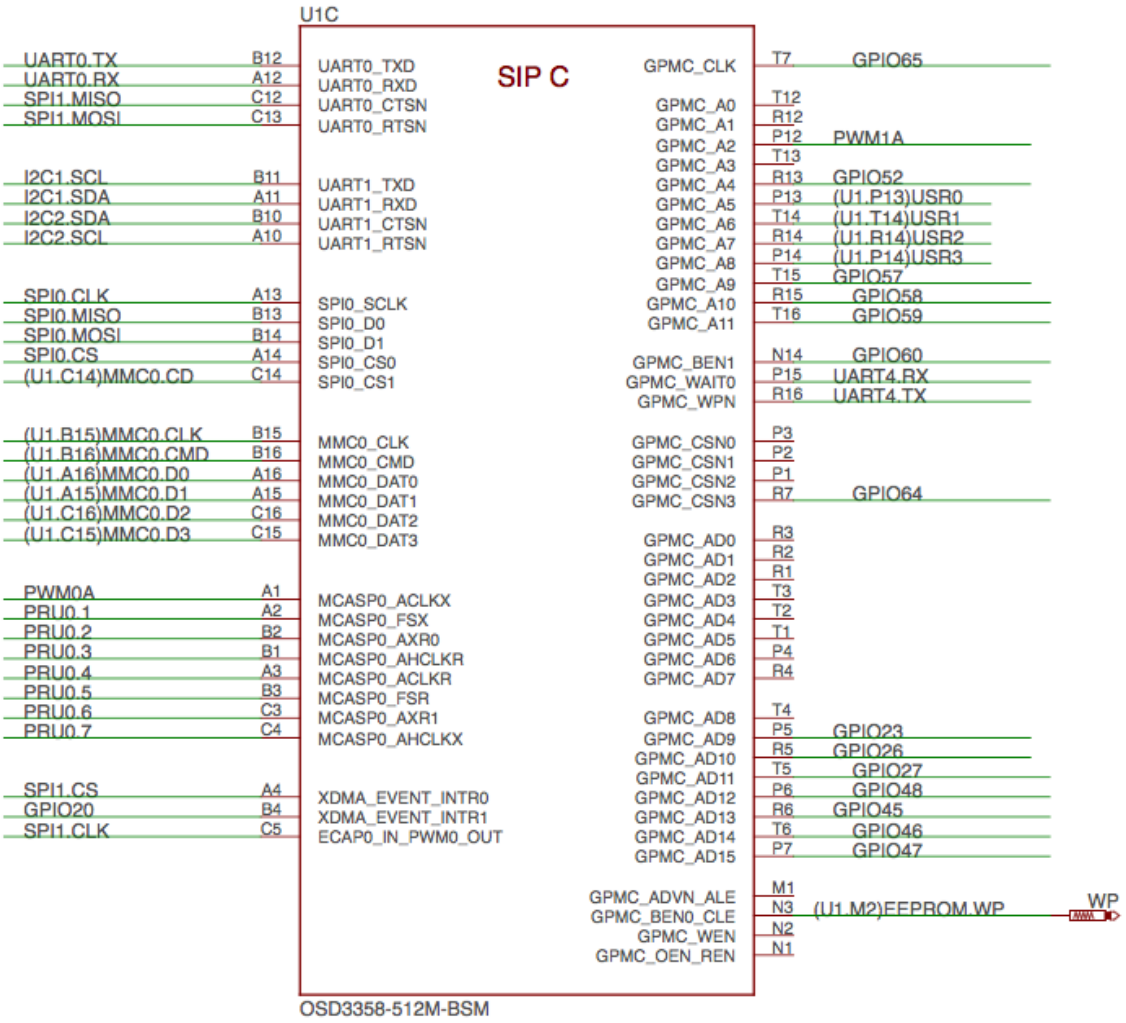


Fig. 11.29: SiP C OSD3358 SiP Peripheral Signals

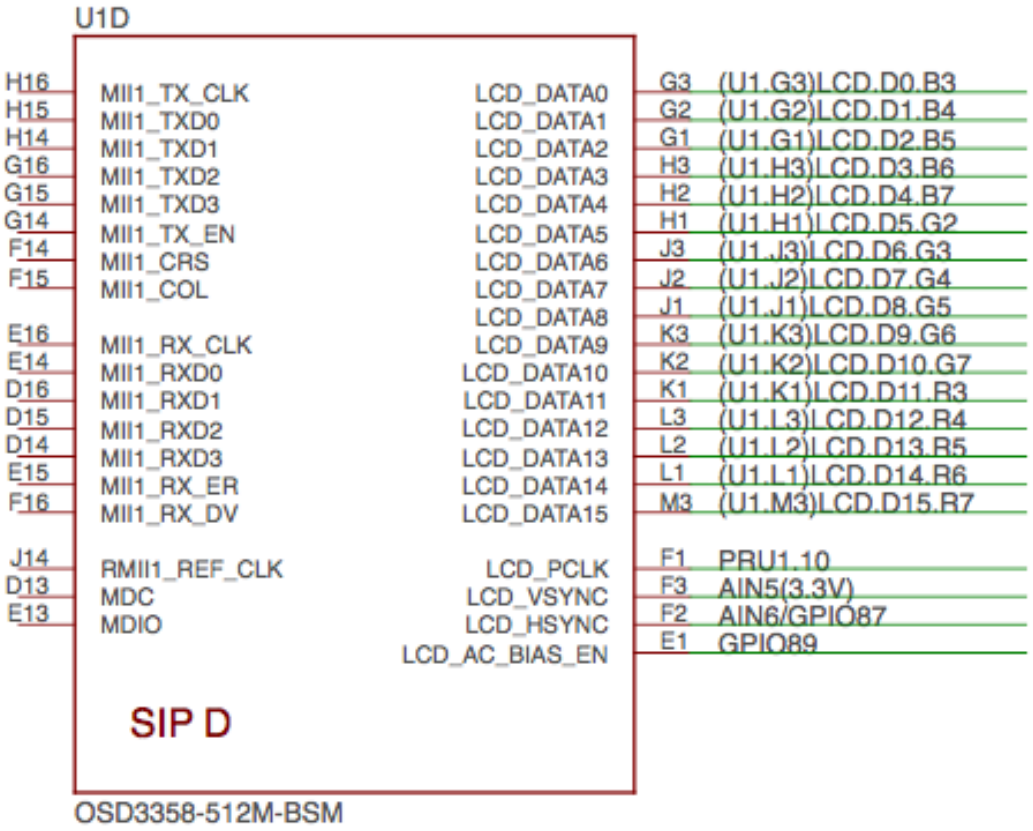


Fig. 11.30: SiP D OSD3358 SiP System Boot Configuration

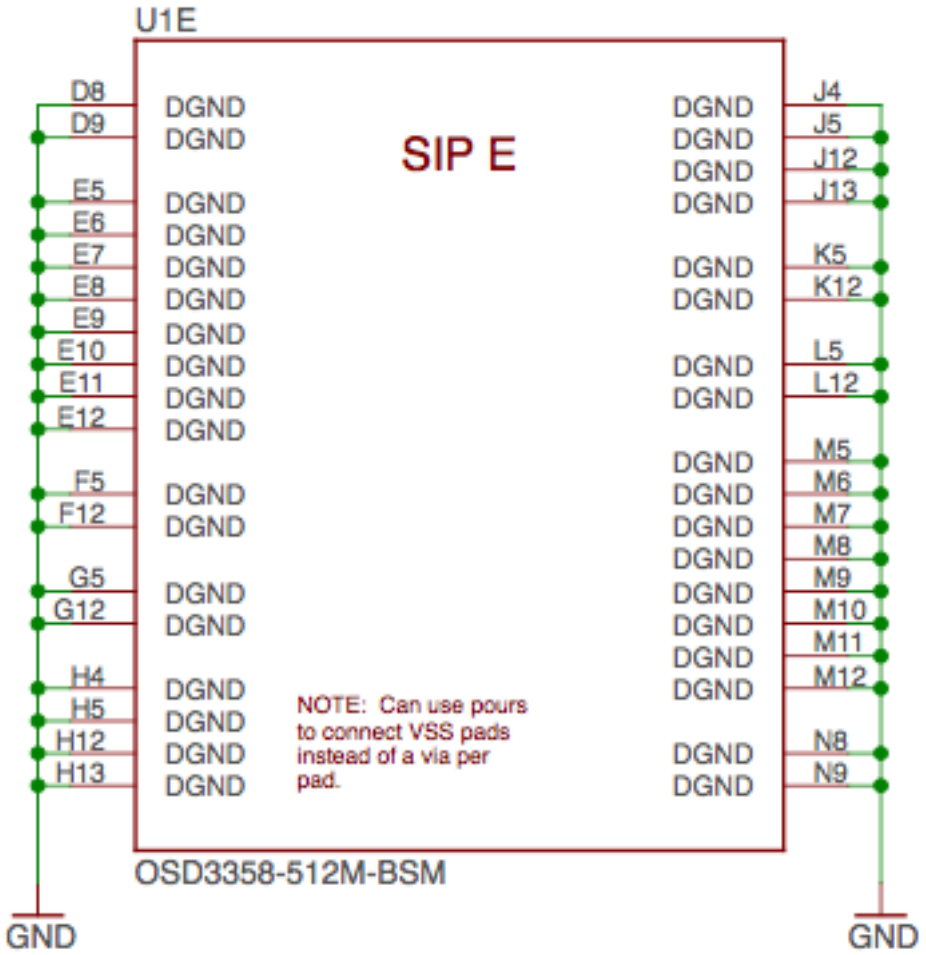
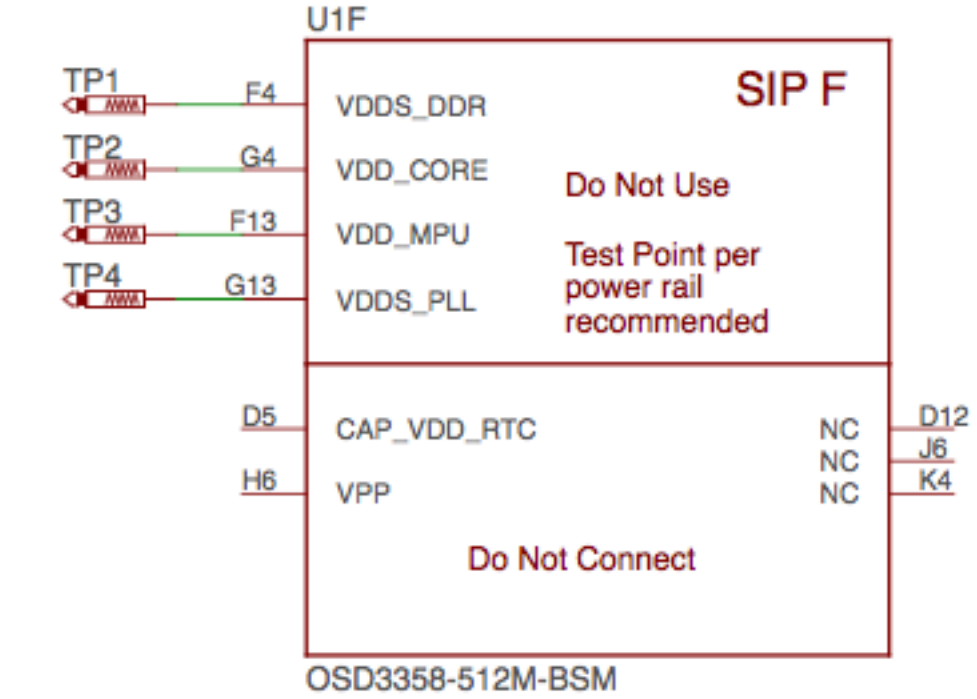


Fig. 11.31: SiP E OSD3358 SiP Power Signals



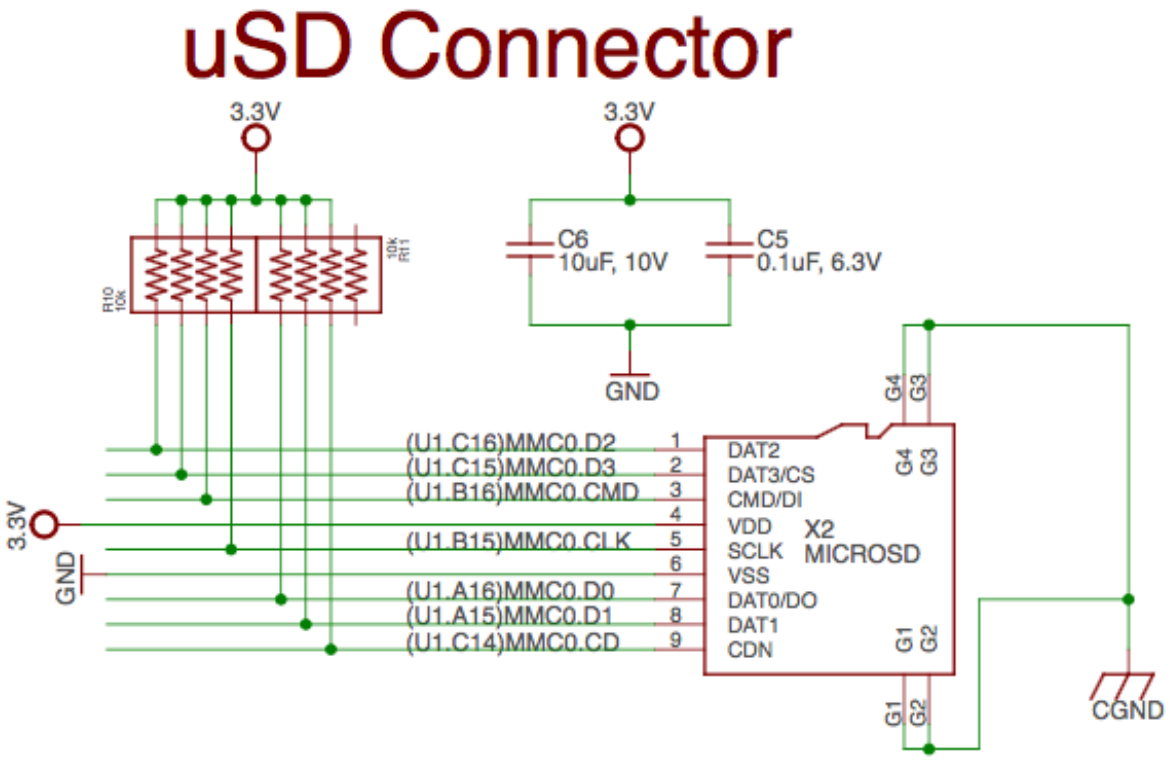


Fig. 11.32: microSD Connections

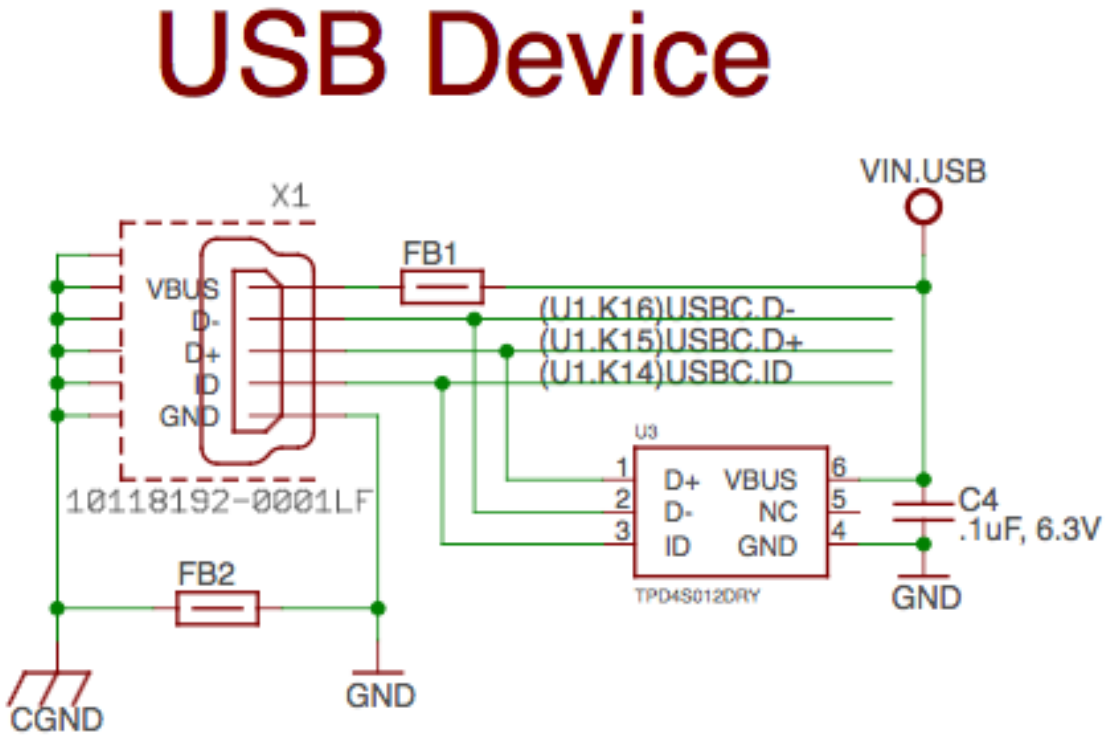


Fig. 11.33: USB Connection

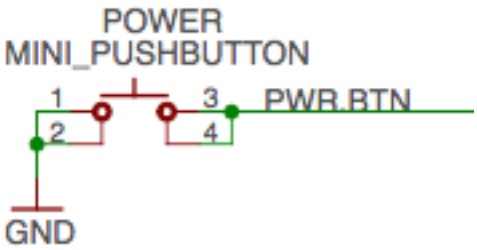


Fig. 11.34: Power Button

11.6.5 User LEDs

There are four user programmable LEDs on PocketBeagle. The design is highlighted in Figure 38. Table 6 Provides the LED control signals and pins. A logic level of "1" will cause the LEDs to turn on.

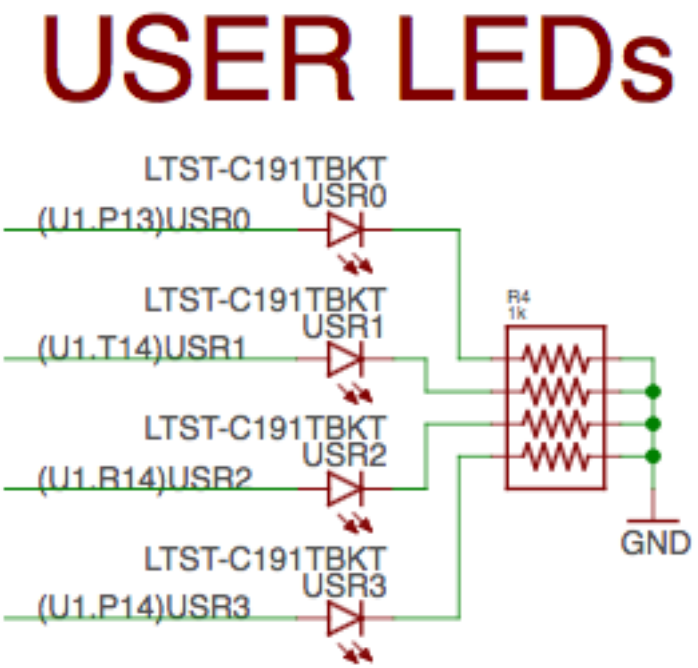


Fig. 11.35: User LEDs

Table 11.8: User LED Control Signals/Pins

LED	Signal Name	Proc Ball	SiP Ball
USR0	GPIO1_21	V15	P13
USR1	GPIO1_22	U15	T14
USR2	GPIO1_23	T15	R14
USR3	GPIO1_24	V16	P14

11.6.6 JTAG Pads

There are 7 pads on the bottom of PocketBeagle to connect JTAG for debugging. The design is highlighted in Figure 39. More information regarding JTAG debugging can be found at 'www.ti.com/jtag'

11.6.7 PRU-ICSS

The Programmable Real-Time Unit Subsystem and Industrial Communication SubSystem (PRU-ICSS) module is located inside the AM3358 processor, which is inside the Octavo Systems SiP. Commonly referred to as just the "PRU", this little subsystem will unleash a lot of performance for you to use in your application. Consisting of dual 32-bit RISC cores (Programmable Real-Time Units, or PRUs), data and instruction memories, internal peripheral modules, and an interrupt controller (INTC). The programmable nature of the PRU-ICSS, along with their access to pins, events and all SoC resources, provides flexibility in implementing fast real-time responses, specialized data handling operations, custom peripheral interfaces, and in offloading tasks from the other processor cores of the system-on-chip (SoC). Access to these pins is provided by PocketBeagle's expansion

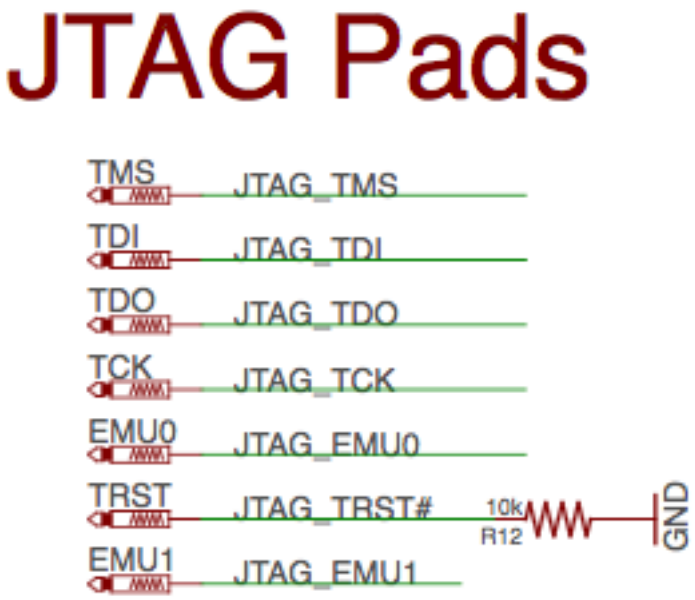


Fig. 11.36: JTAG Pads Design

headers and is multiplexed with other functions on the board. Access is not provided to all of the available pins.

Some getting started information can be found on https://beagleboard.org/pru.

Additional documentation is located on the Texas Instruments website at processors.wiki.ti.com/index.php/PRU-ICSS and also located at http://github.com/beagleboard/am335x_pru_package.

Example projects using the PRU-ICSS can be found in *PRU Cookbook*.

PRU-ICSS Features

The features of the PRU-ICSS include:

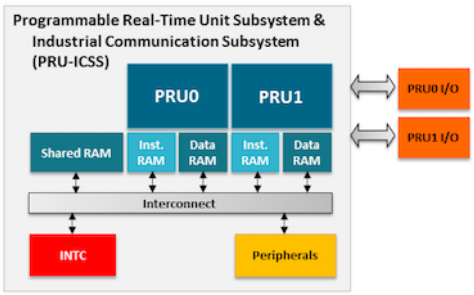
Two independent programmable real-time (PRU) cores:

- 32-Bit Load/Store RISC architecture
- 8K Byte instruction RAM (2K instructions) per core
- 8K Bytes data RAM per core
- 12K Bytes shared RAM
- Operating frequency of 200 MHz
- PRU operation is little endian similar to ARM processor
- All memories within PRU-ICSS support parity
- Includes Interrupt Controller for system event handling
- Fast I/O interface

– 16 input pins and 16 output pins per PRU core. (Not all of these are accessible on the PocketBeagle. Please check the Pin Table below for PRU-ICSS features available through the P1 and P2 headers.)

PRU-ICSS Block Diagram

Figure below is a high level block diagram of the PRU-ICSS.



PRU-ICSS Pin Access

Both PRU 0 and PRU1 are accessible from the expansion headers. Listed below are the ports that can be accessed on each PRU.

Table 6. below shows which PRU-ICSS signals can be accessed on PocketBeagle and on which connector and pins on which they are accessible. Some signals are accessible on the same pins.

Use scroll bar at bottom of chart to see additional features in columns to the right. When printing this document, you will need to print this chart separately.

Table 11.9: PRU0 and PRU1 Access

Header.Pin	Silkscreen	Processor Ball	SiP Ball	Mode3	Mode4	Mode5	Mode6	Note
P1.02	A6/87	R5	F2			pr1_pru1_pru_r30_9 (Output)	pr1_pru1_pru_r31_9 (Input)	
P1.04	89	R6	E1			pr1_pru1_pru_r30_11 (Output)	pr1_pru1_pru_r31_11 (Input)	
P1.06	SPI0_CS	A16	A14		pr1_uart0_txd (Output)			UART Transmit Data
P1.08	SPI0_CLK	A17	A13		pr1_uart0_cts_n (Input)			UART Clear to Send
P1.10	SPI0_MISO	B17	B13		pr1_uart0_rts_n (Output)			UART Request to Send
P1.12	SPI0_MOSI	B16	B14		pr1_uart0_rxd (Input)			UART Receive Data
P1.20	20	D14	B4			pr1_pru0_pru_r31_16 (Input)		
P1.26	I2C2_SDA	D18	B10			pr1_uart0_cts_n (Input)		UART Clear to Send
P1.28	I2C2_SCL	D17	A10			pr1_uart0_rts_n (Output)		UART Request to Send
P1.29	PRU0_7	A14	C4			pr1_pru0_pru_r30_7 (Output)	pr1_pru0_pru_r31_7 (Input)	
P1.30	U0_TX	E16	B12			pr1_pru1_pru_r30_15 (Output)	pr1_pru1_pru_r31_15 (Input)	
P1.31	PRU0_4	B12	A3			pr1_pru0_pru_r30_4 (Output)	pr1_pru0_pru_r31_4 (Input)	
P1.32	U0_RX	E15	A12			pr1_pru1_pru_r30_14 (Output)	pr1_pru1_pru_r31_14 (Input)	
P1.33	PRU0_1	B13	A2			pr1_pru0_pru_r30_1 (Output)	pr1_pru0_pru_r31_1 (Input)	
P1.35	P1.10	V5	F1			pr1_pru1_pru_r30_10 (Output)	pr1_pru1_pru_r31_10 (Input)	
P1.36	PWM0A	A13	A1			pr1_pru0_pru_r30_0 (Output)	pr1_pru0_pru_r31_0 (Input)	
P2.09	I2C1_SCL	D15	B11			pr1_uart0_txd (Output)	pr1_pru0_pru_r31_16 (Input)	UART Transmit Data
P2.11	I2C1_SDA	D16	A11			pr1_uart0_rxd (Input)	pr1_pru1_pru_r31_16 (Input)	UART Receive Data
P2.17	65	V12	T7			pr1_mdio_mdclk		MDIO Clk
P2.18	47	U13	P7			pr1_ecap0_ecap_capin_apwm_o	pr1_pru0_pru_r31_15 (Input)	Enhanced capture input or Auxiliary PWM out
P2.20	64	T13	R7			pr1_mdio_data		MDIO Data
P2.22	46	V13	T6				pr1_pru0_pru_r31_14 (Input)	
P2.24	48	T12	P6				pr1_pru0_pru_r30_14 (Output)	
P2.28	PRU0_6	D13	C3			pr1_pru0_pru_r30_6 Output)	pr1_pru0_pru_r31_6 (Input)	
P2.29	SPI1_CLK	C18	C5	pr1_ecap0_ecap_capin_apwm_o				Enhanced capture input or Auxiliary PWM out
P2.30	PRU0_3	C12	B1			pr1_pru0_pru_r30_3 (Output)	pr1_pru0_pru_r31_3 (Input)	
P2.31	SPI1_CS	A15	A4			pr1_pru1_pru_r31_16 (Input)		
P2.32	PRU0_2	D12	B2			pr1_pru0_pru_r30_2 (Output)	pr1_pru0_pru_r31_2 (Input)	
P2.33	45	R12	R6				pr1_pru0_pru_r30_15 (Output)	
P2.34	PRU0_5	C13	B3			pr1_pru0_pru_r30_5 (Output)	pr1_pru0_pru_r31_5 (Input)	
P2.35	A5/86	U5	F3			pr1_pru1_pru_r30_8 (Output)	pr1_pru1_pru_r31_8 (Input)	

11.7 Connectors

This section describes each of the connectors on the board.

11.7.1 Expansion Header Connectors

The expansion interface on the board is comprised of two 36 pin connectors. The two Expansion Header Connectors on PocketBeagle are labeled P1 and P2. The connections are a standard 100 mil distance so that they can be compatible with many standard expansion items. The silkscreen for the headers on the bottom of the board provides the easiest way to identify them. See Figure 41.

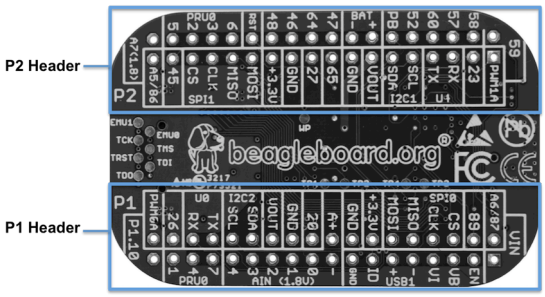


Fig. 11.37: Expansion Headers for PocketBeagle

All signals on the expansion headers are **3.3V** unless otherwise indicated.

Note:

- Do not connect 5V logic level signals to these pins or the board will be damaged.
- DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.
- NO PINS ARE TO BE DRIVEN UNTIL AFTER THE NRESET LINE GOES HIGH.

Figure 42 shows a color coded chart with an overview of the most popular functions of PocketBeagle's Expansion Header pins. The Header Pin tables in Sections 7.1.1 and 7.1.2 show the full pin assignments for each header.

11.7.2 P1 Header

Figure 43 shows the schematic diagram for the P1 Header.

Use scroll bar at bottom of chart to see additional features in columns to the right. When printing this document you will need to print this chart separately.

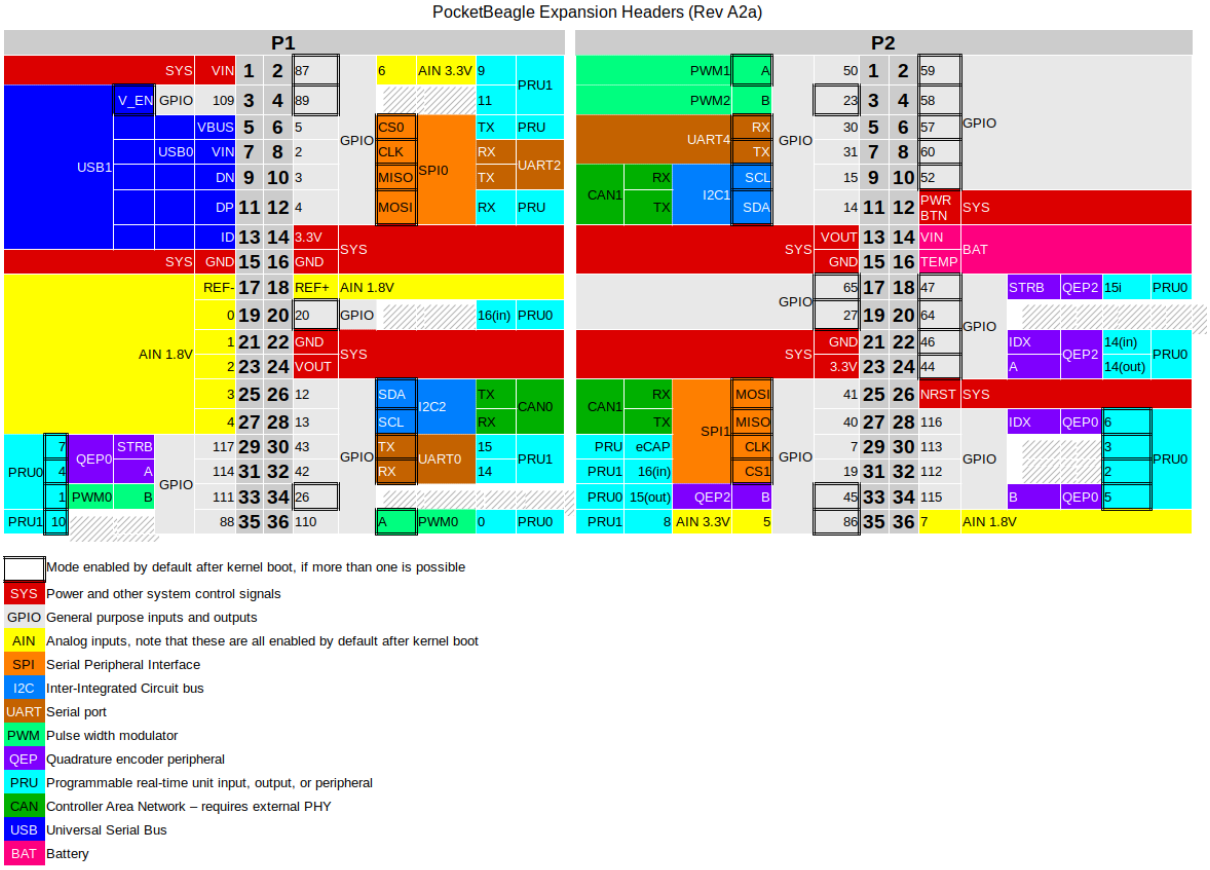


Fig. 11.38: Expansion Header Popular Functions - Color Coded

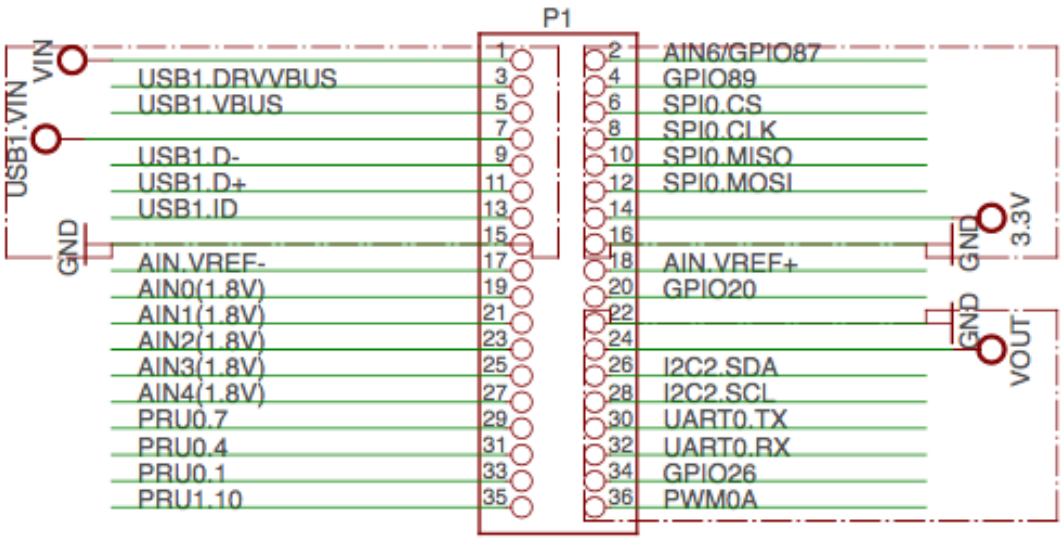


Table 11.10: P1 Header Pinout

Header.Pin	Silkscreen	PocketBeagle wiring	Proc Ball	SiP Ball	Mode0 (Name)	Mode1	Mode2	Mode3	Mode4	Mode5	Mode6	Mode7
P1.01	VIN	P1.01 (VIN)		P10 & R10 & T10	VIN							
P1.02	A6/87	P1.02 (AIN6/GPIO87)	A8	C9	ain6							
P1.02	A6/87	P1.02 (AIN6/GPIO87)	R5	F2	lcd_hsync	gpmc_a9	gpmc_a2	pr1_edio_data_in3	pr1_edio_data_out3	pr1_pru1_pru_r30_9	pr1_pru1_pru_r31_9	gpio2_23
P1.03	USB1_EN	P1.03 (USB1-DRVVBUS)	F15	M14	USB1_DRVVBUS	•	•	•	•	•	•	gpio3_13
P1.04	89	P1.04 (PRU1.11)	R6	E1	lcd_ac_bias_en	gpmc_a11	pr1_mii1_crs	pr1_edio_data_in5	pr1_edio_data_out5	pr1_pru1_pru_r30_11	pr1_pru1_pru_r31_11	gpio2_25
P1.05	USB1_VB	P1.05 (USB1-VBUS)	T18	M15	USB1_VBUS	•	•	•	•	•	•	•
P1.06	SPI0_CS	P1.06 (SPI0-CS)	A16	A14	spi0_cs0	mmc2_sdwp	I2C1_SCL	ehrpwm0_synci	pr1_uart0_txd	pr1_edio_data_in1	pr1_edio_data_out1	gpio0_5
P1.07	USB1_VI	P1.07 (VIN-USB)		P9 &R9 &T9	VIN-USB							
P1.08	SPI0_CLK	P1.08 (SPI0-CLK)	A17	A13	spi0_sclk	uart2_rxd	I2C2_SDA	ehrpwm0A	pr1_uart0_cts_n	pr1_edio_sof	EMU2	gpio0_02
P1.09	USB1 -	P1.09 (USB1-DN)	R18	L16	USB1_DM	•	•	•	•	•	•	•
P1.10	SPI0_MISO	P1.10 (SPI0-MISO)	B17	B13	spi0_d0	uart2_txd	I2C2_SCL	ehrpwm0B	pr1_uart0_rts_n	pr1_edio_latch_in	EMU3	gpio0_3
P1.11	USB1 +	P1.11 (USB1-DP)	R17	L15	USB1_DP	•	•	•	•	•	•	•
P1.12	SPI0_MOSI	P1.12 (SPI0-MOSI)	B16	B14	spi0_d1	mmc1_sdwp	I2C1_SDA	ehrpwm0_tripzone_input	pr1_uart0_rxd	pr1_edio_data_in0	pr1_edio_data_out0	gpio0_04
P1.13	USB1_ID	P1.13 (USB1-ID)	P17	L14	USB1_ID	•	•	•	•	•	•	•
P1.14	+3.3V	P1.14 (VOUT-3.3V)		F6 & F7 & G6 & G7	VOUT-3.3V							
P1.15	USB1_GND	P1.15 (GND)			GND							
P1.16	GND	P1.16 (GND)			GND							
P1.17	AIN(1.8V)-	P1.17 (VREFN)	A9	B9	VREFN							
P1.18	AIN(1.8V)A+	P1.18 (VREFP)	B9	B7	VREFP							
P1.19	AIN(1.8V)0	P1.19 (AIN0-1.8V)	B6	A8	ain0							
P1.20	20	P1.20 (PRU0.16)	D14	B4	xdma_event_intr1	•	tclkin	clkout2	timer7	pr1_pru0_pru_r31_16	EMU3	gpio0_20
P1.21	AIN(1.8V)1	P1.21 (AIN1-1.8V)	C7	B8	ain1							
P1.22	GND	P1.22 (GND)			GND							
P1.23	AIN(1.8V)2	P1.23 (AIN2-1.8V)	B7	B6	ain2							

continues on next page

Table 11.10 – continued from previous page

Header.Pin	Silkscreen	PocketBeagle wiring	Proc Ball	SiP Ball	Mode0 (Name)	Mode1	Mode2	Mode3	Mode4	Mode5	Mode6	Mode7
P1.24	VOUT	P1.24 (VOUT-5V)		K6 & K7 & L6 & L7	VOUT-5V							
P1.25	AIN(1.8V)3	P1.25 (AIN3-1.8V)	A7	C6	ain3							
P1.26	I2C2_SDA	P1.26 (I2C2-SDA)	D18	B10	uart1_ctsn	timer6	dcan0_tx	I2C2_SDA	spi1_cs0	pr1_uart0_cts_n	pr1_edc_latch0_in	gpio0_12
P1.27	AIN(1.8V)4	P1.27 (AIN4-1.8V)	C8	C7	ain4							
P1.28	I2C2_SCL	P1.28 (I2C2-SCL)	D17	A10	uart1_rtsn	timer5	dcan0_rx	I2C2_SCL	spi1_cs1	pr1_uart0_rts_n	pr1_edc_latch1_in	gpio0_13
P1.29	PRU0_7	P1.29 (PRU0.7)	A14	C4	mcasp0_ahclkx	eQEP0_strobe	mcasp0_axr3	mcasp1_axr1	EMU4	pr1_pru0_pru_r30_7	pr1_pru0_pru_r31_7	gpio3_21
P1.30	U0_TX	P1.30 (UART0-TX)	E16	B12	uart0_txd	spi1_cs1	dcan0_rx	I2C2_SCL	eCAP1_in_PWM1_out	pr1_pru1_pru_r30_15	pr1_pru1_pru_r31_15	gpio1_11
P1.31	PRU0_4	P1.31 (PRU0.4)	B12	A3	mcasp0_aclkr	eQEP0A_in	mcasp0_axr2	mcasp1_aclkx	mmc0_sdwp	pr1_pru0_pru_r30_4	pr1_pru0_pru_r31_4	gpio3_18
P1.32	U0_RX	P1.32 (UART0-RX)	E15	A12	uart0_rxd	spi1_cs0	dcan0_tx	I2C2_SDA	eCAP2_in_PWM2_out	pr1_pru1_pru_r30_14	pr1_pru1_pru_r31_14	gpio1_10
P1.33	PRU0_1	P1.33 (PRU0.1)	B13	A2	mcasp0_fsx	ehrpwm0B	•	spi1_d0	mmc1_sdcd	pr1_pru0_pru_r30_1	pr1_pru0_pru_r31_1	gpio3_15
P1.34	26	P1.34 (GPIO0.26)	T11	R5	gpmc_ad10	lcd_data21	mmc1_dat2	mmc2_dat6	ehrpwm2_tripzone_input	pr1_mii0_txen	•	gpio0_26
P1.35	P1.10	P1.35 (PRU1.10)	V5	F1	lcd_pclk	gpmc_a10	pru_mii0_crs	pr1_edio_data_in4	pr1_edio_data_out4	pr1_pru1_pru_r30_10	pr1_pru1_pru_r31_10	gpio2_24
P1.36	PWM0A	P1.36 (PWM0A)	A13	A1	mcasp0_aclkx	ehrpwm0A	•	spi1_sclk	mmc0_sdcd	pr1_pru0_pru_r30_0	pr1_pru0_pru_r31_0	gpio3_14

11.7.3 P2 Header

Figure 44 shows the schematic diagram for the P2 Header.

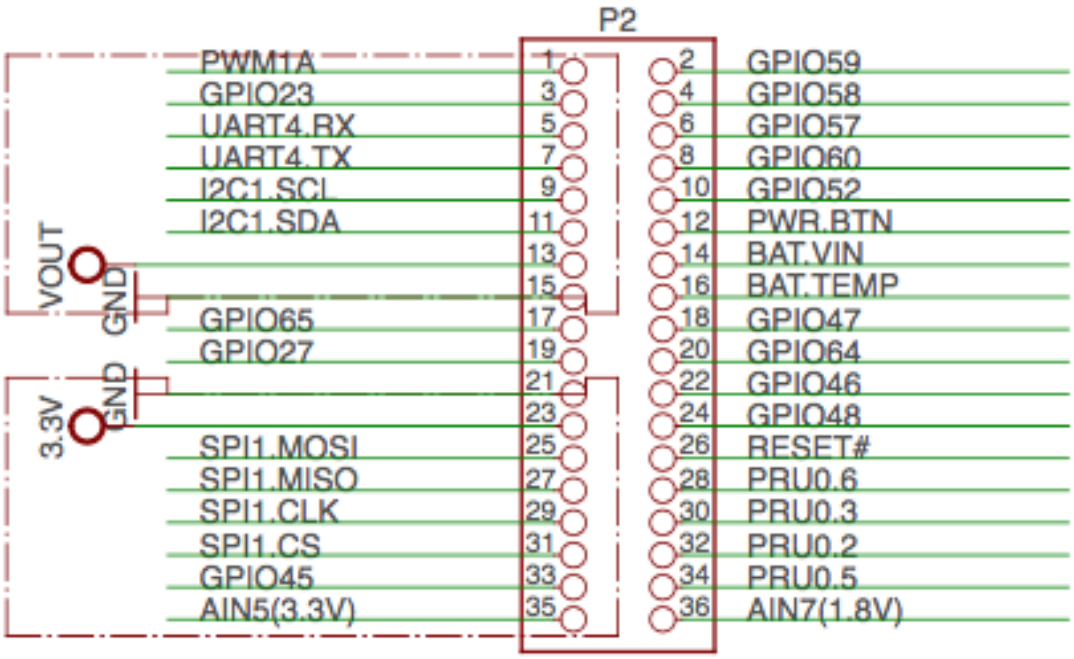


Fig. 11.39: P2 Header

Use scroll bar at bottom of chart to see additional features in columns to the right. When printing this document you will need to print this chart separately.

Table 11.11: P2 Header Pinout

Header.Pin	Silkscreen	PocketBeagle wiring	Proc Ball	SiP Ball	Mode0 (Name)	Mode1	Mode2	Mode3	Mode4	Mode5	Mode6	Mode7
P2.01	PWM1A	P2.01 (PWM1A)	U14	P12	gpmc_a2	gmii2_txd3	rgmii2_td3	mmc2_dat1	gpmc_a18	pr1_mii1_txd2	ehrpwm1A	gpio1_18
P2.02	59	P2.02 (GPIO1.27)	V17	T16	gpmc_a11	gmii2_rxd0	rgmii2_rd0	rmii2_rxd0	gpmc_a27	pr1_mii1_rxer	mcasp0_axr1	gpio1_27
P2.03	23	P2.03 (GPIO0.23)	T10	P5	gpmc_d9	lcd_data22	mmc1_dat1	mmc2_dat5	ehrpwm2B	pr1_mii0_col	•	gpio0_23
P2.04	58	P2.04 (GPIO1.26)	T16	R15	gpmc_a10	gmii2_rxd1	rgmii2_rd1	rmii2_rxd1	gpmc_a26	pr1_mii1_rxdv	mcasp0_axr0	gpio1_26
P2.05	U1_RX	P2.05 (UART4-RX)	T17	P15	gpmc_wait0	gmii2_crs	gpmc_csn4	rmii2_crs_dv	mmc1_sdcd	pr1_mii1_col	uart4_rxd	gpio0_30
P2.06	57	P2.06 (GPIO1.25)	U16	T15	gpmc_a9	gmii2_rxd2	rgmii2_rd2	mmc2_dat7 / rmii2_crs_dv	gpmc_a25	pr1_mii_mr1_clk	mcasp0_fsx	gpio1_25
P2.07	U1_TX	P2.07 (UART4-TX)	U17	R16	gpmc_wp	gmii2_rxerr	gpmc_csn5	rmii2_rxerr	mmc2_sdcd	pr1_mii1_txen	uart4_txd	gpio0_31
P2.08	60	P2.08 (GPIO1.28)	U18	N14	gpmc_be1n	gmii2_col	gpmc_csn6	mmc2_dat3	gpmc_dir	pr1_mii1_rxlink	mcasp0_aclkr	gpio1_28
P2.09	I2C1_SCL	P2.09 (I2C1-SCL)	D15	B11	uart1_txd	mmc2_sdwp	dcan1_rx	I2C1_SCL	•	pr1_uart0_txd	pr1_pru0_pru_r31_1	gpio0_15
P2.10	52	P2.10 (GPIO1.20)	R14	R13	gpmc_a4	gmii2_txd1	rgmii2_td1	rmii2_txd1	gpmc_a20	pr1_mii1_txd0	eQEP1A_in	gpio1_20
P2.11	I2C1_SDA	P2.11 (I2C1-SDA)	D16	A11	uart1_rxd	mmc1_sdwp	dcan1_tx	I2C1_SDA	•	pr1_uart0_rxd	pr1_pru1_pru_r31_1	gpio0_14
P2.12	PB	P2.12 (POWER_BTN)		T11	POWER							
P2.13	VOUT	P2.13 (VOUT-5V)		K6, K7, L6, L7	VOUT-5V							
P2.14	BAT +	P2.14 (VIN-BAT)		P8, R8, T8	VIN-BAT							
P2.15	GND	P2.15 (GND)			GND							
P2.16	BAT -	P2.16 (BAT-TEMP)		N6	BAT-TEMP							
P2.17	65	P2.17 (GPIO2.1)	V12	T7	gpmc_clk	lcd_memory_clk	gpmc_wait1	mmc2_clk	pr1_mii1_crs	pr1_mdio_mdclk	mcasp0_fsr	gpio2_01
P2.18	47	P2.18 (PRU0.15i)	U13	P7	gpmc_ad15	lcd_data16	mmc1_dat7	mmc2_dat3	eQEP2_strobe	pr1_ecap0_ecap_capin_apwm_o	pr1_pru0_pru_r31_15	gpio1_15P
P2.19	27	P2.19 (GPIO0.27)	U12	T5	gpmc_ad11	lcd_data20	mmc1_dat3	mmc2_dat7	ehrpwm0_synco	pr1_mii0_txd3	•	gpio0_27
P2.20	64	P2.20 (GPIO2.0)	T13	R7	gpmc_csn3	gpmc_a3	rmii2_crs_dv	mmc2_cmd	pr1_mii0_crs	pr1_mdio_data	EMU4	gpio2_00
P2.21	GND	P2.21 (GND)			GND							
P2.22	46	P2.22 (GPIO1.14)	V13	T6	gpmc_ad14	lcd_data17	mmc1_dat6	mmc2_dat2	eQEP2_index	pr1_mii0_txd0	pr1_pru0_pru_r31_14	gpio1_14
P2.23	+3.3V	P2.23 (VOUT-3.3V)		F6 & F7 & G6 & G7	VOUT-3.3V							
P2.24	48	P2.24 (GPIO1.12)	T12	P6	gpmc_ad12	lcd_data19	mmc1_dat4	mmc2_dat0	eQEP2A_in	pr1_mii0_txd2	pr1_pru0_pru_r30_14	gpio1_12

continues on next page

Table 11.11 – continued from previous page

Header.Pin	Silkscreen	PocketBeagle wiring	Proc Ball	SiP Ball	Mode0 (Name)	Mode1	Mode2	Mode3	Mode4	Mode5	Mode6	Mode7
P2.25	SPI1_MOSI	P2.25 (SPI1-MOSI)	E17	C13	uart0_rtsn	uart4_txd	dcan1_rx	I2C1_SCL	spi1_d1	spi1_cs0	pr1_edc_sync1_out	gpio1_09
P2.26	RST	P2.26 (NRESET)	A10	R11	nRESETIN_OUT	•	•	•	•	•	•	•
P2.27	SPI1_MISO	P2.27 (SPI1-MISO)	E18	C12	uart0_ctsn	uart4_rxd	dcan1_tx	I2C1_SDA	spi1_d0	timer7	pr1_edc_sync0_out	gpio1_08
P2.28	PRU0_6	P2.28 (PRU0.6)	D13	C3	mcasp0_axr1	eQEP0_index	•	mcasp1_axr0	EMU3	pr1_pru0_pru_r30_6	pr1_pru0_pru_r31_6	gpio3_20
P2.29	SPI1_CLK	P2.29 (SPI1-CLK)	C18	C5	eCAP0_in_PWM0_out	uart3_txd	spi1_cs1	pr1_ecap0_ecap_capin_apwm_o	spi1_sclk	mmc0_sdwp	xdma_event_intr2	gpio0_7
P2.30	PRU0_3	P2.30 (PRU0.3)	C12	B1	mcasp0_ahclkr	ehrpwm0_synci	mcasp0_axr2	spi1_cs0	eCAP2_in_PWM2_out	pr1_pru0_pru_r30_3	pr1_pru0_pru_r31_3	gpio3_17
P2.31	SPI1_CS	P2.31 (SPI1-CS1)	A15	A4	xdma_event_intr0	•	timer4	clkout1	spi1_cs1	pr1_pru1_pru_r31_16	EMU2	gpio0_19
P2.32	PRU0_2	P2.32 (PRU0.2)	D12	B2	mcasp0_axr0	ehrpwm0_tripzone_input	•	spi1_d1	mmc2_sdcd	pr1_pru0_pru_r30_2	pr1_pru0_pru_r31_2	gpio3_16
P2.33	45	P2.33 (GPIO1.13)	R12	R6	gpmc_ad13	lcd_data18	mmc1_dat5	mmc2_dat1	eQEP2B_in	pr1_mii0_txd1	pr1_pru0_pru_r30_15	gpio1_13
P2.34	PRU0_5	P2.34 (PRU0.5)	C13	B3	mcasp0_fsr	eQEP0B_in	mcasp0_axr3	mcasp1_fsx	EMU2	pr1_pru0_pru_r30_5	pr1_pru0_pru_r31_5	gpio3_19
P2.35	A5/86	P2.35 (AIN5/GPIO86)	B8	C8	ain5							
P2.35	A5/86	P2.35 (AIN5/GPIO86)	U5	F3	lcd_vsync	gpmc_a8	gpmc_a1	pr1_edio_data_in2	pr1_edio_data_out2	pr1_pru1_pru_r30_8	pr1_pru1_pru_r31_8	gpio2_22
P2.36	A7(1.8)	P2.36 (AIN7)		N13	ain7							

11.7.4 mikroBUS socket connections

mikroBUS and, by extension "mikroBUS Click boards", are trademarks of MikroElektronika. We do not make any claims of compatibility nor adherence to their specification. We've just seen that many of the Click boards "just work".

The Expansion Headers on PocketBeagle have been designed to accept up to two Click Boards added to the header pins at the same time. This provides an exciting opportunity to add functionality easily to PocketBeagle from 'hundreds of existing add-on Click Boards'.

The mikroBUS standard comprises a pair of 1×8 female headers with a standardized pin configuration. The pinout (always laid out in the same order) consists of three groups of communications pins (SPI, UART and I2C), six additional pins (PWM, Interrupt, Analog input, Reset and Chip select), and two power groups (+3.3V and 5V).

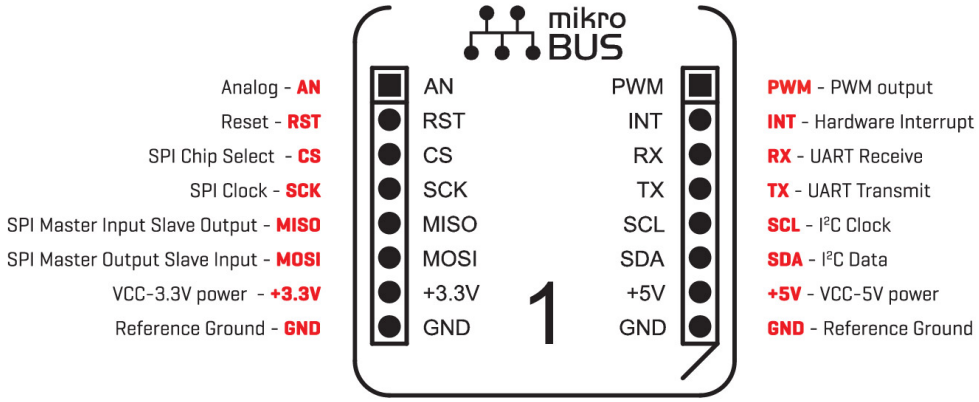


Fig. 11.40: mikroBUS

The Expansion Header pin alignment enables 2 Click Boards on the top side of PocketBeagle using the inside rails of the headers. This leaves the outside rails open to be accessed from either the top or the bottom of PocketBeagle. Place each Click Board into the position shown in Figure 46, with one Click Board facing each direction. When choosing Click boards, make sure you are checking that they meet the 3.3V requirements for PocketBeagle. A growing number of community members are trying out various Click Boards and posting results on the 'PocketBeagle Wiki mikroBus Click Boards page'.

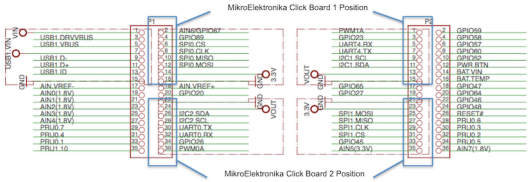
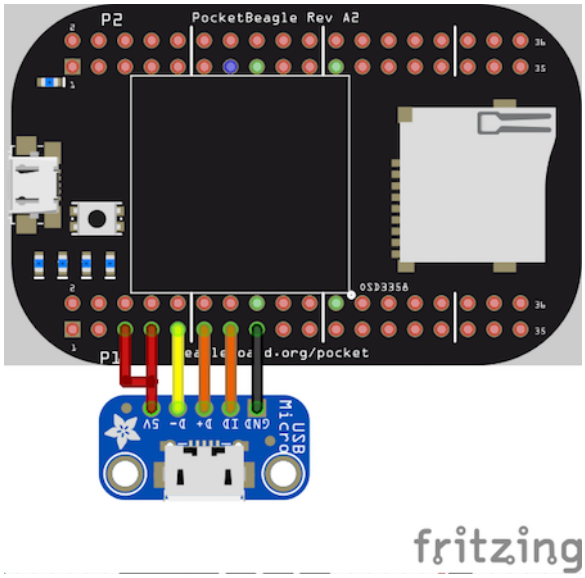


Fig. 11.41: PocketBeagle Both Headers

11.7.5 Setting up an additional USB Connection

You can add an additional USB connection to PocketBeagle easily by connecting a microUSB breakout. By default in the current software, the system should be configured to use this port as a host. Keep up to date on this project on the 'PocketBeagle Wiki FAQ'.



11.8 PocketBeagle Cape Support

This is a placeholder for recommendations for those building their own PocketBeagle Cape designs. If you'd like to join the conversation 'check out the discussion on the forum for PocketBeagle'

See also PocketBeagle under 'BeagleBoard Capes'

11.9 PocketBeagle Mechanical

11.9.1 9.1 Dimensions and Weight

Size: 2.21" x 1.38" (56mm x 35mm)

Max height: .197" (5mm)

PCB size: 55mm x 35mm

PCB Layers: 4

PCB thickness: 1.6mm

RoHS Compliant: Yes

Weight: 10g

Rough model can be found at PocketBeagle models

11.10 Additional Pictures

11.11 Support Information

All support for this design is through the BeagleBoard.org community at:

- beagleboard@googlegroups.com or
- beagleboard.org/discuss.

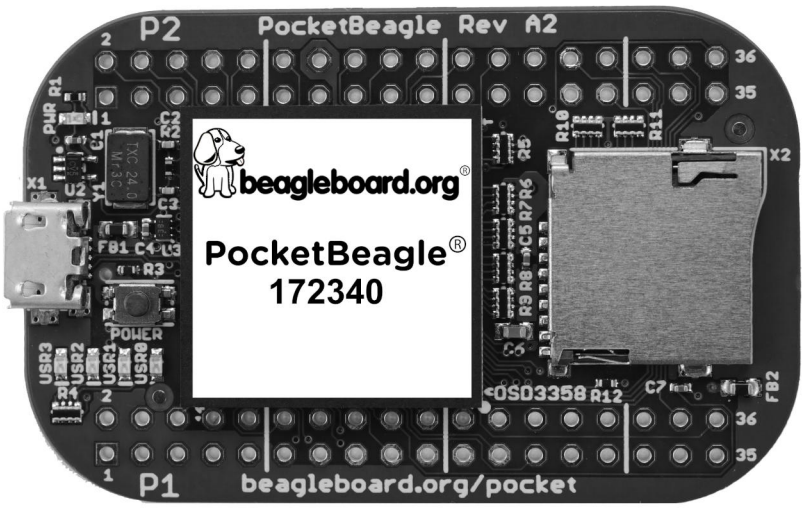


Fig. 11.42: PocketBeagle Front BW

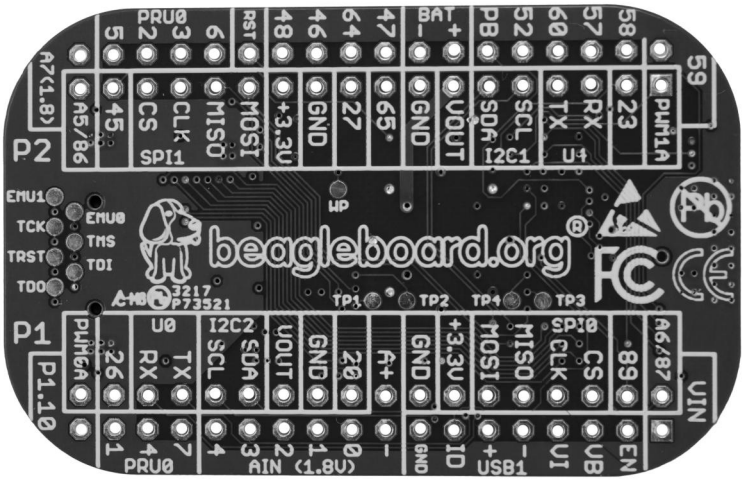


Fig. 11.43: PocketBeagle Back BW

11.11.1 Hardware Design

Design documentation can be found on the wiki. https://git.beagleboard.org/beagleboard/pocketbeagle/ Including:

- Schematic in PDF https://git.beagleboard.org/beagleboard/pocketbeagle/-/blob/master/PocketBeagle_sch.pdf
- Schematic and layout in EAGLE https://git.beagleboard.org/beagleboard/pocketbeagle/-/tree/master/EAGLE
- Schematic and layout in KiCAD https://git.beagleboard.org/beagleboard/pocketbeagle/-/tree/master/KiCAD
- Bill of Materials https://git.beagleboard.org/beagleboard/pocketbeagle/-/blob/master/PocketBeagle_BOM.csv
- *PocketBeagle* docs.

11.11.2 Software Updates

It is a good idea to always use the latest software. Instructions for how to update your software to the latest version can be found at:

Download the latest software files from www.beagleboard.org/distros

11.11.3 Export Information

- ECCN: EAR99
- CCATS: G173833
- Documentation: PocketBeagle_Export_Classification.pdf

11.11.4 RMA Support

If you feel your board is defective or has issues and before returning merchandise, please seek approval from the manufacturer using beagleboard.org/support/rma. You will need the manufacturer, model, revision and serial number of the board.

11.11.5 Getting Help

If you need some up to date troubleshooting techniques, the Wiki is a great place to start PocketBeagle wiki.

If you need professional support, check out beagleboard.org/resources.

Chapter 12

BeagleConnect Freedom

BeagleConnect™ Freedom is an open-hardware wireless hardware platform developed by BeagleBoard.org and built around the TI CC1352P7 microcontroller, which supports both 2.4-GHz and long-range, low-power Sub-1 GHz wireless protocols. Rapidly prototyping of IoT applications is accelerated by hardware compatibility with over 1,000 mikroBUS add-on sensors, acutators, indicators and additional connectivity and storage options, and backed with software support utilizing the Zephyr scalable and modular real-time operating system, allowing developers to tailor the solution to their specific needs. BeagleConnect Freedom further includes MSP430F5503 for USB-to-UART functionality, temperature and humidity sensor, light sensor, SPI flash, battery charger, buzzer, LEDs, and JTAG connections to make it a comprehensive solution for IoT development and prototyping.

The TI CC1352P7 microcontroller (MCU) includes a 48-MHz Arm Cortex-M4F processor, 704KB Flash memory, 256KB ROM, 8KB Cache SRAM, 144KB of ultra-low leakage SRAM, and over-the-air upgrades (OTA) capability. This MCU provides flexible support for many different protocols and bands making it suitable for many different communication requirements.



Important: This is a work in progress, for latest documentation please visit https://docs.beagleboard.org/latest/

OSHW
US002175

License Terms

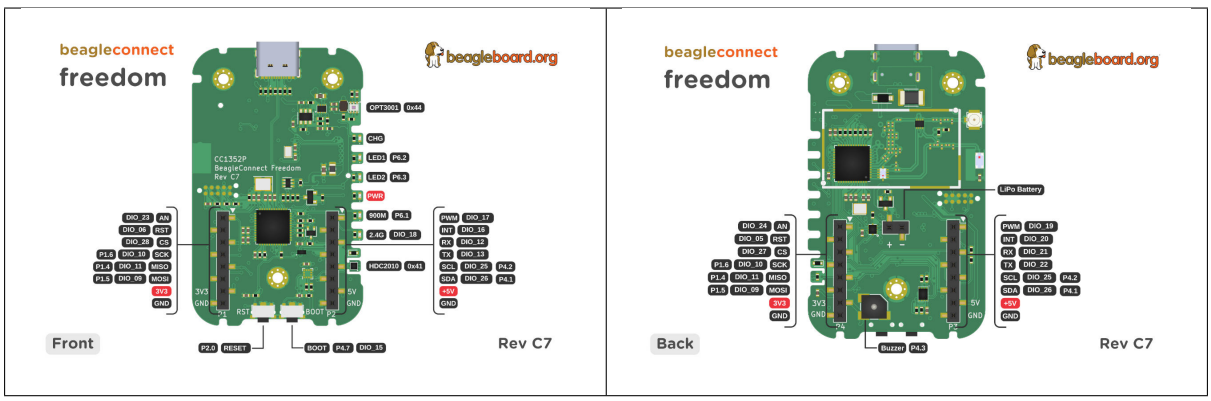
- This documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
- Design materials and license can be found in the git repository
- Use of the boards or design materials constitutes an agreement to the *Terms & Conditions*
- Software images and purchase links available on the board page
- For export, emissions and other compliance, see *Support*

12.1 Introduction

12.1.1 What is BeagleConnect™ Freedom?

BeagleConnect™ Freedom is based on a TI Arm Cortex-M4 wireless-enabled microcontroller and is the first available BeagleConnect™ solution. It features:

- BeagleConnect™ node device for Bluetooth Low-Energy (BLE) and Sub-GHz 802.15.4 long range wireless,
- Works with BeaglePlay® gateway,
- USB-based serial console and firmware updates,
- 2x mikroBUS sockets,
- On-board light and humidity/temperature sensors,
- Battery-charger circuit, and
- Buzzer, LEDs and buttons for user programming.





12.1.2 What makes BeagleConnect™ new and different?

Plug & Play approach

BeagleConnect™ uses the collaboratively developed Linux kernel to contain the intelligence required to speak to these devices (sensors, actuators, and indicators), rather than relying on writing code on a microcontroller specific to these devices. Some existing solutions rely on large libraries of microcontroller code, but the integration of communications, maintenance of the library with a limited set of developer resources and other constraints to be explained later make those other solutions less suitable for rapid prototyping than BeagleConnect™.

Linux presents these devices abstractly in ways that are self-descriptive. Add an accelerometer to the system and you are automatically fed a stream of force values in standard units. Add a temperature sensor and you get it back in standard units again. Same for sensing magnetism, proximity, color, light, frequency, orientation, or multitudes of other inputs. Indicators, such as LEDs and displays, are similarly abstracted with a few other kernel subsystems and more advanced actuators with and without feedback control are in the process of being developed and standardized. In places where proper Linux kernel drivers exist, no new specialized code needs to be created for the devices.

Important: BeagleConnect™ solves IoT in a different and better way than any previous solution. For hundreds of devices, users won't have to write a single line of code to add them their systems. The automation code they do write can be extremely simple, done with graphical tools or in any language they want. Maintenance of the code is centralized in a small reusable set of microcontroller firmware and the Linux kernel, which is highly peer reviewed under a highly-regarded governance model.

Reliable software update mechanism

Because there isn't code specific to any given network-of-devices configuration , we can all leverage the same software code base. This means that when someone fixes an issue in either BeagleConnect™ firmware or the Linux kernel, you benefit from the fixes. The source for BeagleConnect™ firmware is also submitted to the Zephyr Project upstream, further increasing the user base. Additionally, we will maintain stable branches of the software and provide mechanisms for updating firmware on BeagleConnect™ hardware. With a single, rela-

tively small firmware load, the potential for bugs is kept low. With large user base, the potential for discovering and resolving bugs is high.

Rapid prototyping without wiring

BeagleConnect™ utilizes the mikroBUS standard. The mikroBUS standard interface is flexible enough for almost any typical sensor or indicator with hundreds of devices available.

Note: Currently, we have support in the Linux kernel for a bit over 100 Click mikroBUS add-on boards from Mikroelektronika and are working with Mikroelektronika on a updated version of the specification for these boards to self-identify. Further, eventually the vast majority of over 800 currently available Click mikroBUS add-on boards will be supported as well as the hundreds of compliant boards developed every year.

Long-range, low-power wireless

BeagleConnect™ Freedom wireless hardware is built around a TI CC1352P7 multiprotocol and multi-band Sub-1 GHz and 2.4-GHz wireless microcontroller (MCU). CC1352P7 includes a 48-MHz Arm® Cortex®-M4F processor, 704KB Flash, 256KB ROM, 8KB Cache SRAM, 144KB of ultra-low leakage SRAM, and Over-the-Air upgrades (OTA).

Fully customizable design

BeagleConnect™ utilizes open source hardware and open source software, making it possible to optimize hardware and software implementations and sourcing to meet end-product requirements. BeagleConnect™ is meant to enable rapid-prototyping and not to necessarily satisfy any particular end-product's requirements, but with full considerations for go-to-market needs.

Each BeagleBoard.org BeagleConnect™ solution will be:

- Readily available for over 10 years,
- Built with fully open source software with submissions to mainline Linux and Zephyr repositories to aide in support and porting,
- Built with fully open source and non-restrictive hardware design including schematic, bill-of-materials, layout, and manufacturing files (with only the BeagleBoard.org logo removed due to licensing restrictions of our brand),
- Built with parts where at least a compatible part is available from worldwide distributors in any quantity,
- Built with design and manufacturing partners able to help scale derivative designs,
- Based on a security model using public/private keypairs that can be replaced to secure your own network, and
- Fully FCC/CE certified.

12.2 Quick Start Guide

12.2.1 What's included in the box?

1. BeagleConnect Freedom board in enclosure
2. Antenna
3. USB cable
4. Quick-start card

Todo: Image with what's inside the box and a better description.



12.2.2 Attaching antenna

To connect the SubGHz antenna with SMA connector to the BeagleConnect Freedom you just have to align, place and rotate the antenna clockwise as shown in the image below. To detach the antenna just twist it anti-clockwise.



Fig. 12.1: Aattaching antenna to BeagleConnect Freedom

12.2.3 Tethering to PC

Todo: Describe how to get a serial connection.

12.2.4 Wireless Connection

Todo: Describe how to get an IEEE802.15.4g connection from BeaglePlay.

12.2.5 Access Micropython

Boards come pre-flashed with Micropython. Read *Using Micropython* for more details.

Todo: Describe how to get to a local console and websockets console.

12.2.6 Demos and Tutorials

- *Using BeagleConnect Greybus*
- *Using Micropython*
- *Using Zephyr*

12.3 Design

12.3.1 Detailed overview

12.3.2 Detailed hardware design

LEDs

Buttons & Buzzer

Sensors

mikroBUS

USB-C port

Buck converter

LiPo battery charger

Battery input protection

MSP430F5503

CC1352P7

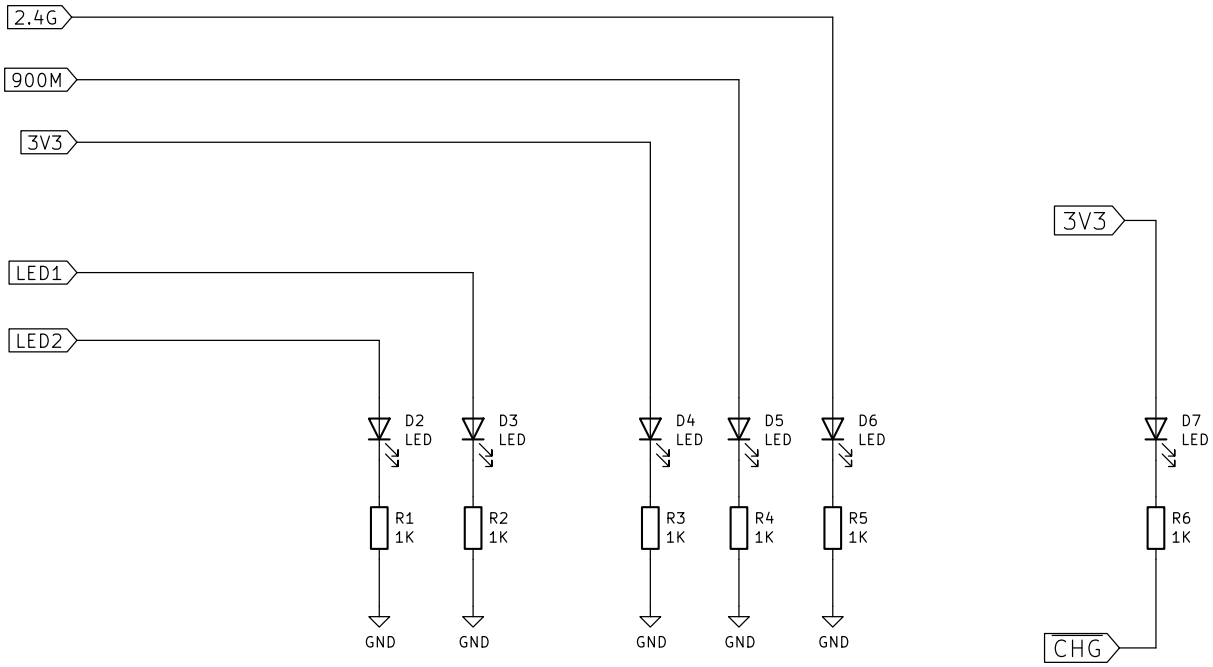


Fig. 12.2: BeagleConnect LEDs

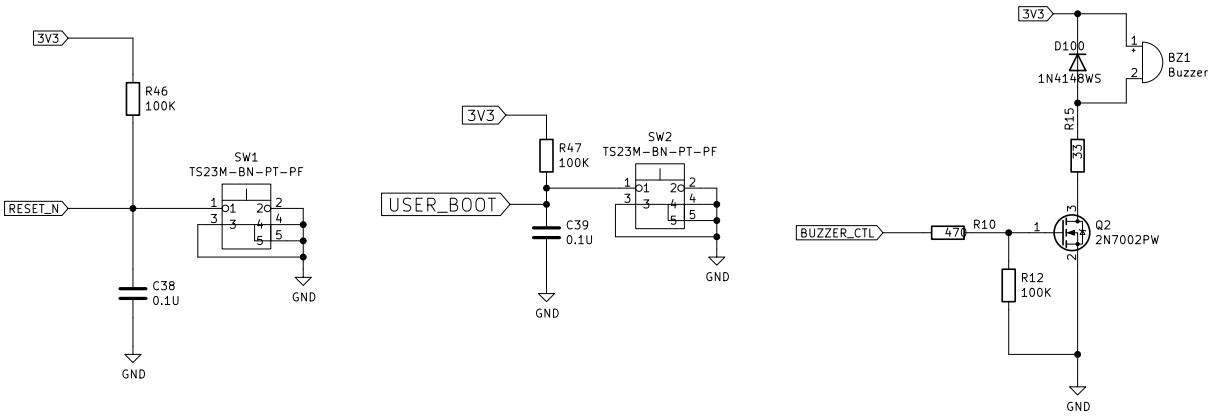


Fig. 12.3: User Input Output (Buttons & Buzzer)

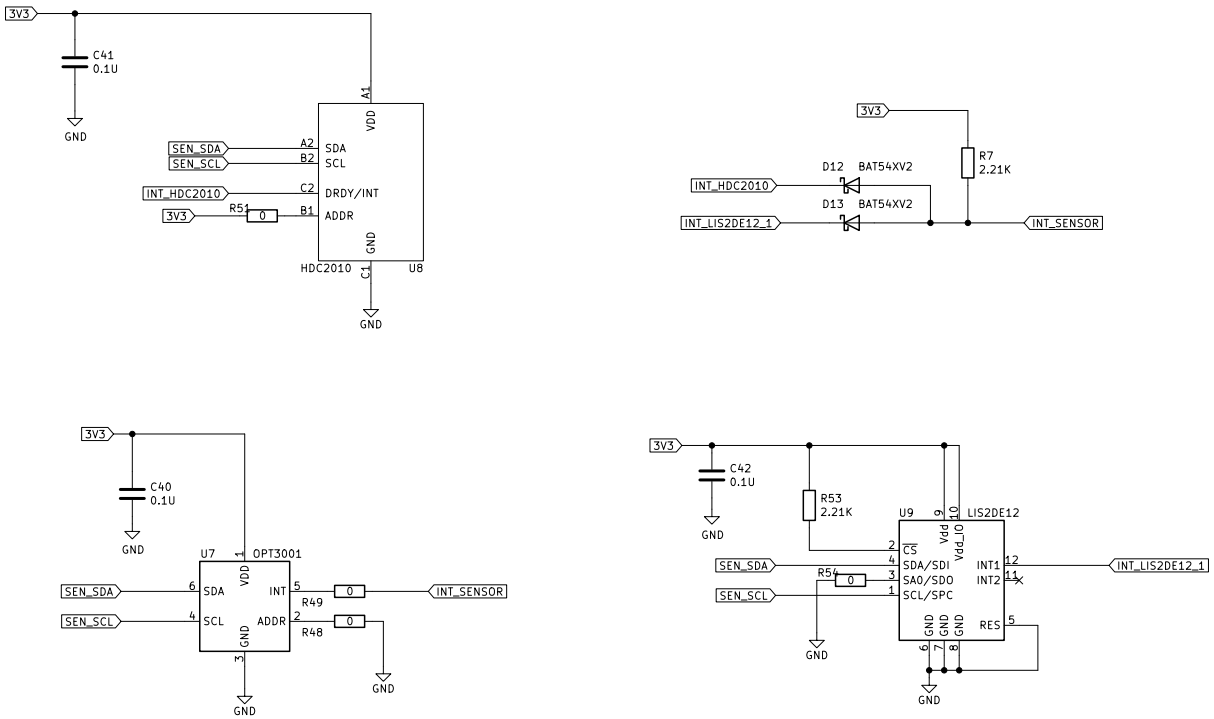


Fig. 12.4: On-board sesnsors

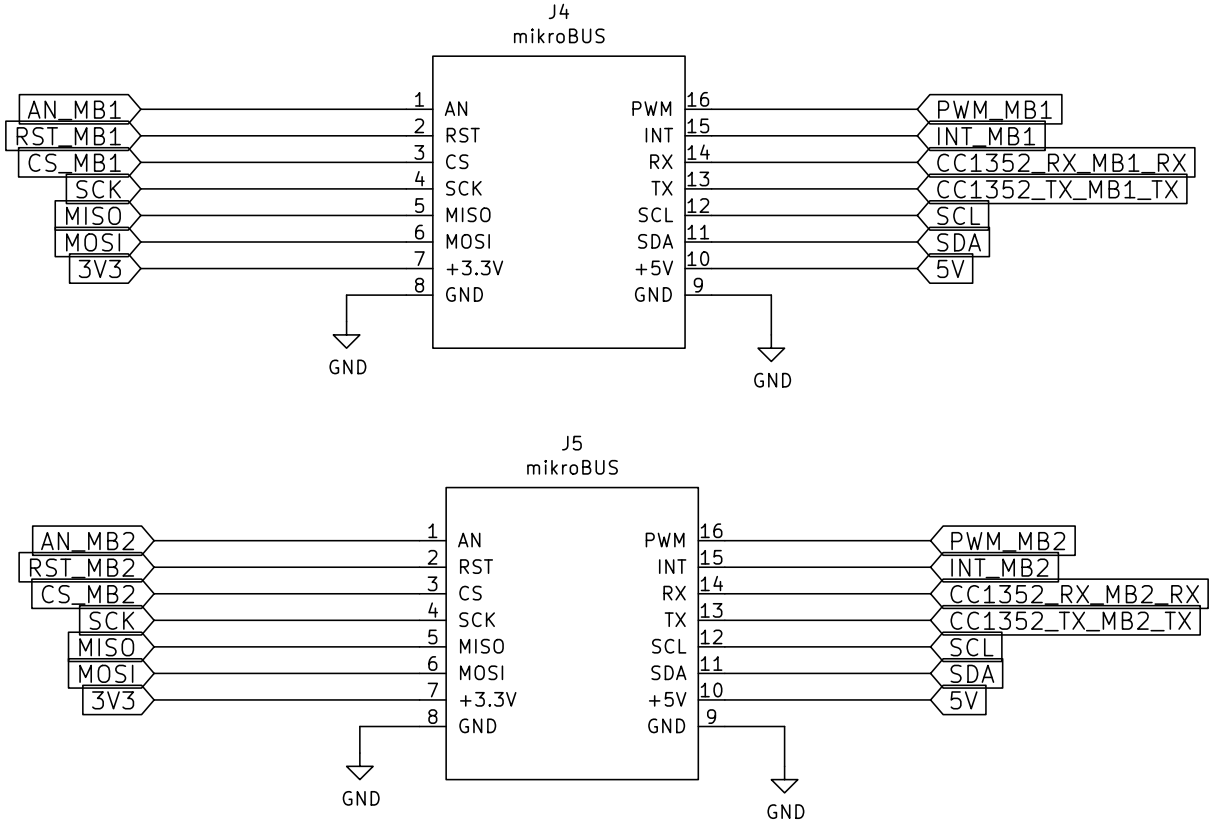


Fig. 12.5: mikroBUS ports

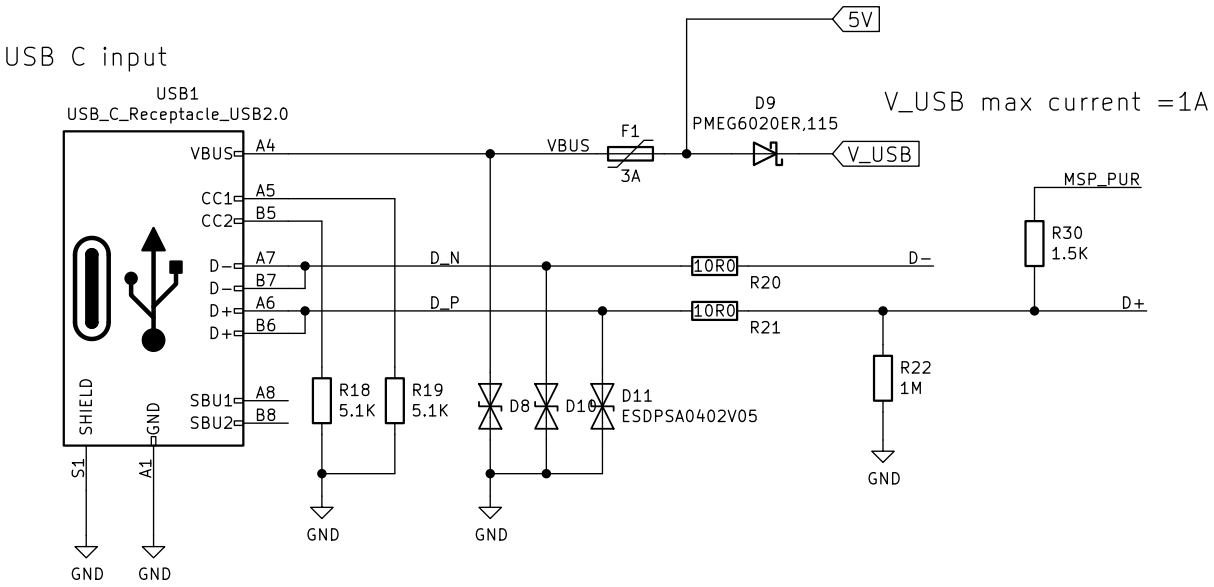


Fig. 12.6: USB-C for power & programming

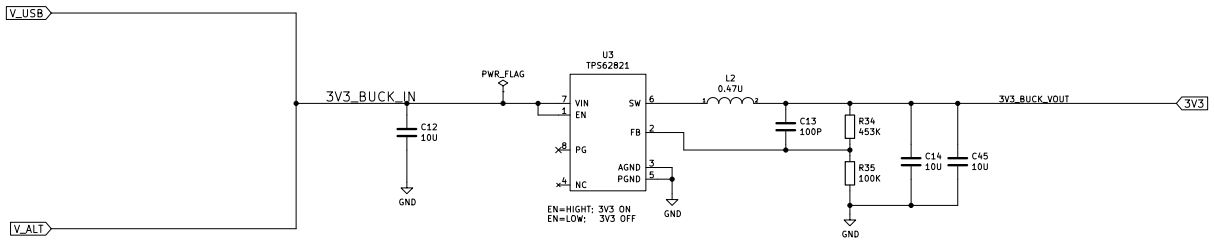


Fig. 12.7: BuckConverter (3.3V output)

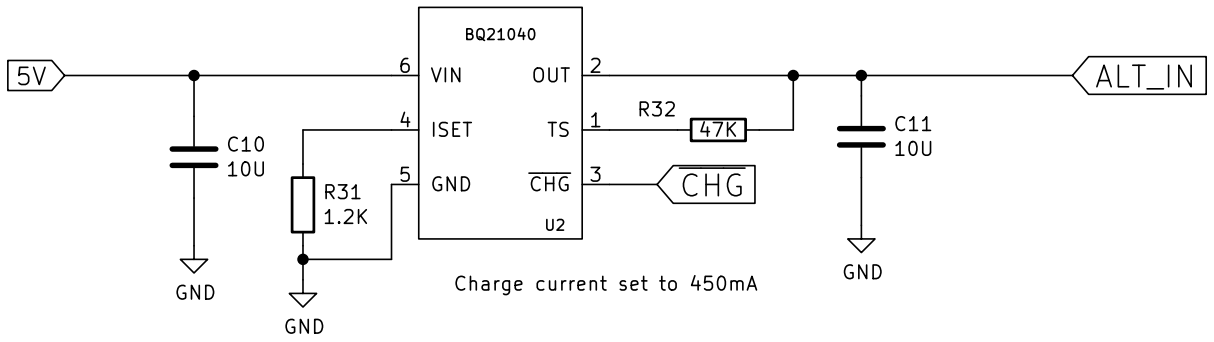


Fig. 12.8: 4.2V LiPo battery charger

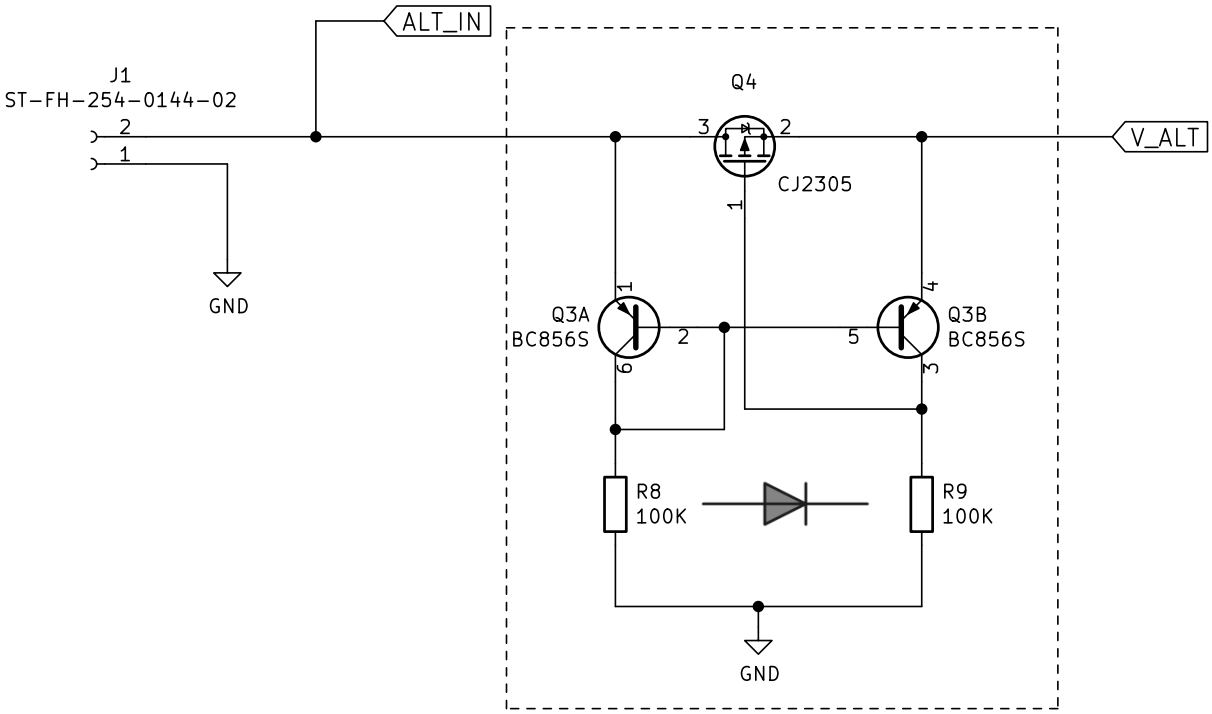


Fig. 12.9: LiPo battery input protection

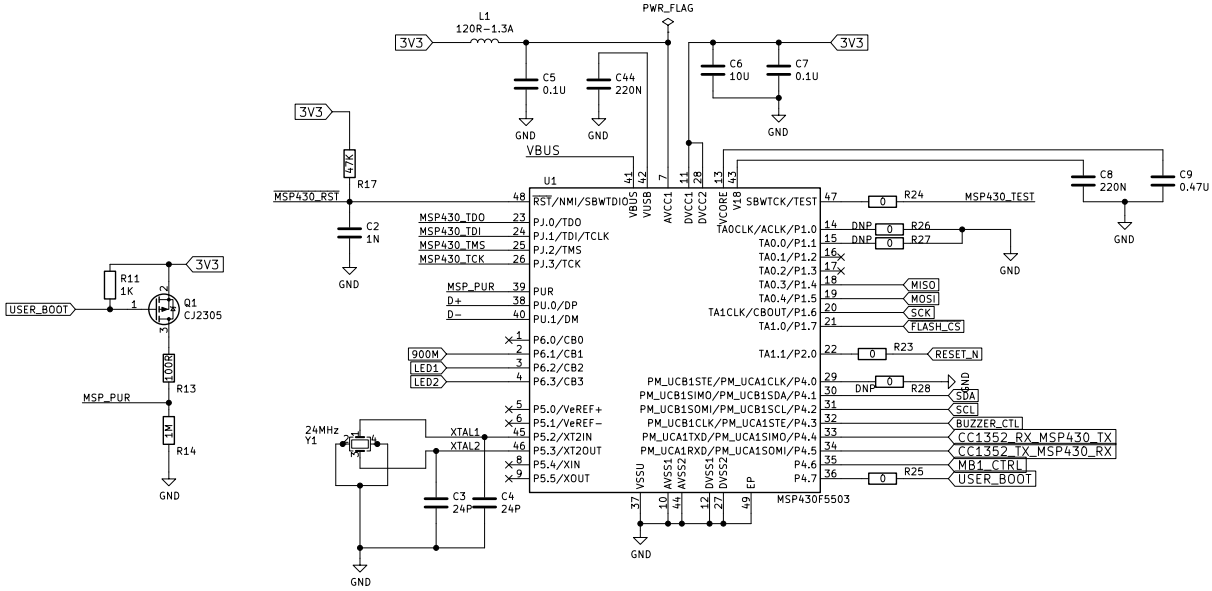


Fig. 12.10: MSP430F5503 (USB to UART & mikroBUS)

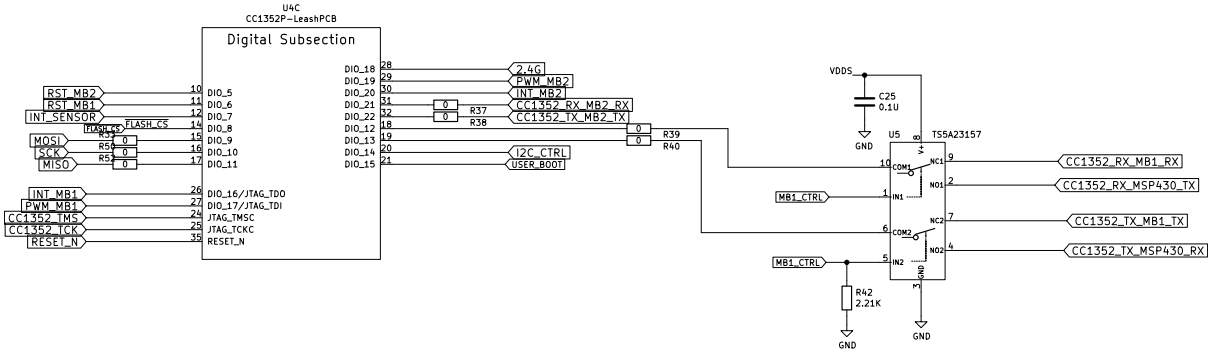


Fig. 12.11: CC1352P7 Digital subsection

Digital subsection

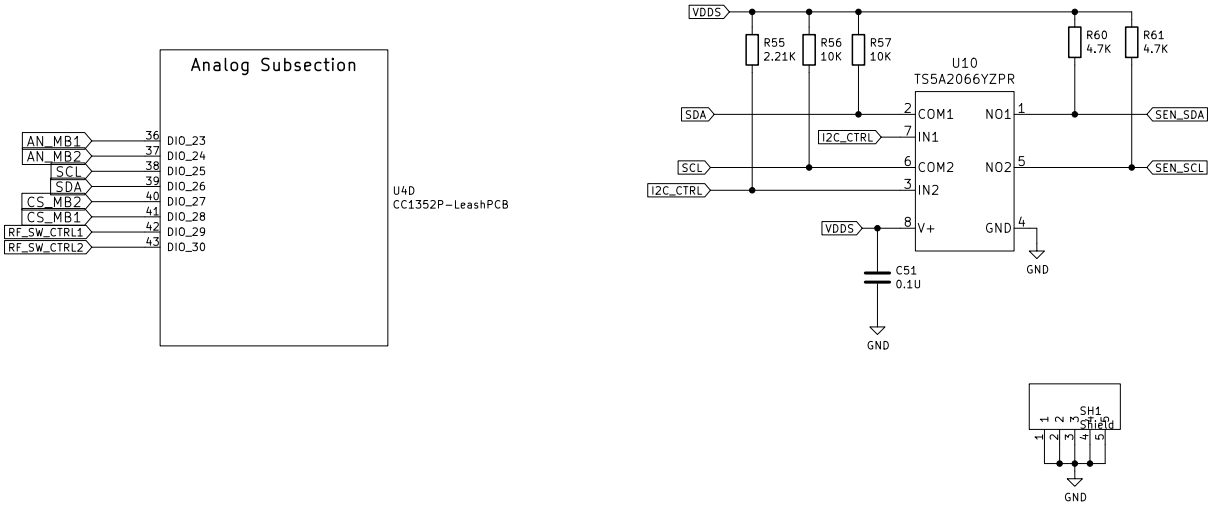


Fig. 12.12: CC1352P7 Analog subsection

Analog subsection

Power subsection

RF subsection

SPI Flash

Debug interface

12.3.3 Mechanical

12.4 Connectors

12.5 Demos & tutorials

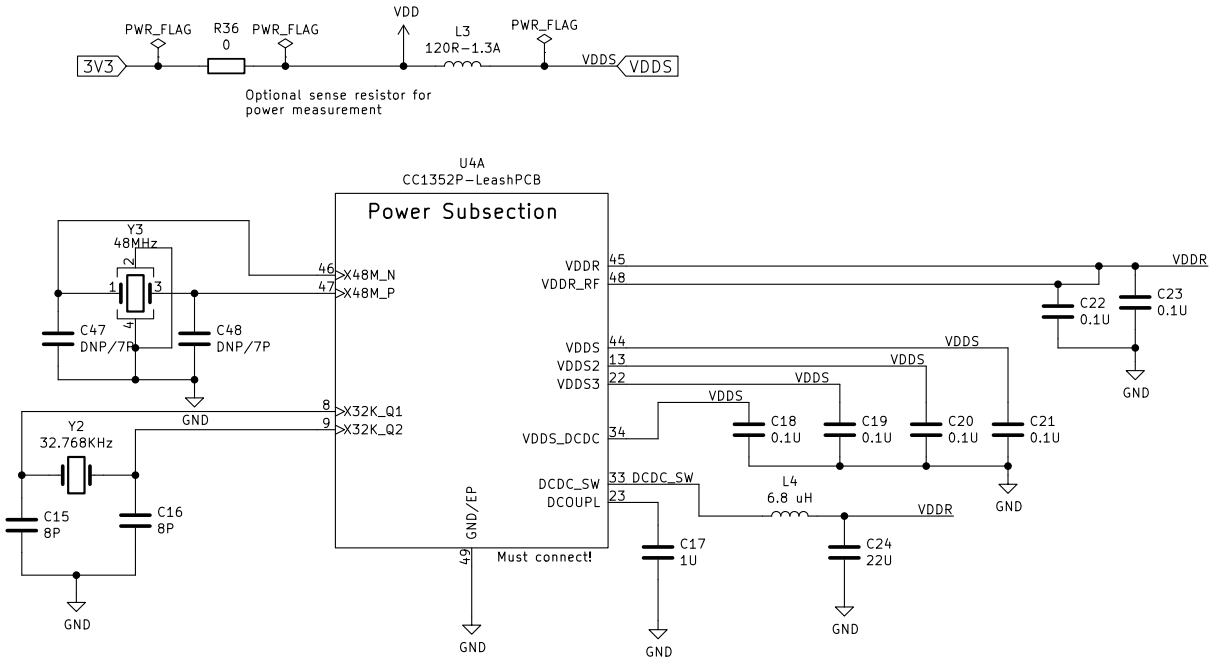


Fig. 12.13: CC1352P7 Power subsection

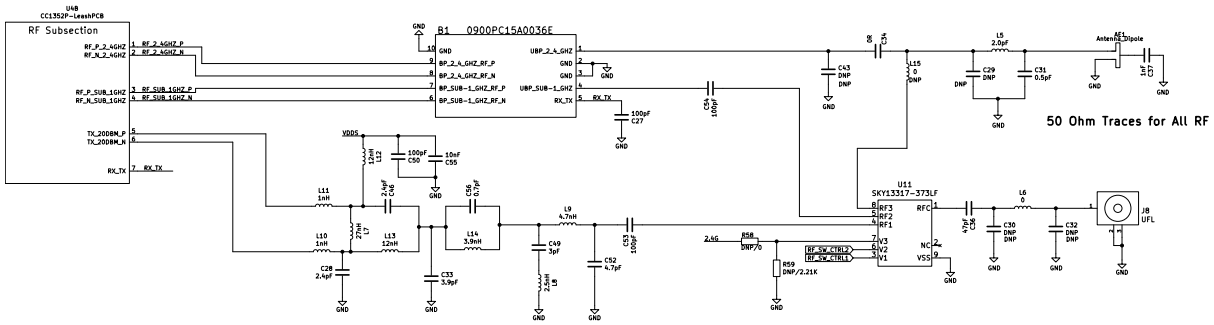


Fig. 12.14: CC1352P7 RF subsection

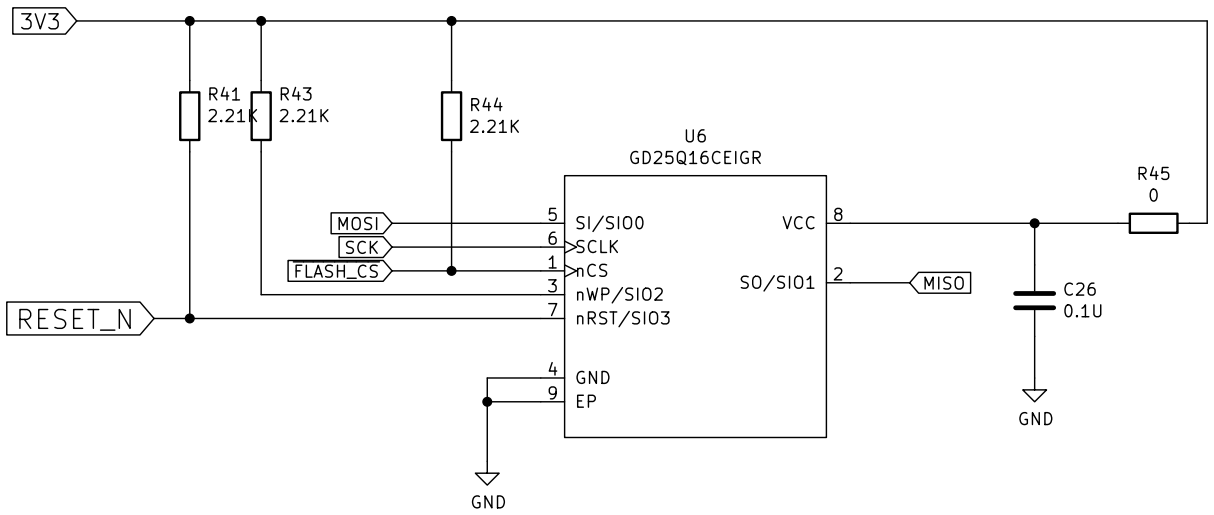


Fig. 12.15: SPIFlash

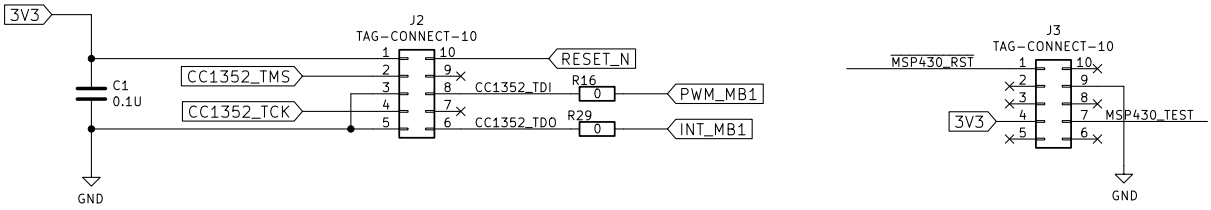


Fig. 12.16: CC1352P7 & MSP430F5503 TagConnect

12.5.1 Using Micropython

Important: Currently under development

Micropython is a great way to get started developing with BeagleConnect Freedom quickly.

Flashed firmware

BeagleConnect Freedom initial production firmware is release 0.0.3 of our own fork of Micropython.

https://git.beagleboard.org/beagleconnect/zephyr/micropython/-/releases/0.0.3

You can verify this version by using `mcumgr` over a UDP connection or `mcuboot` over the serial console shell.

Latest releases are part of our Zephyr SDK releases.

https://git.beagleboard.org/beagleconnect/zephyr/zephyr/-releases

Examples

0.0.3 The first boards were flashed with this firmware.

```
debian@BeaglePlay:~$ sudo systemd-resolve --set-mdns=yes --interface=lowpan0
debian@BeaglePlay:~$ avahi-browse -r -t _zephyr._tcp
+ lowpan0 IPv6 zephyr                                          _zephyr._tcp   ␣
↪      local
= lowpan0 IPv6 zephyr                                          _zephyr._tcp   ␣
↪      local
   hostname = [zephyr.local]
   address = [fe80::3265:842a:4b:1200]
   port = [12345]
   txt = []
debian@BeaglePlay:~$ avahi-resolve -6 -n zephyr.local
zephyr.local    fe80::ec0f:7a22:4b:1200
debian@BeaglePlay:~$ mcumgr conn add bcf0 type="udp" connstring=
↪"[fe80::3265:842a:4b:1200%lowpan0]:1337"
Connection profile bcf0 successfully added
debian@BeaglePlay:~$ mcumgr -c bcf0 image list
Images:
 image=0 slot=0
    version: hu.hu.hu
    bootable: true
    flags: active confirmed
    hash: 3697bcef05a6becda7dc14150d46c05dbed5fa78633657b20cf34e1418affee9
Split status: N/A (0)
debian@BeaglePlay:~$ mcumgr -c bcf0 shell exec "device list"
status=0

devices:
```

(continues on next page)

(continued from previous page)

```
- GPIO_0 (READY)
- random@40028000 (READY)
- UART_1 (READY)
- UART_0 (READY)
- i2c@40002000 (READY)
- I2C_0S (READY)
  requires: GPIO_0
  requires: i2c@40002000
- flash-controller@40030000 (READY)
- spi@40000000 (READY)
  requires: GPIO_0
- ieee802154g (READY)
- gd25q16c@0 (READY)
  requires: spi@40000000
- leds (READY)
- HDC2010-HUMIDITY (READY)
  requires: I2C_0S
-
debian@BeaglePlay:~$ mcumgr -c bcf0 shell exec "net iface"
status=0

Hostname: zephyr


Interface 0x20002de4 (IEEE 802.15.4) [1]
========================================
Link addr : 30:65:84:2A:00:4B:12:00
MTU       : 125
Flags     : AUTO_START,IPv6
IPv6 unicast addresses (max 3):
        fe80::3265:842a:4b:1200 autoconf preferred infinite
        2001:db8::1 manual preferred infinite
IPv6 multicast addresses (max 4):
        ff02::1
        ff02::1:ff4b:1200
        ff02::1:ff00:1
debian@BeaglePlay:~$ tio /dev/ttyACM0
[tio 07:32:17] tio v1.32
[tio 07:32:17] Press ctrl-t q to quit
[tio 07:32:17] Connected
gd25q16c@0: SFDP v 1.0 AP ff with 2 PH
I: PH0: ff00 rev 1.0: 9 DW @ 30
I: gd25q16c@0: 2 MiBy flash
I: PH1: ffc8 rev 1.0: 3 DW @ 60
*** Booting Zephyr OS build zephyr-v3.2.0-3470-g14e193081b1f ***
I: Starting bootloader
I: Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Boot source: primary slot
I: Swap type: test
I: Bootloader chainload address offset: 0x20000
I: Jumping to the first image slot


[00:00:00.001,647] <inf> spi_nor: gd25q16c@0: SFDP v 1.0 AP ff with 2 PH
[00:00:00.001,647] <inf> spi_nor: PH0: ff00 rev 1.0: 9 DW @ 30
[00:00:00.001,983] <in
>>>
```

Press reset

```
I: gd25q16c@0: SFDP v 1.0 AP ff with 2 PH
I: PH0: ff00 rev 1.0: 9 DW @ 30
I: gd25q16c@0: 2 MiBy flash
I: PH1: ffc8 rev 1.0: 3 DW @ 60
*** Booting Zephyr OS build zephyr-v3.2.0-3470-g14e193081b1f ***
I: Starting bootloader
I: Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Boot source: primary slot
I: Swap type: test
I: Bootloader chainload address offset: 0x20000
I: Jumping to the first image slot


[00:00:00.001,495] <inf> spi_nor: gd25q16c@0: SFDP v 1.0 AP ff with 2 PH
[00:00:00.001,525] <inf> spi_nor: PH0: ff00 rev 1.0: 9 DW @ 30
[00:00:00.001,800] <inf> spi_nor: gd25q16c@0: 2 MiBy flash
[00:00:00.001,831] <inf> spi_nor: PH1: ffc8 rev 1.0: 3 DW @ 60
uart:~$ build time: Feb 22 2023 07:13:09MicroPython v1.19.1 on 2023-02-22;␣
↪zephyr-beagleconnect_freedom with unknown-cpu
Type "help()" for more information.
>>> help()
Welcome to MicroPython!

Control commands:
  CTRL-A        -- on a blank line, enter raw REPL mode
  CTRL-B        -- on a blank line, enter normal REPL mode
  CTRL-C        -- interrupt a running program
  CTRL-D        -- on a blank line, do a soft reset of the board
  CTRL-E        -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)

See https://beagleconnect.org/micropython for examples.
>>> import zsensor
>>> light=zsensor.Sensor("OPT3001-LIGHT")
>>> humidity=zsensor.Sensor("HDC2010-HUMIDITY")
>>> light.measure()
>>> light.get_float(zsensor.LIGHT)
35.94
>>> humidity.measure()
>>> humidity.get_float(zsensor.HUMIDITY)
24.32861
>>> humidity.get_float(zsensor.AMBIENT_TEMP)
22.37704
>>> dir(zsensor)
['__name__', 'ACCEL_X', 'ACCEL_Y', 'ACCEL_Z', 'ALTITUDE', 'AMBIENT_TEMP',
↪'BLUE', 'CO2', 'DIE_TEMP', 'DISTANCE', 'GAS_RES', 'GREEN', 'GYRO_X', 'GYRO_
↪Y', 'GYRO_Z', 'HUMIDITY', 'IR', 'LIGHT', 'MAGN_X', 'MAGN_Y', 'MAGN_Z', 'PM_
↪10', 'PM_1_0', 'PM_2_5', 'PRESS', 'PROX', 'RED', 'Sensor', 'VOC', 'VOLTAGE
↪']
>>> import os
>>> with open('/flash/test.txt', 'w') as f:
...     f.write("My test.txt\n")
...     ^H
12
>>> print(open('/flash/test.txt').read())
My test.txt

>>> import socket
>>> sock = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
>>> sock.bind(('ff02::1', 9999))
```

(continues on next page)

(continued from previous page)

```
>>> for i in range(3):
...     data, sender = sock.recvfrom(1024)
...     print(str(sender) + '  ' + repr(data))
...     ^H
('fe80::ec0f:7a22:4b:1200', <>, 0, 7)  b'4h:32.71;4t:17.29;'
('fe80::ec0f:7a22:4b:1200', <>, 0, 7)  b'2l:0.35;'
('fe80::ec0f:7a22:4b:1200', <>, 0, 7)  b'4h:32.71;4t:17.29;'
>>> import machine
>>> AN=machine.Pin(("GPIO_0", 23), machine.Pin.OUT)
>>> AN.init(machine.Pin.OUT, machine.Pin.PULL_UP, value=1)
>>> LNK_LED=machine.Pin(("GPIO_0", 18), machine.Pin.OUT)
>>> LNK_LED.init(machine.Pin.OUT, machine.Pin.PULL_UP, value=1)
>>> LNK_LED.off()
>>> LNK_LED.on()
>>>
^Tq
[tio 07:40:16] Disconnected
debian@BeaglePlay:~$
```

0.2.2

Todo: Need to describe functionality of 0.2.2

Updating

Look for the latest firmware release on https://www.beagleboard.org/distros or on https://beagleconnect.org.

Download, unzip and flash the `micropython-w-boot` image.

```
wget https://files.beagle.cc/file/beagleboard-public-2021/images/zephyr-
↪beagle-cc1352-0.2.2.zip
unzip zephyr-beagle-cc1352-0.2.2.zip
./build/freedom/cc2538-bsl.py build/freedom/micropython-w-boot
```

Contributing

Repository: https://git.beagleboard.org/beagleconnect/zephyr/micropython

12.5.2 Using Zephyr

Developing directly in Zephyr will not be ultimately required for end-users who won't touch the firmware running on BeagleConnect™ Freedom and will instead use the BeagleConnect™ Greybus functionality, but is important for early adopters as well as people looking to extend the functionality of the open source design. If you are one of those people, this is a good place to get started.

Equipment to begin development

There are many options, but using BeaglePlay gives a reasonable common environment. Please adjust as you see fit.

Required

- BeaglePlay with provided antennas
- BeagleConnect Freedom with provided USB cable

- 2x 5V/3A USB power adapters
- USB Type-C cable for use with BeaglePlay

Recommended

- Ethernet cable and Internet connection

Install the SDK on BeaglePlay

See *Setup Zephyr development on BeaglePlay*.

Important: TODO: note the tested version of software for BeaglePlay

Important: TODO: describe how to know it is working

Change default board The instructions linked above setup the environment for targeting BeaglePlay's on CC1352. We need to change it to target BeagleConnec Freedom.

```
echo "export BOARD=beagleconnect_freedom" >> $HOME/zephyr-beagle-
↪cc1352-sdk/zephyr-beagle-cc1352-env/bin/activate
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
```

Try demo applications

Now you can build various Zephyr applications

Build and flash Blinky Make sure your BeagleConnect Freedom is connected to your BeaglePlay via the USB cable provided.

```
cd $ZEPHYR_BASE
west build zephyr/samples/basic/blinky
west flash
```

Debug applications over the serial terminal

Note: #TODO#

12.5.3 Using BeagleConnect Greybus

Note: This is still in development.

BeagleConnect wireless user experience



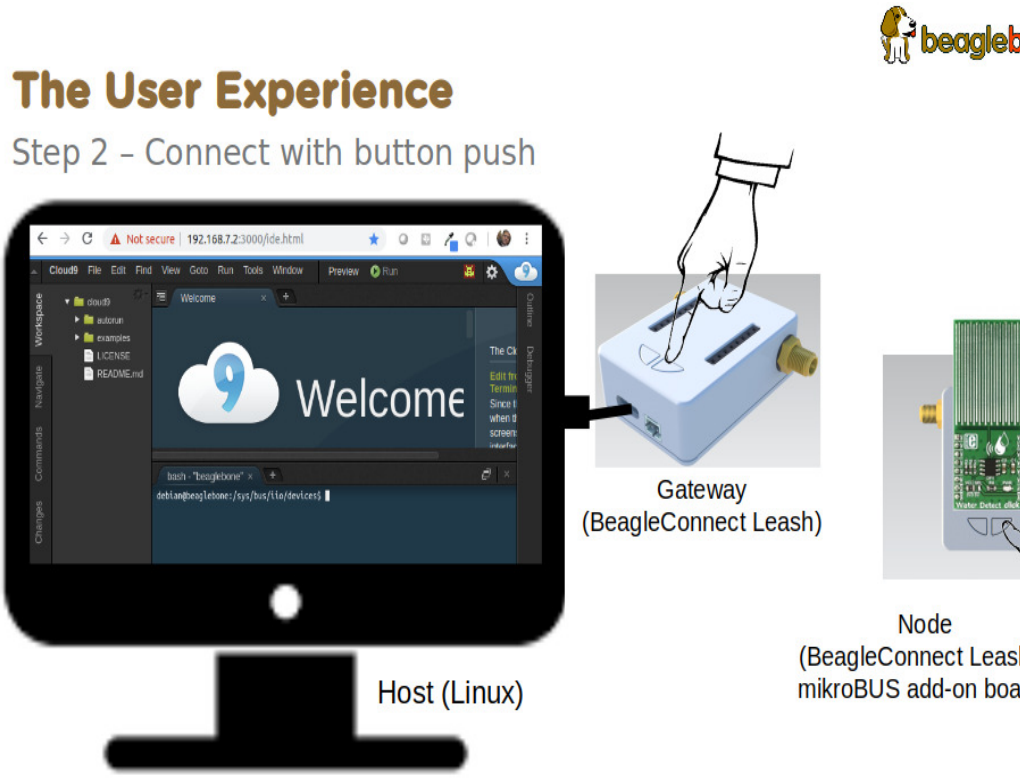
Enable a Linux host with BeagleConnect

Log into a host system running Linux that is BeagleConnect™ enabled. Enable a Linux host with BeagleConnect™ by plugging a **BeagleConnect™ gateway device** into its USB port. You'll also want to have a **BeagleConnect™ node device** with a sensor, actuator or indicator device connected.

Note: BeagleConnect™ Freedom can act as either a BeagleConnect™ gateway device or a BeagleConnect™ node device.

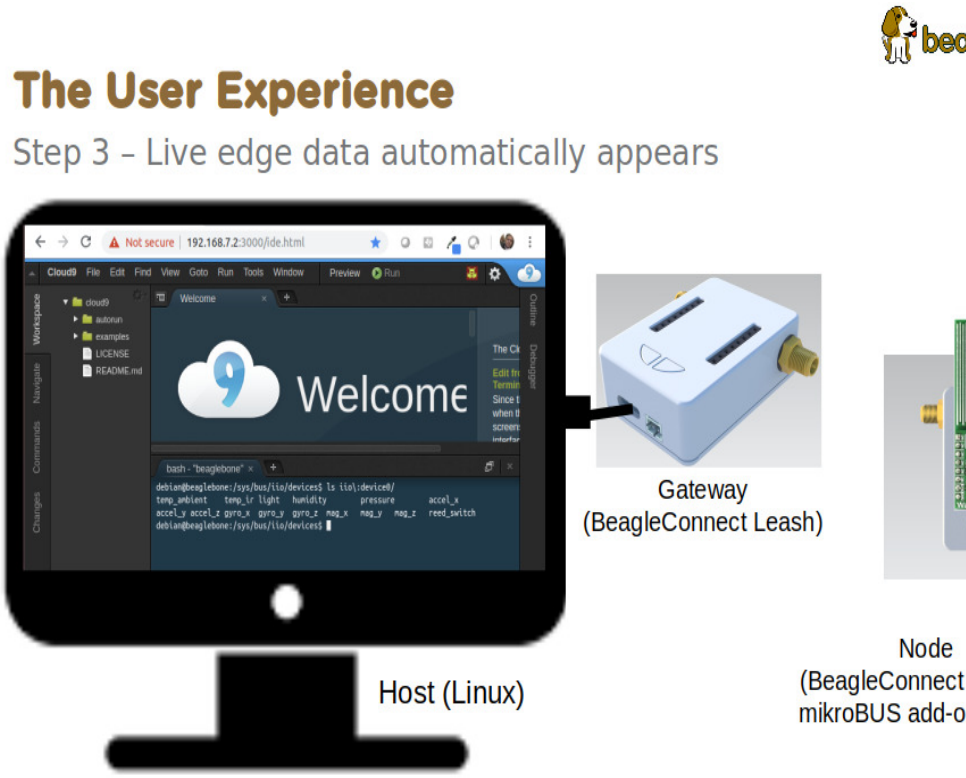
Important: The Linux host will need to run the BeagleConnect™ management software, most of which is incorporated into the Linux kernel. Support will be provided for BeagleBoard and BeagleBone boards, x86 hosts, and Raspberry Pi.

#TODO#: Clean up images



Connect host and device

Initiate a connection between the host and devices by pressing the discovery button(s).



Device data shows up as files

New streams of self-describing data show up on the host system using native device drivers.

High-level applications, like Node-RED, can directly read/write these high-level data streams (including data-type information) to Internet-based MQTT brokers, live dashboards, or other logical operations without requiring

any sensor-specific coding. Business logic can be applied using simple if-this-then-that style operations or be made as complex as desired using virtually any programming language or environment.

Components

BeagleConnect™ enabled host Linux computer, possibly single-board computer (SBC), with BeagleConnect™ management software and BeagleConnect™ gateway function. BeagleConnect™ gateway function can be provided by a BeagleConnect™ compatible interface or by connecting a BeagleConnect™ **gateway** device over USB.

Note: If the Linux host has BLE, the BeagleConnect™ **gateway** is optional for short distances

---

BeagleConnect™ Freedom Board, case, and wireless MCU with Zephyr based firmware for acting as either a BeagleConnect™ gateway device or BeagleConnect™ node device.

- In BeagleConnect™ **gateway** device mode: Provides long-range, low-power wireless communications, Connects with the host via USB and an associated Linux kernel driver, and is powered by the USB connector.
- In BeagleConnect™ **node** device mode: Powered by a battery or USB connector Provides 2 mikroBUS connectors for connecting any of hundreds of Click Board mikroBUS add-on devices Provides new Linux host controllers for SPI, I2C, UART, PWM, ADC, and GPIO with interrupts via Greybus

BeagleConnect gateway device Provides a BeagleConnect™ compatible interface to a host. This could be a built-in interface device or one connected over USB. BeagleConnect™ Freedom can provide this function.

BeagleConnect node device Utilizes a BeagleConnect™ compatible interface and TODO

BeagleConnect compatible interface Immediate plans are to support Bluetooth Low Energy (BLE), 2.4GHz IEEE 802.15.4, and Sub-GHz IEEE 802.15.4 wireless interfaces. A built-in BLE interface is suitable for this at short range, whereas IEEE 802.15.4 is typically significantly better at long ranges. Other wired interfaces, such as CAN and RS-485, are being considered for future BeagleConnect™ gateway device and BeagleConnect™ node device designs.

Greybus TODO

#TODO: Find a place for the following notes:

- The device interfaces get exposed to the host via Greybus BRIDGED_PHY protocol
- The I2C bus is probed for a an identifier EEPROM and appropriate device drivers are loaded on the host
- Unsupported Click Boards connected are exposed via userspace drivers on the host for development

What's different?

So, in summary, what is so different with this approach?

- No microcontroller code development is required by users
- Userspace drivers make rapid prototyping really easy
- Kernel drivers makes the support code collaborative parts of the Linux kernel, rather than cut-and-paste

12.6 Support

12.6.1 Certifications and export control

Export designations

- HS: 8471504090
- US HS: 8473301180
- EU HS: 8471707000

Size and weight

- Bare product dimensions (without antenna): 63 x 56 x 16.6 mm
- Bare product weight (with antenna): 53.2 g
- Full package dimensions: 188 x 85 x 35 mm
- Full package weight: 95.2 g

12.6.2 Additional documentation

Hardware docs

For any hardware document like schematic diagram PDF, EDA files, issue tracker, and more you can checkout the BeagleConnect Freedom repository.

Software docs

For BeagleConnect Freedom specific software projects you can checkout all the BeagleConnect project repositories group.

Support forum

For any additional support you can submit your queries on our forum, https://forum.beagleboard.org/tag/bcf

Pictures

12.6.3 Change History

Note: This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

12.6.4 Document Changes

For all changes, see https://git.beagleboard.org/docs/docs.beagleboard.io. Frozen releases tested against specific hardware and software revisions are noted below.

Rev	Changes	Date	By

Board Changes

For all changes, see https://git.beagleboard.org/beagleconnect/freedom. Versions released into production are noted below.

Table 12.1: BeagleConnect Freedom board change history

Rev	Changes	Date	By
C7	Initial production version	2023-03-08	JK

Chapter 13

BeagleBoard (all)

BeagleBoard.org single board computers (SBCs) and microcontroller development boards are fully open-source, low-cost, RISC-V & ARM based boards. Which makes them suitable for students to learn about embedded electronics & embedded Linux. Beagle development platforms also enable rapid prototyping for professionals to develop industrial & production systems.

Note: Make sure to read and accept all the terms & condition provided in the *Terms & Conditions* page.

Use of either the boards or the design materials constitutes agreement to the T&C including any modifications done to the hardware or software solutions provided by beagleboard.org foundation.

The latest unified docs PDF is linked below.

- BeagleBoard.org unified docs PDF

The latest System Reference Manual (PDF) for older BeagleBoard boards are linked below.

- BeagleBoard
- BeagleBoard-xM
- BeagleBoard-X15

Chapter 14

Projects

This is a collection of reasonably well-supported projects useful to Beagle developers.

14.1 simpPRU

14.1.1 simpPRU Basics

The PRU is a dual core micro-controller system present on the AM335x SoC which powers the BeagleBone. It is meant to be used for high speed jitter free IO control. Being independent from the linux scheduler and having direct access to the IO pins of the BeagleBone Black, the PRU is ideal for offloading IO intensive tasks.

Programming the PRU is a uphill task for a beginner, since it involves several steps, writing the firmware for the PRU, writing a loader program. This can be a easy task for a experienced developer, but it keeps many creative developers away. So, I propose to implement a easy to understand language for the PRU, hiding away all the low level stuff and providing a clean interface to program PRU.

This can be achieved by implementing a language on top of PRU C. It will directly compile down to PRU C. This could also be solved by implementing a bytecode engine on the PRU, but this will result in waste of already limited resources on PRU. With this approach, both PRU cores can be run independent of each other.



simpPRU

Intuitive language for PRU which compiles down to PRU C.

What is simpPRU

- simpPRU is a procedural programming language.
- It is a statically typed language. Variables and functions must be assigned data types during compilation.
- It is type-safe, and data types of variables are decided during compilation.
- simpPRU codes have a `.sim` extension.

- simpPRU provides a console app to use Remoteproc functionality.

14.1.2 Build from source

Dependencies

- flex
- bison
- gcc
- gcc-pru
- gnuprumcu
- cmake

Build

```
git clone https://github.com/VedantParanjape/simpPRU.git
cd simpPRU
mkdir build
cd build
cmake ..
make
```

Install

```
sudo make install
```

Generate debian package

```
sudo make package
```

14.1.3 Install

Dependencies

- gcc-pru
- gnuprumcu
- config-pin utility (for autoconfig)

Installation

For Instructions head over to Installation

Requirements

Currently this only supports am335x systems: PocketBeagle, BeagleBone Black and BeagleBone Black Wireless:

- `gcc-pru`
- `gnuprumcu`
- beaglebone image with official support for remoteproc: `ti-4.19+ kernel`
- `config-pin` utility

Build from source

For Instructions head over to Building from source

```
simppru-console
```

For detailed usage head to Detailed Usage

amd64

```
wget https://github.com/VedantParanjape/simpPRU/releases/download/1.4/
↪simppru-1.4-amd64.deb

sudo dpkg -i simppru-1.4-amd64.deb
```

armhf

```
wget https://github.com/VedantParanjape/simpPRU/releases/download/1.4/
↪simppru-1.4-armhf.deb

sudo dpkg -i simppru-1.4-armhf.deb
```

Issues

- For full source code of simPRU visit
- To report a bug or start a issue visit

14.1.4 Language Syntax

- simpPRU is a procedural programming language.
- It is a statically typed language. Variables and functions must be assigned data types during compilation.
- It is type-safe, and data types of variables are decided during compilation.
- simPRU codes have a `.sim` extension.

Datatypes

- `int` - Integer datatype
- `bool` - Boolean datatype
- `char` / `uint8` - Character / Unsigned 8 bit integer datatype
- `void` - Void datatype, can only be used a return type for functions

Constants

- `<any_integer>` - Integer constant. Integers can be decimal, hexadecimal (start with 0x or 0X) or octal (start with 0)
- `'<any character>'` - Character constant. These can be assigned to both int and char/uint8 variables
- `true` - Boolean constant (True)
- `false` - Boolean constant (False)
- `Px_yz` - Pin mapping constants are Integer constant, where x is 1,2 or 8,9 and yz are the header pin numbers.

Operators

- `{,}` - Braces
- `(,)` - Parenthesis
- `/,*,+,-,%` - Arithmetic operators
- `>,<,==,!=,>=,<=` - Comparison operators
- `~,&,|,<<,>>` - Bitwise operators: not, and, or and bitshifts
- `not,and,or` - Logical operators: not, and, or
- `:=` - Assignment operator
- Result of Arithmetic and Bitwise operators is Integer constant.
- Result of Comparison and Logical operators is Boolean constant.
- Characters are treated as integers when used in Arithmetic expressions.
- Only Integer constants can be used with Arithmetic and Bitwise operators.
- Only Integer constants can be used with Comparison operators.
- Only Boolean constants can be used with Logical operators.
- Operators are evaluated following these precedence rules.

```
Correct: bool out := 5 > 6;
Wrong:   int yy := 5 > 6;
```

Variable declaration

- Datatype of variable needs to be specified during compile time.
- Variables can be assigned values after declarations.
- If variable is not assigned a value after declaration, it is set to `0` for `integer` and `char/uint8` and to `false` for `boolean` by default.
- Variables can be assigned other variables of same datatype. `int`s and `char`s can be assigned to each other.
- Variables can be assigned expressions whose output is of same datatype.

Declaration

```
int var;
char char_var;
bool test_var;
```

Assignment during Declaration

```
int var := 99;
char char_var := 'a';
uint8 short_var := 255;
bool test_var := false;
```

Assignment

```
var := 45;
short_var := var;
test_var := true;
```

- Variables to be assigned must be declared earlier.
- Datatype of the variables cannot change. Only appropriate expressions/constants of their respective datatypes can be assigned to the variables.
- Integer and Character variable can be assigned only Integer expression/Integer constant/Character constant.
- Boolean variable can be assigned only Boolean expression/constant.

Arrays

- Arrays are static - their size has to be known at compile time and this size cannot be changed later.
- Arrays can be used with bool, int and char.
- Arrays do not support any arithmetic / logical / comparison / bitwise operators, however these operators work fine on their elements.

Declaration and Assignment

- The data type has to be specified as data_type[size].
- Array of char can be initialized from a double quoted string, where the length of the array would be at least the length of the string plus 1.

```
int[16] a; /* array of 16 integers */
char[20] string1 := "I love BeagleBoards";
```

Indexing:

- Arrays are zero-indexed.
- The index can be either a char or an int or an expression involving chars and ints.
- Accessing elements of an array:

```
int a := arr[4]; /* Copy the 5th element of arr to a */
```

- Changing elements of an array:

```
arr[4] := 5; /* The 5th element of arr is now 5 */

int i := 4;
arr[i] := 6; /* The 5th element of arr is now 6 */

char j := 4;
arr[j] := 7; /* The 5th element of arr is now 7 */
```

(continues on next page)

(continued from previous page)

```
arr[i+j] := 1; /* The 9th element of arr is now 1 */

/* Declaring and initializing an array with all zeros */
int[16] arr;
for: i in 0:16 {
    arr[i] := 0;
}
```

Comments

- simpPRU supports C style multiline comments.

```
/* This is a comment */

/* Comments can span
multiple lines */
```

Keyword and Identifiers

Table 14.1: Reserved keywords

“true“	**“read_counter“**	**“stop_counter“**
“false“	**“start_counter“**	**“pwm“**
“int“	**“delay“**	**“digital_write“**
“bool“	**“digital_read“**	**“def“**
“void“	**“return“**	**“or“**
“if“	**“and“**	**“not“**
“elif“	**“continue“**	**“break“**
“else“	**“while“**	**“in“**
“for“	**“init_message_channel“**	**“send_message“**
“receive_message“	**“print“**	**“println“**

Valid identifier naming

- An identifier/variable name must be start with an alphabet or underscore (_) only, no other special characters, digits are allowed as first character of the identifier/variable name.

  `product_name, age, _gender`

- Any space cannot be used between two words of an identifier/variable; you can use underscore (_) instead of space.

  `product_name, my_age, gross_salary`

- An identifier/variable may contain only characters, digits and underscores only. No other special characters are allowed, and we cannot use digit as first character of an identifier/variable name (as written in the first point).

  `length1, length2, _City_1`

Detailed info: https://www.includehelp.com/c/identifier-variable-naming-conventions.aspx

Expressions

Arithmetic expressions

```
=> (9 + 8) * 2 + -1;
33
=> 11 % 3;
2
```

(continues on next page)

(continued from previous page)

```
=> 2 * 6 << 2 + 1;
96
=> ~0xFFFFFFFF;
0
```

Boolean expressions

```
=> 9 > 2 or 8 != 2 and not( 2 >= 5 or 9 <= 5 ) or 9 != 7;
true
=> 0xFFFFFFFF != 0XFFFFFFFF;
false
=> 'a' < 'b';
true
```

- **Note** : Expressions are evaluated following the *operator precedence <#operators>*

If-else statement

Statements in the if-block are executed only if the if-expression evaluates to `true`. If the value of expression is `true`, statement1 and any other statements in the block are executed and the else-block, if present, is skipped. If the value of expression is `false`, then the if-block is skipped and the else-block, if present, is executed. If elif-block are present, they are evaluated, if they become `true`, the statement is executed, otherwise, it goes on to eval next set of statements

Syntax

```
if : boolean_expression {
    statement 1
    ...
    ...
}
elif : boolean_expression {
    statement 2
    ...
    ...
    ...
}
else {
    statement 3
    ...
    ...
}
```

Examples

```
int a := 3;

if : a != 4 {
    a := 4;
}
elif : a > 4 {
    a := 10;
}
else {
    a := 0;
}
```

- This will evaluate as follows, since `a = 3`, if-block (`3!=4`) will evaluate to true, and value of a will be set to 4, and program execution will stop.

For-loop statement

For loop is a range based for loop. Range variable is a local variable with scope only inside the for loop.

Syntax

```
for : var in start:stop {
    statement 1
    ....
    ....
}
```

- Here, for loop is a range based loop, value of integer variable `var` will vary from `start` to `stop - 1`. Value of `var` does not equal `stop`. Here, `increment` is assumed to be 1, so `start` will have to less than `stop`.
- Optionally, `start` can be skipped, and it will automatically start from 0, like this:

```
for : var in :stop {
    statement 1
    ....
    ....
}
```

- Optionally, `increment` can also be specified like this. Here, `stop` can be less than `start` if `increment` is negative.

```
for : var in start:stop:increment {
    statement 1
    ....
    ....
}
```

- **Note** : **var** is a **integer**, and **start, stop, increment** can be **arithmetic expression, integer or character variable, or integer or character constant**.

Examples

```
int sum := 0;

for : i in 1:4 {
    sum = sum + i;
}
```

```
int mx := 32;
int nt;

for : j in 2:mx-10 {
    nt := nt + j;
}
```

```
int sum := 0;

for : i in in 10:1:-2 { /*10, 8, 6, 4, 2*/
    sum = sum + i;
}
```

While-loop statement

While loop statement repeatedly executes a target statement as long as a given condition is true.

Syntax

```
while : boolean_expression {
    statement 1
    ...
    ...
}
```

Examples

- Infinite loop

```
while true {
    do_something..
    ...
}
```

- Normal loop, will repeat 30 times, before exiting

```
int tag := 0;

while : tag < 30 {
    tag := tag + 1;
}
```

Control statements

- **Note** : `break` and `continue` can only be used inside looping statements

break `break` is used to break execution in a loop statement, either `for loop` or `while loop`. It exits the loop upon calling.

Syntax `break;`

Examples

```
for : i in 0:9 {
    if : i == 3 {
        break;
    }
}
```

continue `continue` is used to continue execution in a loop statement, either `for loop` or `while loop`.

Syntax `continue;`

Examples

```
for : j in 9:19 {
    if : i == 12 {
        continue;
    }
    else {
        break;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Functions

Function definition A function is a group of statements that together perform a task. You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task. A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

- **Warning** : Function must be defined before calling it.

Syntax

```
def <function_name> : <data_type> : <data_type> <param_name>, <data_type>
↪<param_name>, ... {
    statement 1;
    ...
    ...

    return <data_type>;
}
```

Note: If return data type is void, then return statement is not needed, and if still it is added, it must be return nothing, i.e., something like this `return ;`

Warning: `return` can only be present in the body of the function only once, that too at the end of the function, not inside any compound statements.

Danger: `return` inside a compound statement, this syntax is not allowed.

```
def test : int : int a {
if : a < 4 {
return a;
}
}
```

- **Correct** : `return` is not inside compound statements, It should be placed only at the end of function definition

```
def test : int : int a {
int gf := 8;
if : a < 4
{
gf := 4;
}
return gf;
}
```

Examples Examples according to return types

- **Integer**

```
def test_func : int : int a, int b
{
   int aa := a + 5;
   if : aa < 3 {
        aa : = 0;
   }

   return aa + b;
}
```

- **Character**

```
def next_char : char : char ch, int inc {
   char chinc := ch + inc;
   return chinc;
}
```

- **Boolean**

```
def compare : bool : int val {
   bool ret :=false;
      if : val < 0 {
         ret := true;
      }
      return ret;
   }
```

- **Void**

```
def example_func_v : void : {
    int temp := 90;

    return;
}
```

Function call Functions can be called only if, they have been defined earlier. They return data types according to their definition. Parameters are passed by value. Only pass by value is supported as of now.

Syntax

```
function_name(var1, var2, ..);
```

Examples

- **Integer** `int a := 55; int ret_val := test_func(4, a);`
- **Character** `char a := 'a'; char b := next_char(a, 1);`
- **Boolean** `bool val := compare(22); compare(-2);`
- **Void** `example_func(false); example_func_v();`

Testing or Debugging For testing or debugging code, use the –test or -t flag to enable print, println and stub functions. Use –preprocess to stop after generating the C code only. Then run the generated C code (at /tmp/temp.c) using `gcc`.

Print functions print can take either a string (double quoted) or any `int` / `char` / `bool` identifier.

println is similar to print but also prints a newline (`\n`).

Examples

```
print("Hello World!");
int a := 2;
print(a);
a := a + 2;
print(a);
println("");
```

Stub functions PRU specific functions will be replaced by stub functions which print **function_name called with arguments arg_name** when called.

14.1.5 IO Functions

- All Header pins are `constant integer variable` by default, with its value equal to respective R30/R31 register bit
  - Example: `P1_20` is an constant integer variable with value `16`, similarly `P1_02` is an constant integer variable with value `9`

Digital Write

`digital_write` is a function which enables PRU to write given logic level at specified output pin. It is a function with void return type and it's parameters are `integer` and `boolean`, first parameter is the pin number to write to or PRU R30 register bit and second parameter is `boolean` value to be written. `true` for HIGH and `false` for LOW.

Syntax `digital_write(pin_number, value);`

Parameters

- `pin_number` is an integer. It must be a header pin name which supports output, or PRU R30 Register bit.
- `value` is a boolean. It is used to set logic level of the output pin, `true` for HIGH and `false` for LOW.

Return Type

- `void` - returns nothing.

Example

```
int a := 32;

if : a < 32 {
    digital_write(P1_29, true);
}
else {
    digital_write(P1_29, false);
}
```

If the value of a < 32, then pin `P1_29` is set to `HIGH` or else it is set to `LOW`.

Digital Read

`digital_read` is a function which enables PRU to read logic level at specified input pin. It is a function with return type `boolean` and it's parameter is a `integer` whose value must be the pin number to be read or PRU R31 register bit.

Syntax `digital_read(pin_number);`

Parameters

- `pin_number` is an integer. It must be a header pin name which supports input, or PRU R31 Register bit.

Return Type

- `boolean` - returns the logic level of the pin number passed to it. It returns `true` for HIGH and `false` for LOW.

Example

```
if digital_read(P1_20) {
    digital_write(P1_29, false);
}
else {
    digital_write(P1_29, true);
}
```

Logic level of pin `P1_20` is read. If it is HIGH, then pin `P1_29` is set to `LOW`, or else it is set to `HIGH`.

Delay

`delay` is a function which makes PRU wait for specified milliseconds. When this is called PRU does absolutely nothing, it just sits there waiting.

Syntax `delay(time_in_ms);`

Parameters

- `time_in_ms` is an integer. It is the amount of time PRU should wait in milliseconds. (1000 milliseconds = 1 second).

Return Type

- `void` - returns nothing.

Example

```
digital_write(P1_29, true);
delay(2000);
digital_write(P1_29, false);
```

Logic level of pin `P1_29` is set to `HIGH`, PRU waits for *2000 ms = 2 seconds*, and then sets the logic level of pin `P1_29` to `LOW`.

Start counter

`start_counter` is a function which starts PRU's internal counter. It counts number of CPU cycles. So it can be used to count time elapsed, as it is known that each cycle takes 5 nanoseconds.

Syntax `start_counter()`

Parameters

- n/a

Return Type

- `void` - returns nothing.

Example

```
start_counter();
```

Stop counter

`stop_counter` is a function which stops PRU's internal counter.

Syntax `stop_counter()`

Parameters

- n/a

Return Type

- `void` - returns nothing.

Example

```
stop_counter();
```

Read counter

`read_counter` is a function which reads PRU's internal counter and returns the value. It counts number of CPU cycles. So it can be used to count time elapsed, as it is known that each cycle takes 5 nanoseconds.

Syntax `read_counter()`

Parameters

- n/a

Return Type

- `integer` - returns the number of cycles elapsed since calling `start_counter`.

Example

```
start_counter();

while : read_counter < 200000000 {
    digital_write(P1_29, true);
}

digital_write(P1_29, false);
stop_counter();
```

while the value of hardware counter is less than 200000000, it will set logic level of pin `P1_29` to `HIGH`, after that it will set it to `LOW`. Here, 200000000 cpu cycles means 1 second of time, as CPU clock is 200 MHz. So, LED will turn on for 1 second, and turn off after.

Init message channel

`init_message_channel` is a function which is used to initialise communication channel between PRU and the ARM core. It is sets up necessary structures to use RPMSG to communicate, it expects a init message from the ARM core to initialise. It is a necessary to call this function before using any of the message functions.

Syntax `init_message_channel()`

Parameters

- n/a

Return Type

- `void` - returns nothing

Example

```
init_message_channel();
```

Receive message

`receive_message` is a function which is used to receive messages from ARM to the PRU, messages can only be `integers`, as only they are supported as of now. It uses RPMSG channel setup by `init_message_channel` to receive messages from ARM core.

Syntax `receive_message()`

Parameters

- n/a

Return Type

- `integer` - returns integer data received from PRU

Example

```
init_message_channel();

int temp := receive_message();

if : temp >= 0 {
    digital_write(P1_29, true);
}
else {
    digital_write(P1_29, false);
}
```

Send message

There are six functions which are used to send messages to ARM core from PRU, messages can be `integers`, `characters`, `bools`, `integer arrays`, `character arrays`, and `boolean arrays`. It uses RPMSG channel setup by `init_message_channel` to send messages from PRU to the ARM core.

For sending arrays, arrays are automatically converted to a string, for example, [1, 2, 3, 4] would become "1 2 3 4".

Syntax

- `send_int(expression)`
- `send_char(expression)`
- `send_bool(expression)`
- `send_ints(identifier)`
- `send_chars(identifier)`
- `send_bools(identifier)`
- `send_message` is an alias for `send_int` to preserve backwards compatibility.

Parameters

- For `send_int` and `send_char`, `expression` would be an arithmetic expression.
- For `send_bool`, `expression` would be a boolean expression
- For `send_ints`, `identifier` should be an identifier for an integer array.
- For `send_chars`, `identifier` should be an identifier for a character array.
- For `send_bools`, `identifier` should be an identifier for a boolean array.

Example

```
init_message_channel();

if : digital_read(P1_29) {
    send_bool(true);
}
else {
    send_int(0);
}
```

14.1.6 Usage(simppru)

```
simppru [OPTION...] FILE

      --device=<device_name> Select for which BeagleBoard to compile
                             (pocketbeagle, bbb, bbbwireless, bbai)
      --load                 Load generated firmware to /lib/firmware/
  -o, --output=<file>        Place the output into <file>
  -p, --pru=<pru_id>         Select which pru id (0/1) for which program is␣
↪to
                             be compiled
      --verbose              Enable verbose mode (dump symbol table and ast
                             graph)
      --preprocess           Stop after generating the intermediate C
                             file (located at /tmp/temp.c)
  -t  --test                 Use stub functions for PRU specific functions␣
↪and
                             enable the print functions, useful for testing␣
↪and debugging
  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or␣
↪optional
for any corresponding short options.
```

simppru autodetects BeagleBoard model and automatically configures pin mux using config-pin. This functionality doesn't work on BeagleBone Blue and AI.

Say we have to compile a example file called `test.sim`, command will be as follows:

```
simppru test.sim --load
```

If we only want to generate binary for pru0

```
simppru test.sim -o test_firmware -p 0
```

this will generate a file named `test_firmware.pru0`

14.1.7 Usage(simppru-console)

simppru-console is a console app, it can be used to send/receive message to the PRU using RPMSG, and also start/stop the PRU. It is built to facilitate easier way to use rpmsg and remoteproc API's to control and communicate with the PRU

- **Warning** : Make sure to stop PRU before exiting. Press `ctrl+c` to exit



Features

Use arrow keys to navigate around the textbox and buttons.

Start/stop buttons Use these button to start/stop the selected PRU. If PRU is already running, on starting simppru-console, it is automatically stopped.



Send message to PRU Use this text box to send data to the PRU, only *Integers* are supported. On pressing enter, the typed message is sent.

PRU0 is running echo program, whatever is sent is echoed back.



Receive message from PRU The large box in the screen shows data received from the PRU, It runs using a for loop, which checks if new message is arrived every 10 ms.

- PRU is running echo program, whatever is sent is echoed back.



- PRU is running countup program, it sends a increasing count every 1 second, which starts from 0



Change PRU ID Using the radio box in the upper right corner, one can change the PRU id, i.e. if one wants to use the features for PRU0 or PRU1

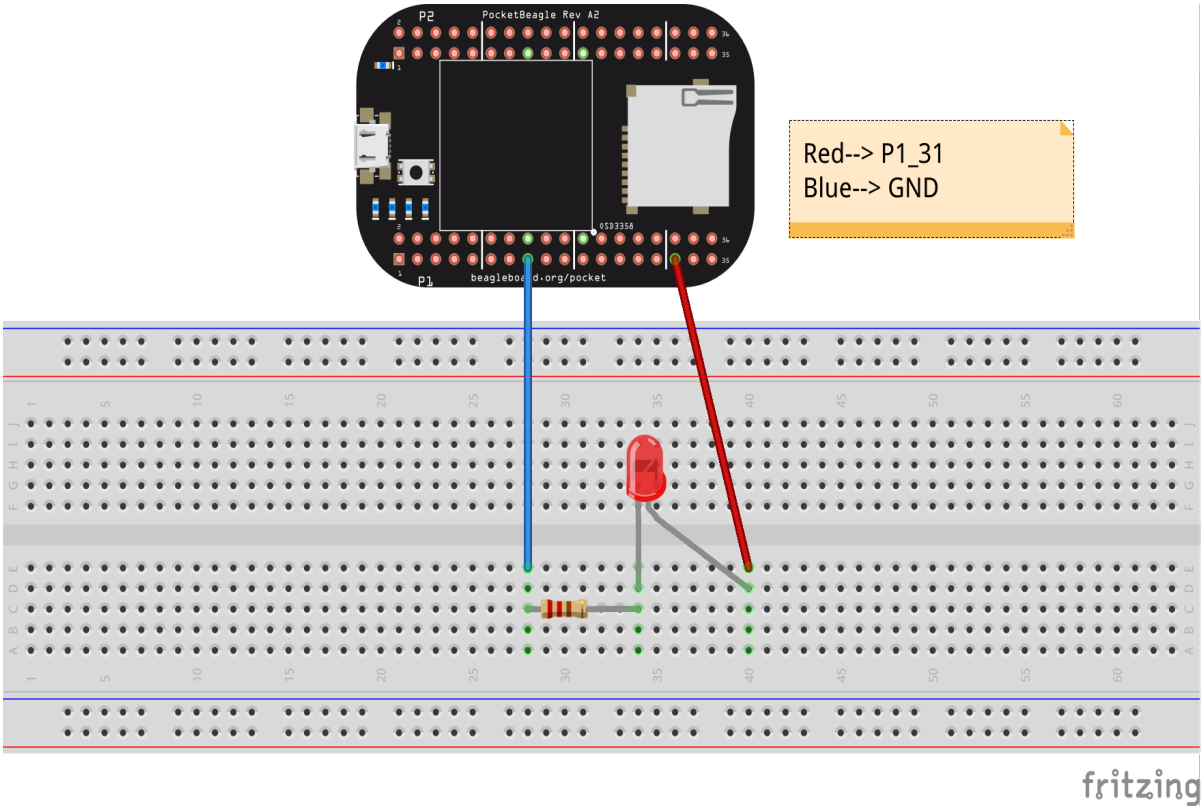


14.1.8 simpPRU Examples

These are the examples which have been tested on simpPRU.These examples will serve as a guide for the users to implement.



Delay example



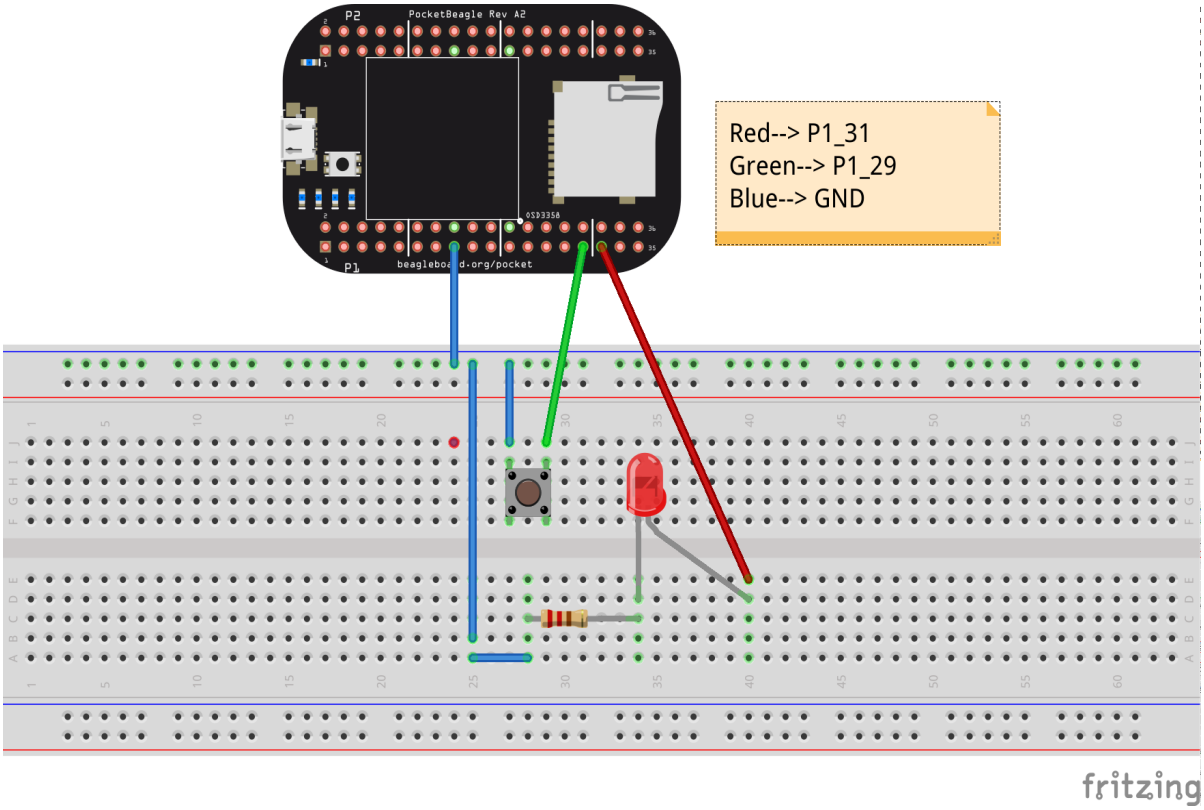
Code

```
digital_write(P1_31, true);
delay(2000);
digital_write(P1_31, false);
delay(5000);
digital_write(P1_31, true);
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation This code snippet writes HIGH to header pin P1_31, then waits for 2000ms using the `delay` call, after that it writes LOW to header pin P1_31, then again waits for 5000ms using the `delay` call, and finally writes HIGH to header pin P1_31.

Digital read example



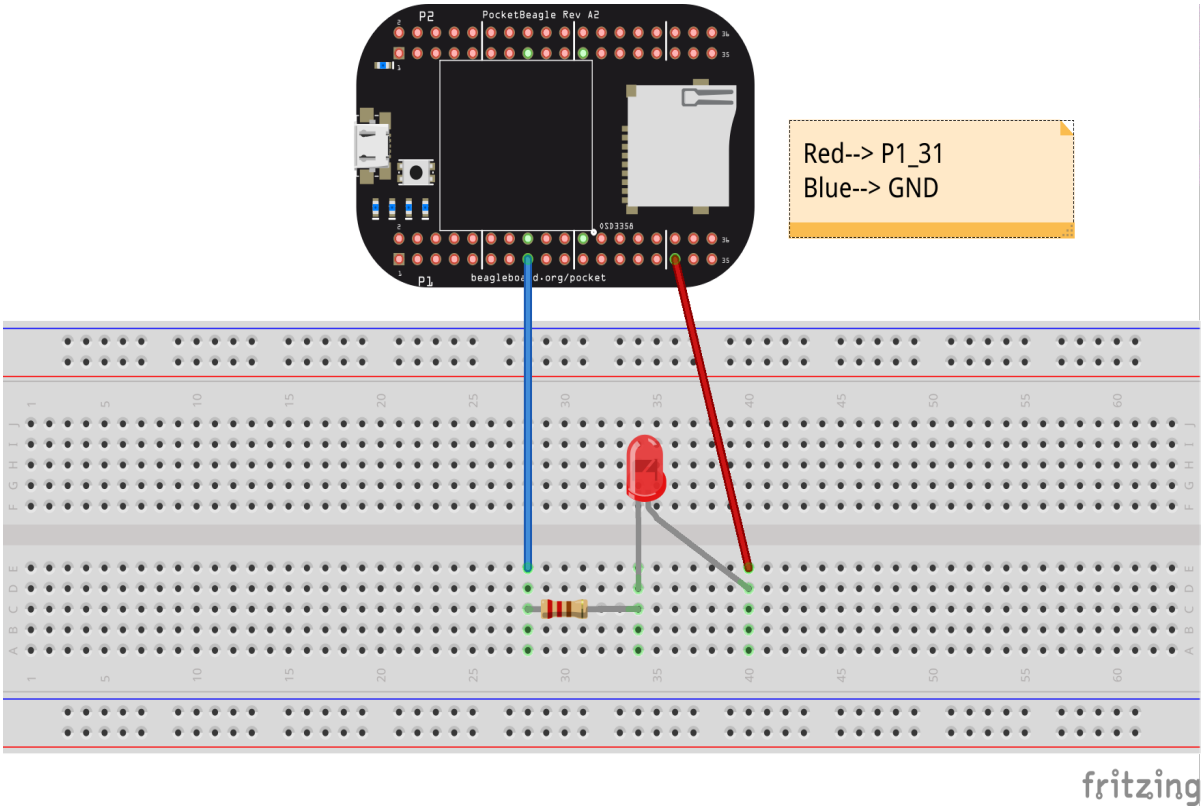
Code

```
while : true {
    if : digital_read(P1_29) {
        digital_write(P1_31, false);
    }
    else {
        digital_write(P1_31, true);
    }
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation This code runs a never ending loop, since it is `while : true`. Inside `while` it checks if header pin P1_29 is HIGH or LOW. If header pin P1_29 is HIGH, header pin P1_31 is set to LOW, and if header pin P1_29 is LOW, header pin P1_31 is set to HIGH.

Digital write example



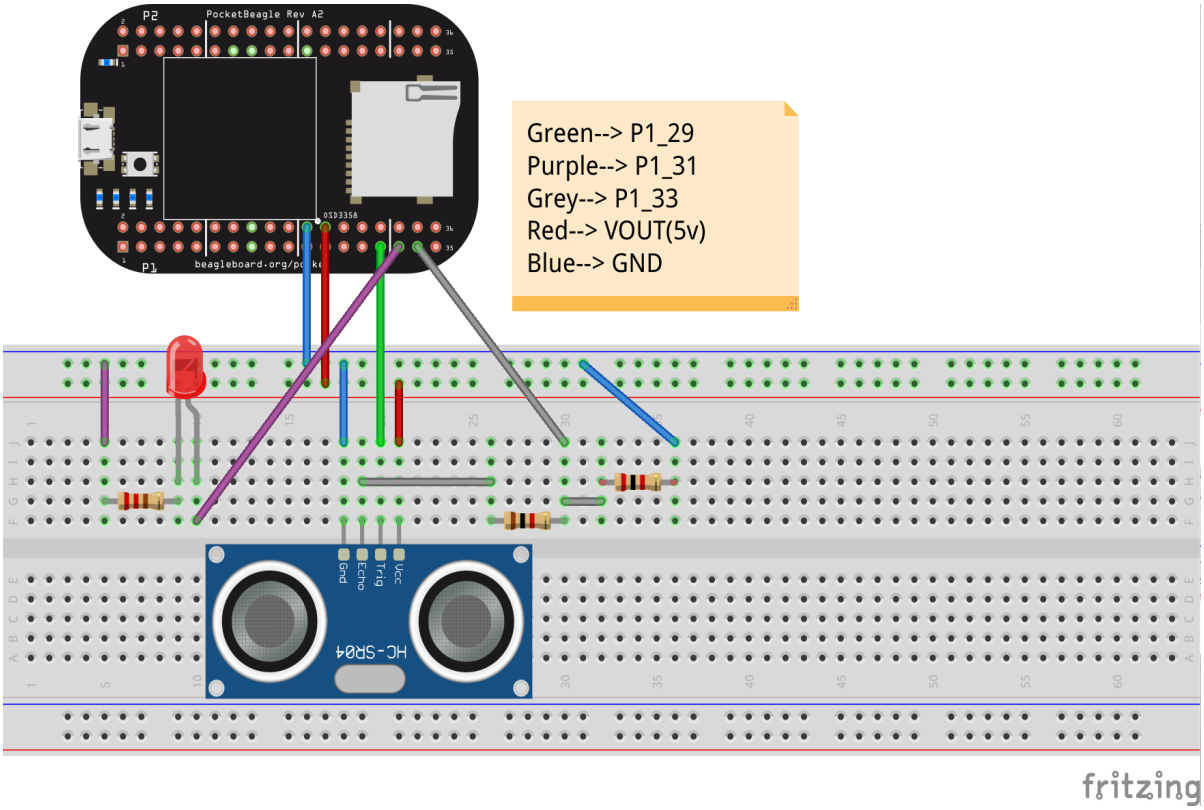
Code

```
while : true {
    digital_write(P1_31, true);
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation This code runs a never ending loop, since it is `while : true`. Inside `while` it sets header pin P1_31 to HIGH.

HCSR04 Distance Sensor example (sending distance data to ARM using RPMSG)



Code

```
def measure : int : {
    bool timeout := false;
    int echo := -1;

    start_counter();

    while : read_counter() <= 2000 {
        digital_write(5, true);
    }
    digital_write(5, false);
    stop_counter();

    start_counter();
    while : not (digital_read(6)) and true {
        if : read_counter() > 200000000 {
            timeout := true;
            break;
        }
    }
    stop_counter();

    if : not(timeout) and true {
        start_counter();
        while : digital_read(6) and true {
            if : read_counter() > 200000000 {
                timeout := true;
                break;
            }
            echo := read_counter();
        }
```

(continues on next page)

(continued from previous page)

```
        stop_counter();
    }

    if : timeout and true {
        echo := 0;
    }

    return echo;
}

init_message_channel();

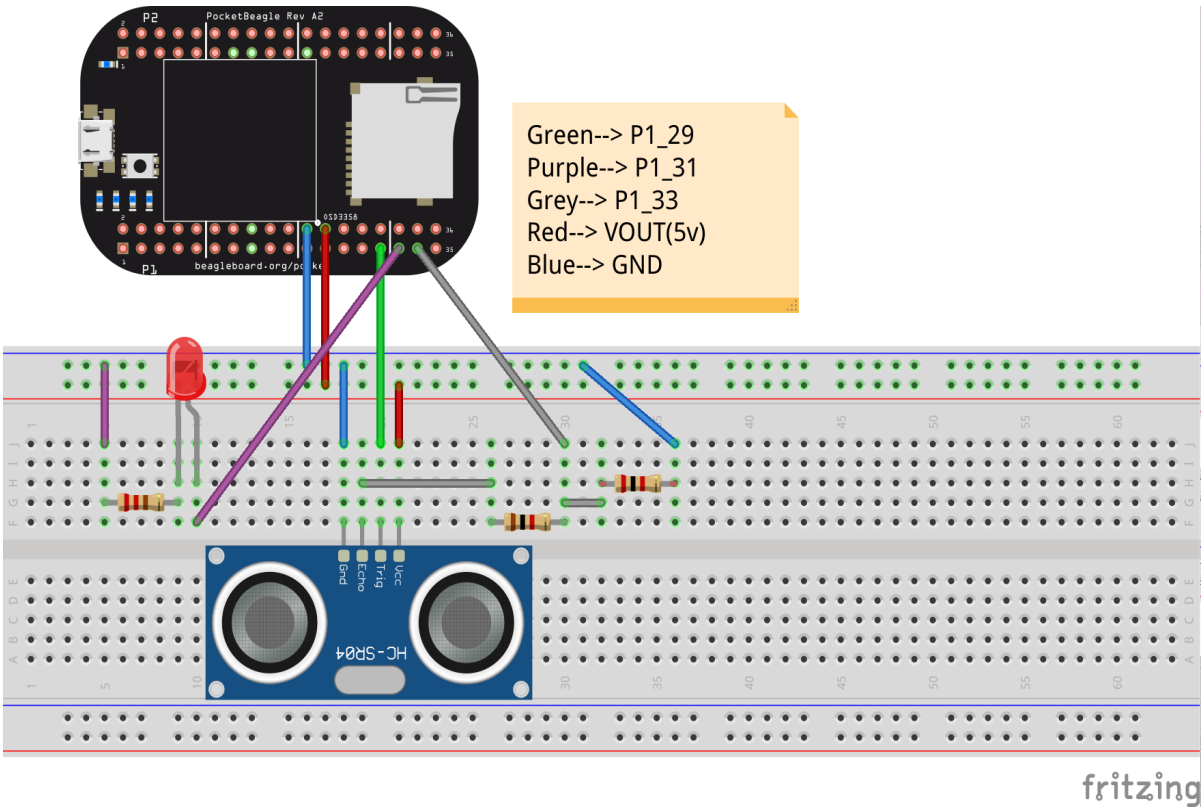
while : true {
    int ping:= measure();

    send_message(ping);
    delay(1000);
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation

Ultrasonic range sensor example



Code

```
def measure : int : {
    bool timeout := false;
    int echo := 0;
```

(continues on next page)

(continued from previous page)

```
    start_counter();

    while : read_counter() <= 2000 {
        digital_write(7, true);
    }
    digital_write(7, false);
    stop_counter();

    start_counter();
    while : not (digital_read(1)) and true {
        if : read_counter() > 200000000 {
            timeout := true;
            break;
        }
    }
    stop_counter();

    if : not(timeout) and true {
        start_counter();
        while : digital_read(1) and true {
            if : read_counter() > 200000000 {
                timeout := true;
                break;
            }
            echo := read_counter();
        }
        stop_counter();
    }

    if : timeout and true {
        echo := 0;
    }

    return echo;
}

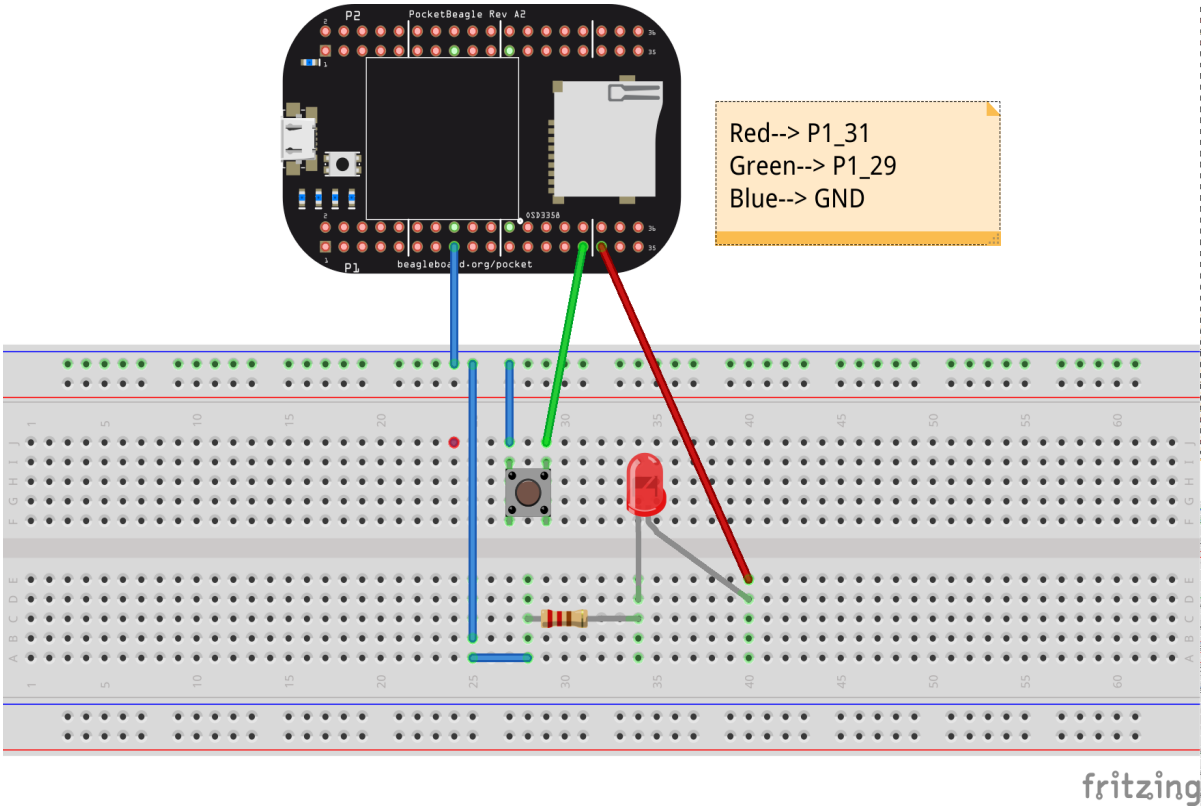
while : true {
    int ping:= measure()*1000;

    if : ping > 292200 {
        digital_write(4, false);
    }
    else
    {
        digital_write(4, true);
    }
    delay(1000);
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation

Sending state of button using RPMSG



Code

```
init_message_channel();

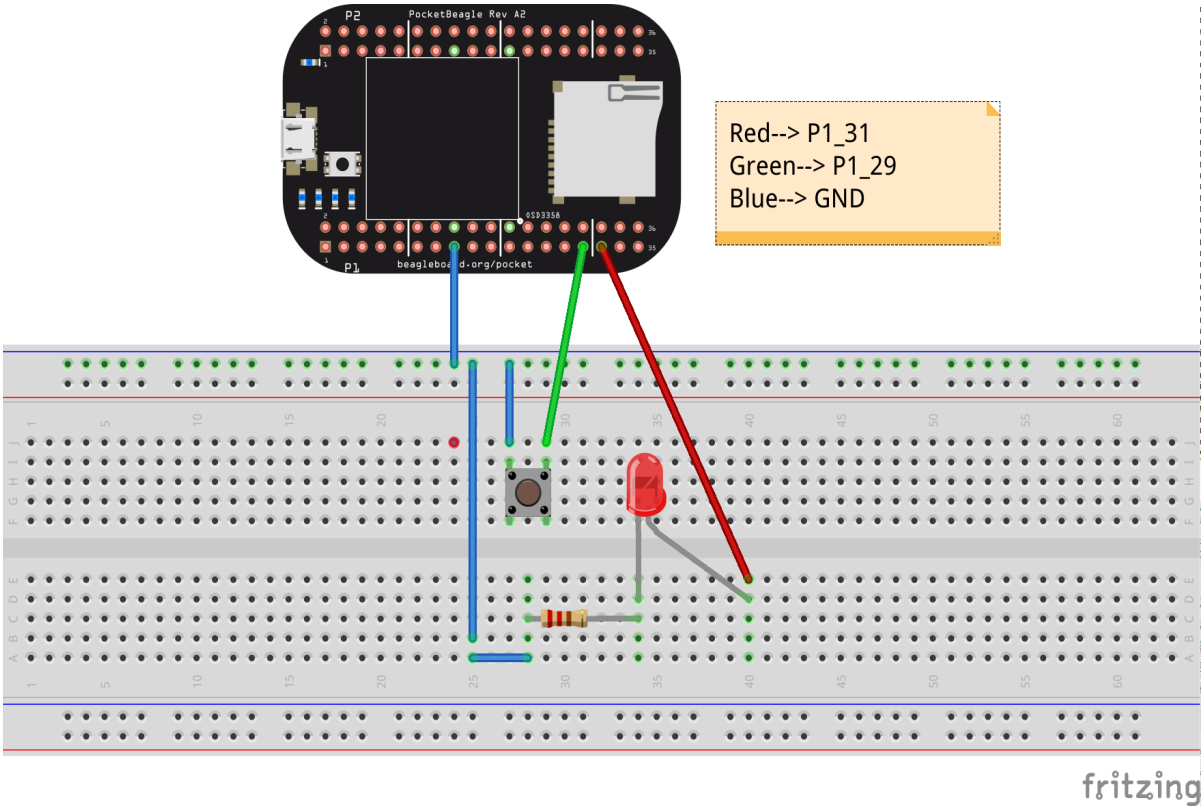
while : true {
    if : digital_read(P1_29) {
        send_message(1);
    }
    else {
        send_message(0);
    }
    delay(100);
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation `init_message_channel` is needed to setup communication channel between ARM<->PRU. It only needs to be called once, before using RPMSG functions.

`while : true` loop runs endlessly, inside this, we check for value of header pin P1_29, if it reads HIGH, 1 is sent to the ARM core using `send_message` and if it is LOW, 0 is sent to ARM core using `send_message`. Then PRU waits for 100ms, and repeats the steps again and again.

LED blink on button press example



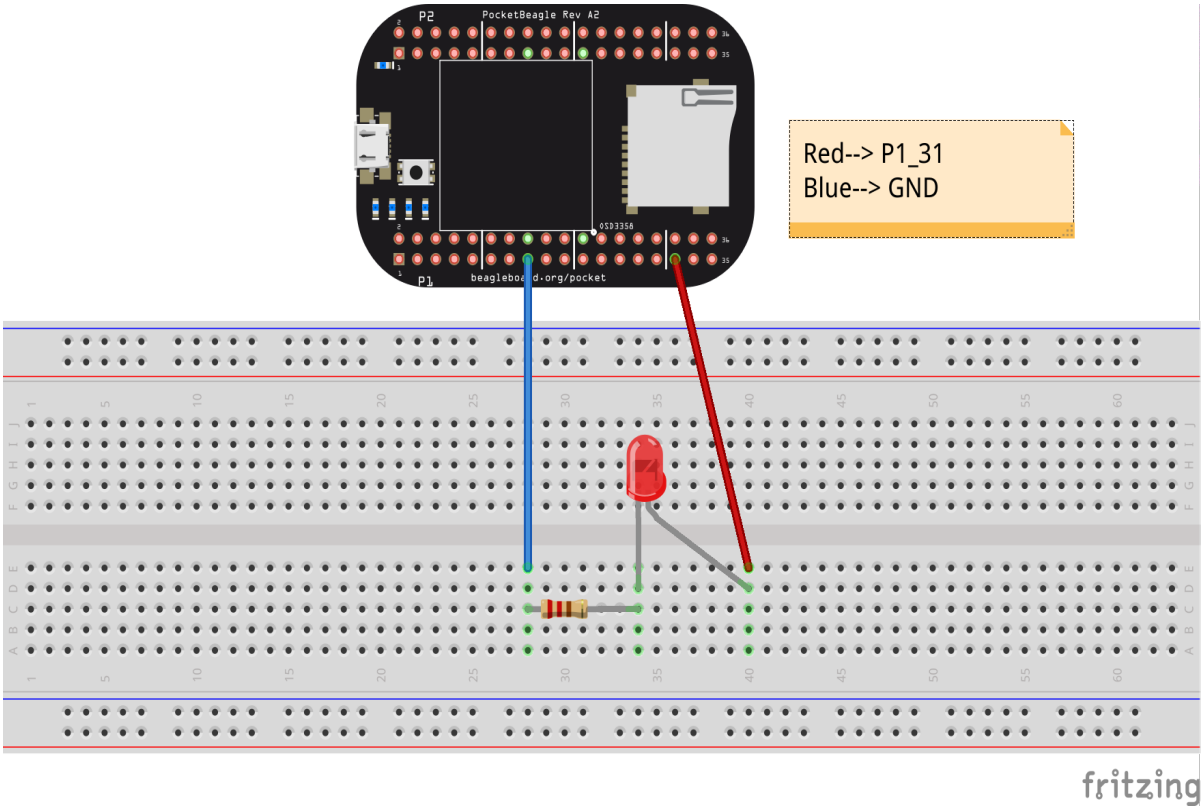
Code

```
while : true {
    if : digital_read(P1_29) {
        digital_write(P1_31, false);
    }
    else {
        digital_write(P1_31, true);
    }
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation This code runs a never ending loop, since it is `while : true`. Inside `while` if header pin P1_29 is HIGH, then header pin P1_31 is set to HIGH, waits for 1000ms, then sets header pin P1_31 to LOW, then again it waits for 1000ms. This loop runs endlessly as long as header pin P1_29 is HIGH, so we get a Blinking output if one connects a LED to output pin.

LED blink using for loop example



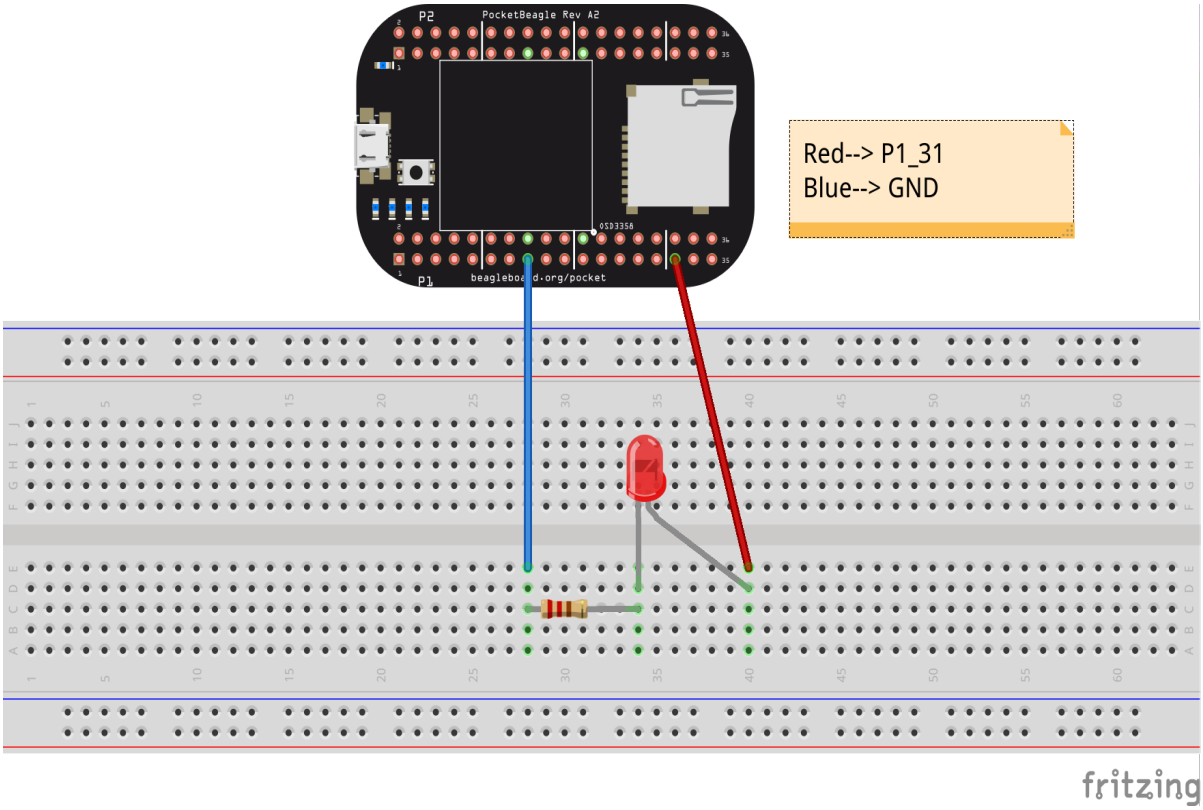
Code

```
for : l in 0:10 {
    digital_write(P1_31, true);
    delay(1000);
    digital_write(P1_31, false);
    delay(1000);
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation This code runs for loop with 10 iterations, Inside `for` it sets header pin P1_31 to HIGH, waits for 1000ms, then sets header pin P1_31 to LOW, then again it waits for 1000ms. This loop runs endlessly, so we get a Blinking output if one connects a LED. So LED will blink 10 times with this code.

LED blink using while loop example



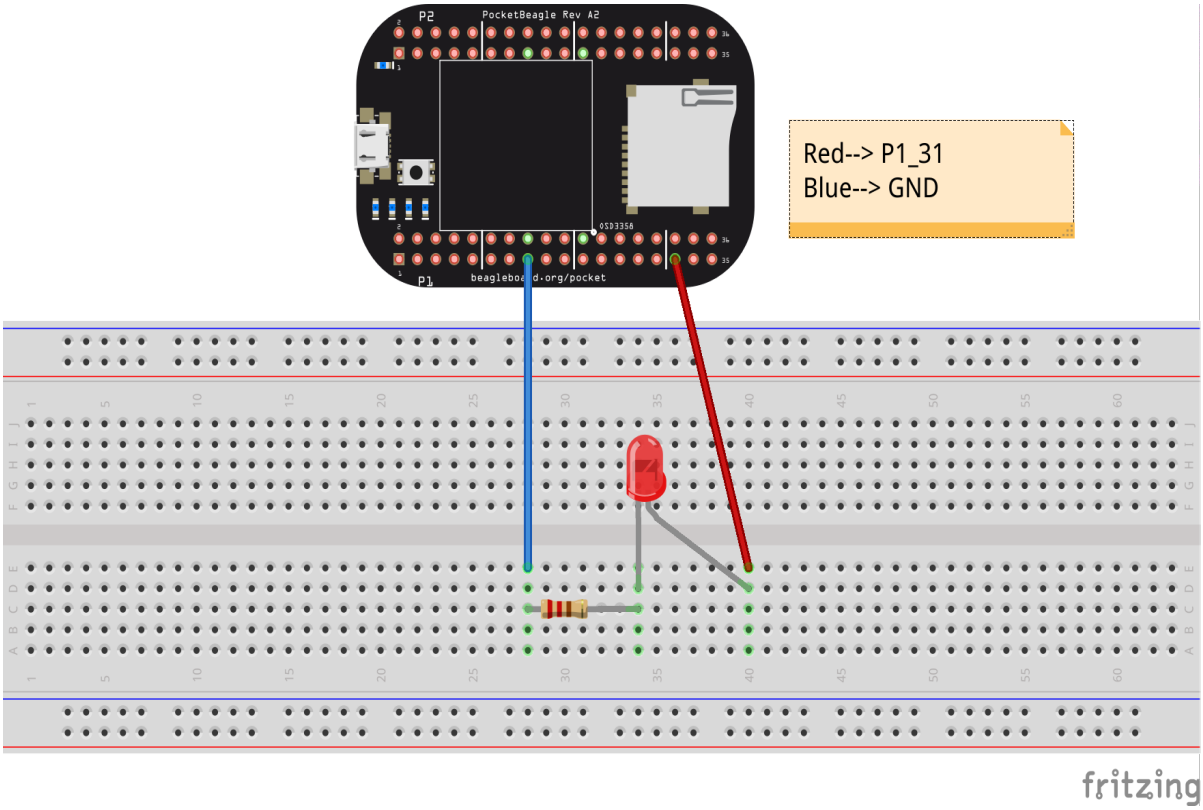
Code

```
while : true {
    digital_write(P1_31, true);
    delay(1000);
    digital_write(P1_31, false);
    delay(1000);
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation This code runs a never ending while loop, since it is `while : true`. Inside `while` it sets header pin P1_31 to HIGH, waits for 1000ms, then sets header pin P1_31 to LOW, then again it waits for 1000ms. This loop runs endlessly, so we get a Blinking output if one connects a LED

LED blink example



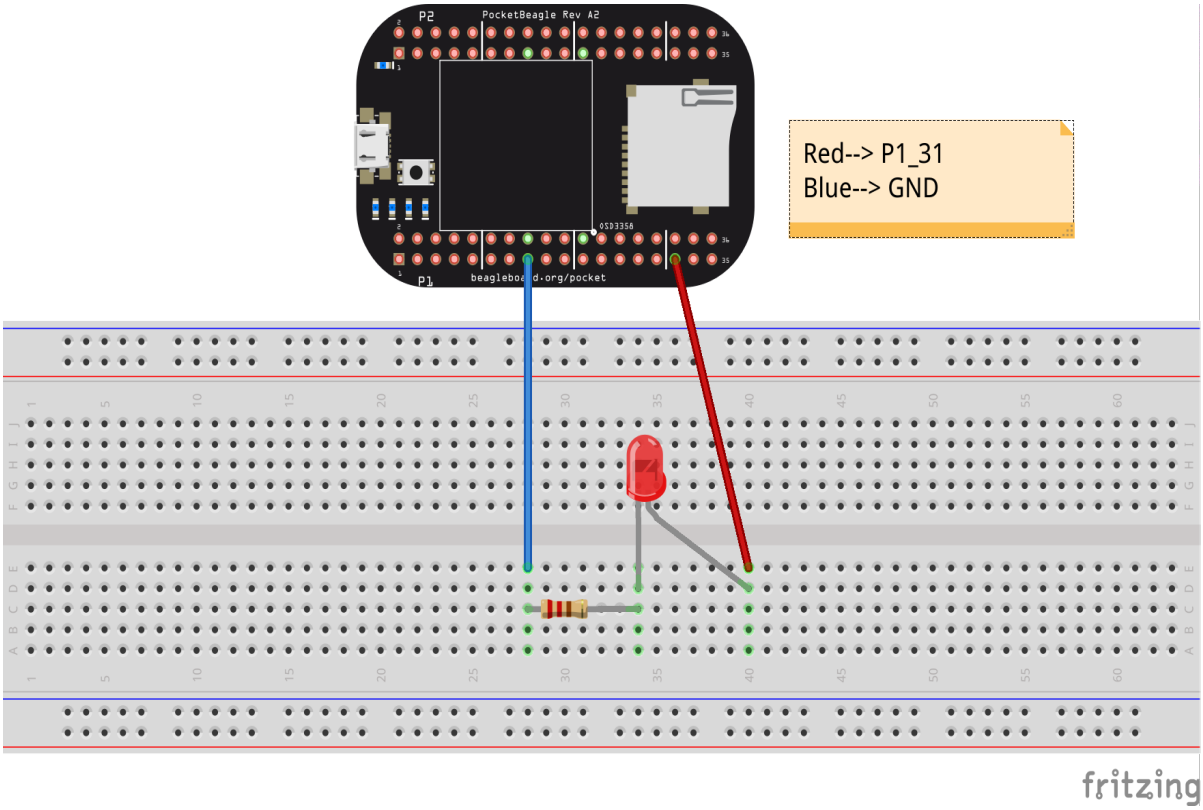
Code

```
while : 1 == 1 {
    digital_write(P1_31, true);
    delay(1000);
    digital_write(P1_31, false);
    delay(1000);
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation This code runs a never ending loop, since it is `while : true`. Inside `while` it sets header pin P1_31 to HIGH, waits for 1000ms, then sets header pin P1_31 to LOW, then again it waits for 1000ms. This loop runs endlessly, so we get a Blinking output if one connects a LED

LED blink using hardware counter



Code

```
while : true {
    start_counter();
    while : read_counter() < 200000000 {
        digital_write(P1_31, true);
    }
    stop_counter();

    start_counter();
    while : read_counter() < 200000000 {
        digital_write(P1_31, false);
    }
    stop_counter();
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation This code runs a never ending while loop, since it is `while : true`. Inside `while` it starts the counter, then in a nested while loop, which runs as long as `read_counter` returns values less than 200000000, so for 200000000 cycles, HIGH is written to header pin P1_31, and after the while loop ends, the counter is stopped.

Similarly counter is started again, which runs as long as `read_counter` returns a value less than 200000000, so for 200000000 cycles, LOW is written to header pin P1_31, and after the while loop ends, the counter is stopped.

This process goes on endlessly as it is inside a never ending while loop. Here, we check if `read_counter` is less than 200000000, as counter takes exactly 1 second to count this much cycles, so basically the LED is turned on for 1 second, and then turned off for 1 second. Thus if a LED is connected to the pin, we get a endlessly blinking LED.

Read hardware counter example

Code

```
start_counter();
while : read_counter() < 200000000 {
    digital_write(4, true);
}
stop_counter();
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation Since, PRU's hardware counter works at 200 MHz, it counts up to 2 x 108 cycles in 1 second. So, this can be reliably used to count time without using `delay`, as we can find exactly how much time 1 cycle takes.

2 x 108 cycles/second.
1 Cycles = 0.5 x 10-8 seconds.

So, it can be used to count how many cycles have passed since, say we received a high input on pin 3. `start_counter` starts the counter, and `read_counter` reads the current state of the counter, and `stop_counter` stops the counter.

Using RPMSG to communicate with ARM core

Code

```
init_message_channel();

int count := receive_message();

while : true {
    send_message(count);
    count := count + 1;
    delay(1000);
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation PRU has a functionality to communicate with the ARM core, it is called RPMSG. This examples show how to use RPMSG functionality to communicate with ARM core using RPMSG.

`init_message_channel` is needed to setup communication channel between ARM<->PRU. It only needs to be called once, before using RPMSG functions.

`int count := receive_message();` waits for a message from ARM Core, we need to send some integer to PRU with which to start the counting. So, say we send 3, then int variable count will be equal to 3.

After this, there is `while : true` block which runs endlessly. Inside the block there is a `send_message` call, this sends message back to the ARM Core.

So, inside the for loop we are sending value of count variable, after this we increase value of count by 1. Then we wait for 1000ms, and repeat the above steps again and again.

Using RPMSG to implement a simple calculator on PRU

Code

```
init_message_channel();

while : true {
    int option := receive_message();
    int a := receive_message();
    int b := receive_message();

    if : option == 1 {
        send_message(a+b);
    }
    elif : option == 2 {
        send_message(a-b);
    }
    elif : option == 3 {
        send_message(a*b);
    }
    elif : option == 4 {
        if : b != 0 {
            send_message(a/b);
        }
        else {
            send_message(a);
        }
    }
    else
    {
        send_message(a+b);
    }
}
```

- Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

Explanation `init_message_channel();` starts the message channel for communication with ARM <-> PRU cores. Then `while : true` loops runs endlessly.

`int option := receive_message();` receives which operator to be executed and stores it in option variable. 1 for addition, 2 for subtractions, 3 for multiplication and 4 for division. `int a := receive_message();` receives the value of first operand, and `int b := receive_message();` receives the value of second operand.

if-elseif ladder checks if value of option is 1, 2, 3 or 4 and accordingly sends the value of operation back to ARM core using `send_message`. While division, it makes sure that divisor is not 0. If value of option is anything other than 1, 2, 3, 4, then it defaults to else condition, that is a+b.

This runs endlessly since it is inside a `while : true` loop.

14.2 BB-Config

14.2.1 BB-Config Detail

Configure your beagle devices easily.

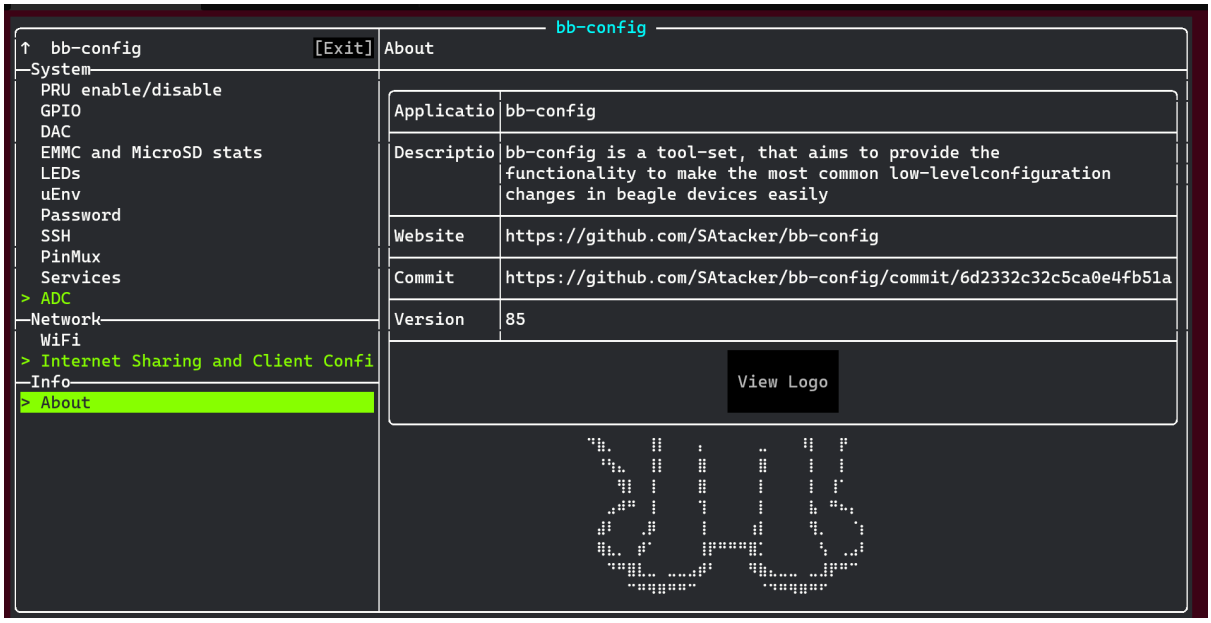
Github



What is BB-Config

BB-Config is software that makes the most common low-level configuration changes of beagle devices easily and provides a terminal UI.

BB-Config is using FTXUI (C++ Functional Terminal User Interface) which provides a simple and elegant looking UI.



14.2.2 Build from Source

Dependencies

- g++
- cmake
- glib-2.0
- libnm

Build

```
git clone https://git.beagleboard.org/gsoc/bb-config
cd bb-config
mkdir build
cd build
cmake ..
make -j$(nproc)
```

Install

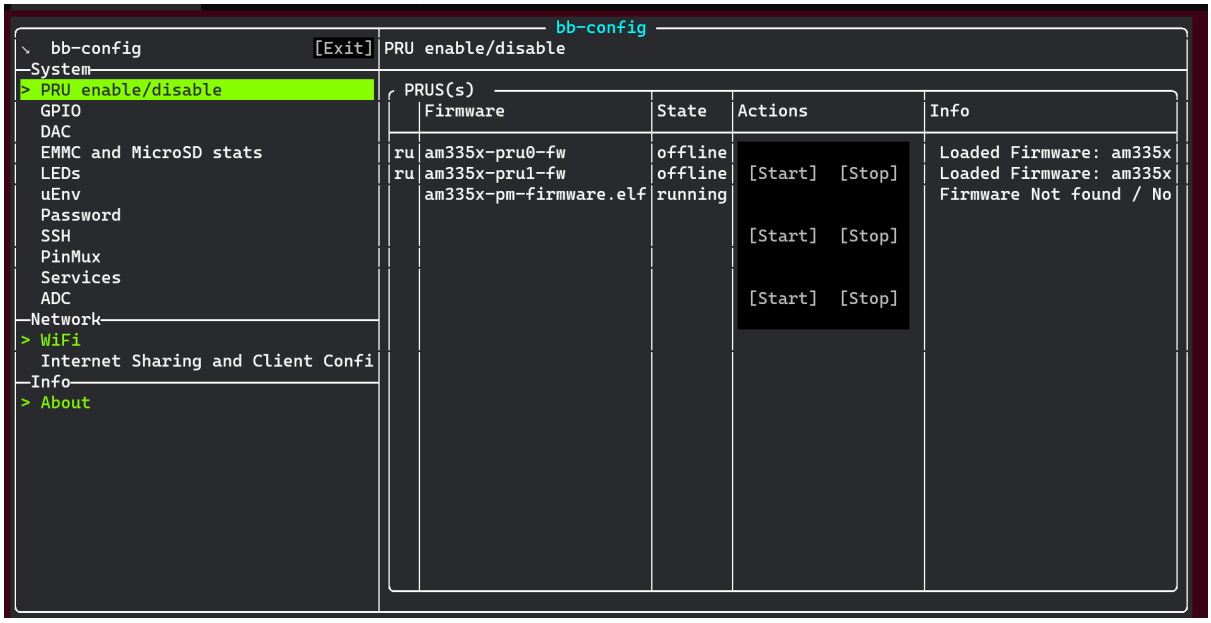
```
sudo make install
```

14.2.3 Features

BB-Config v1.x

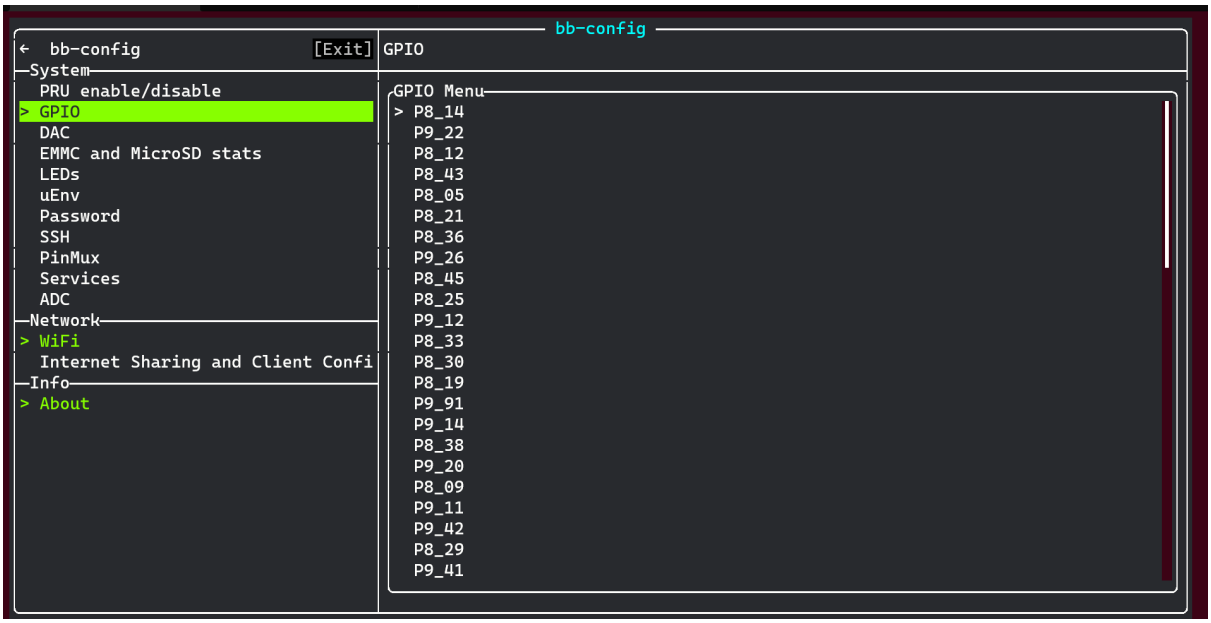
PRU Enable/Disable

- Enable/Disable PRU



GPIO

- Turn On/Off gpio



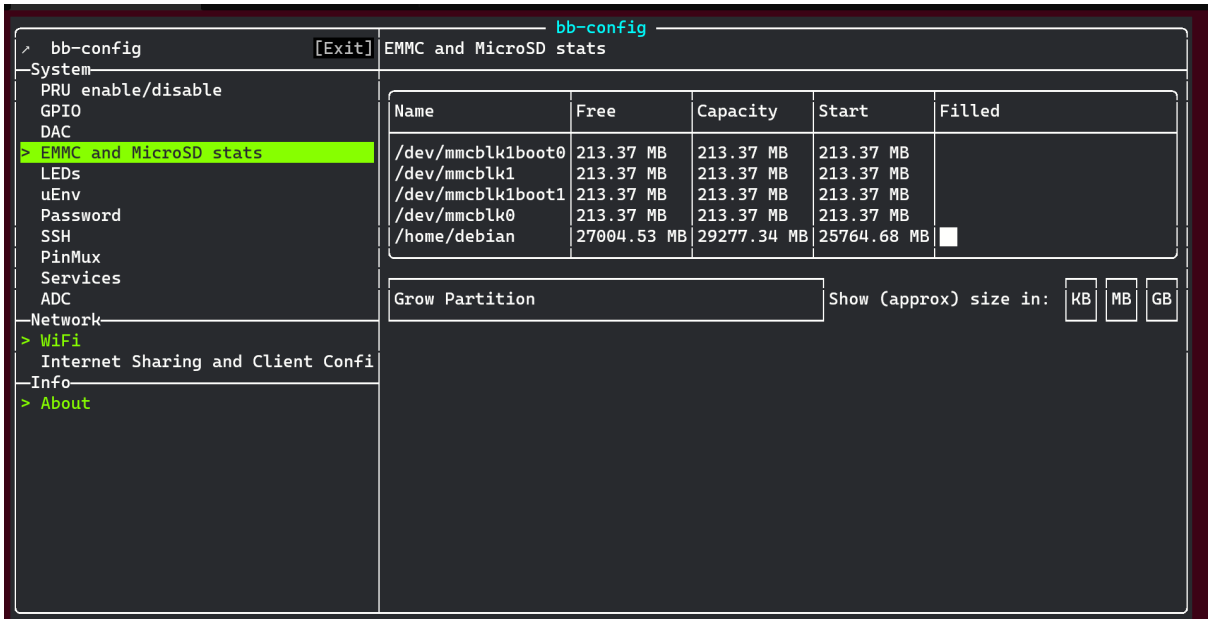
GPIO Menu



GPIO Setting

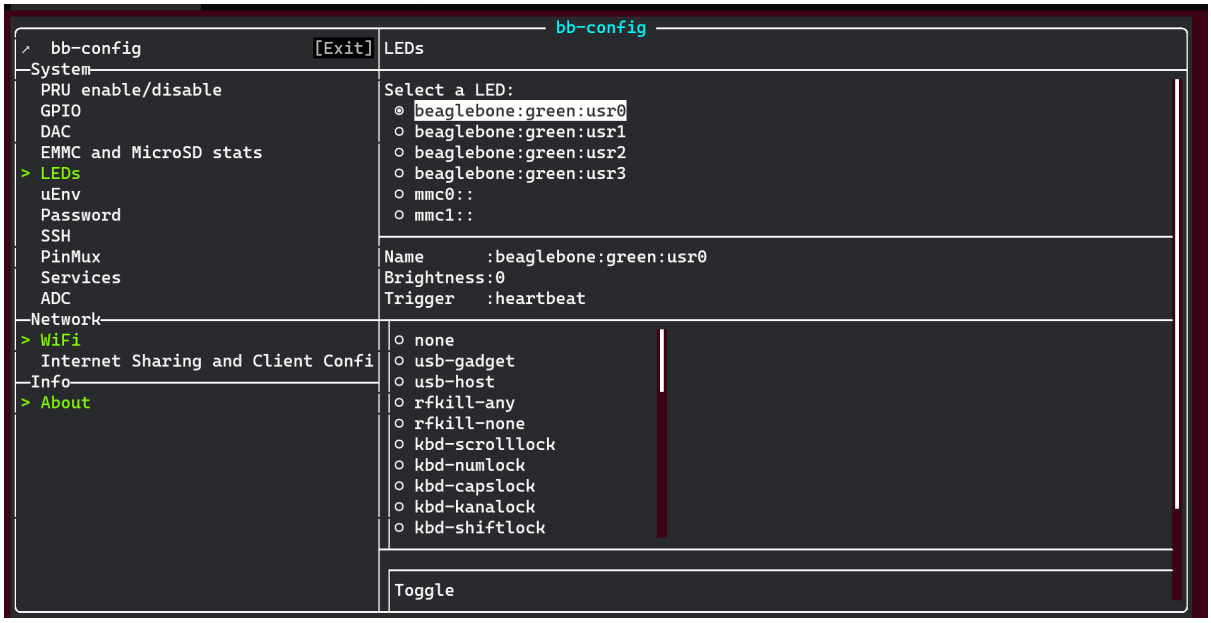
EMMC and MicroSD Stats

- Storage stats & grow partition



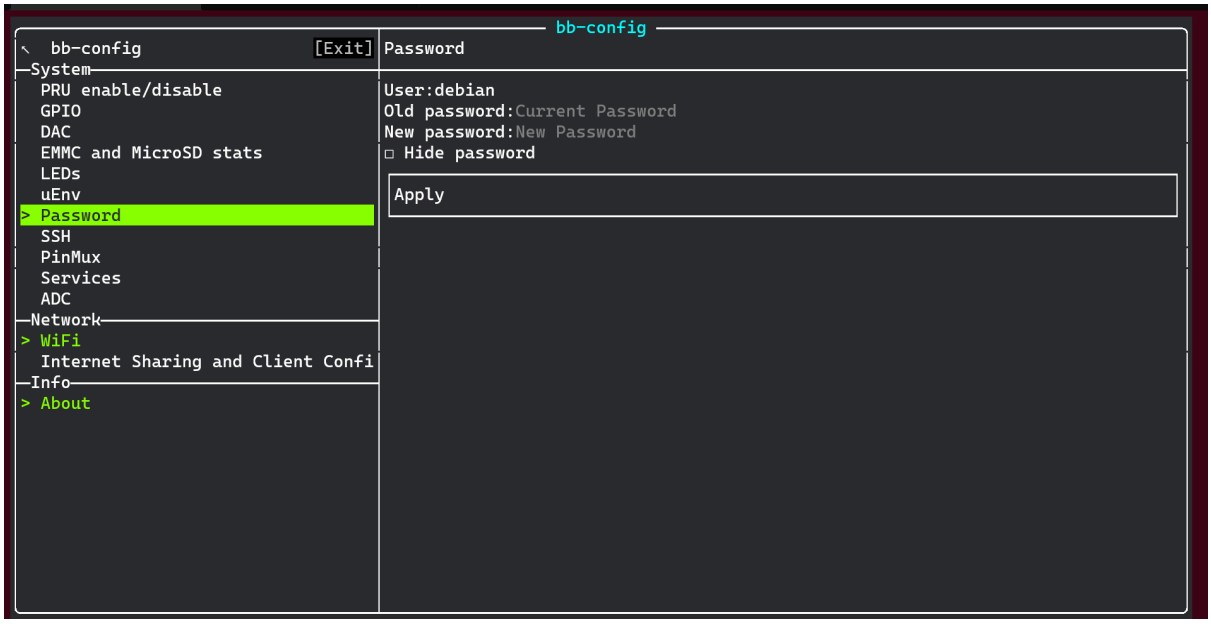
LEDs

- Config board build in LEDs



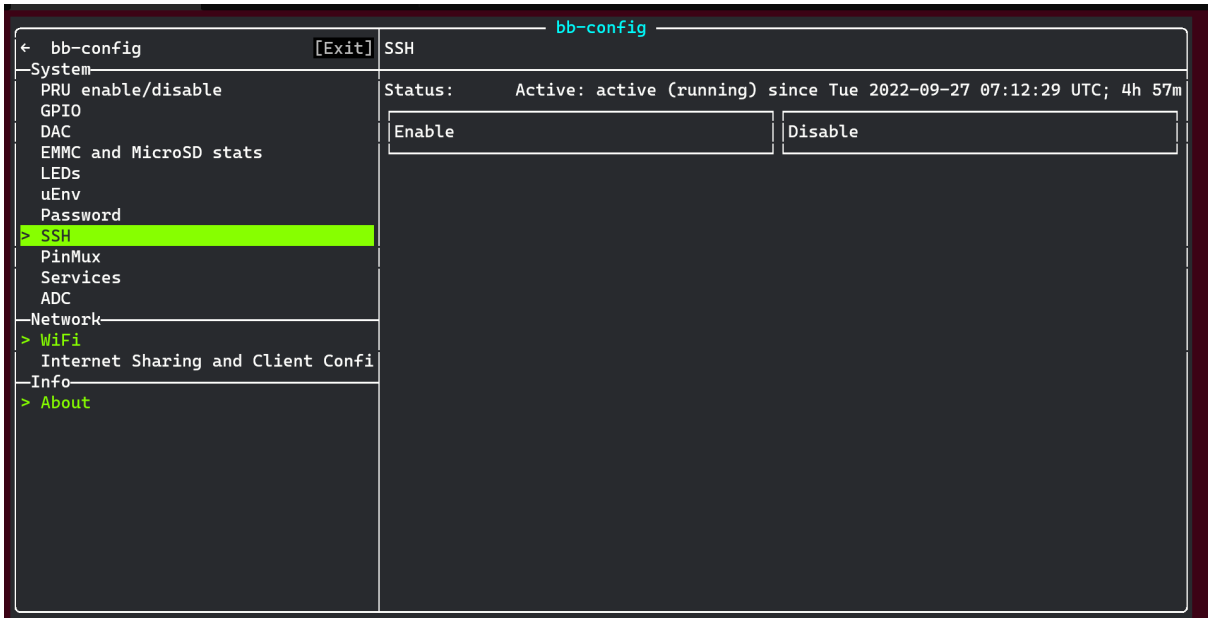
Password

- Change users password



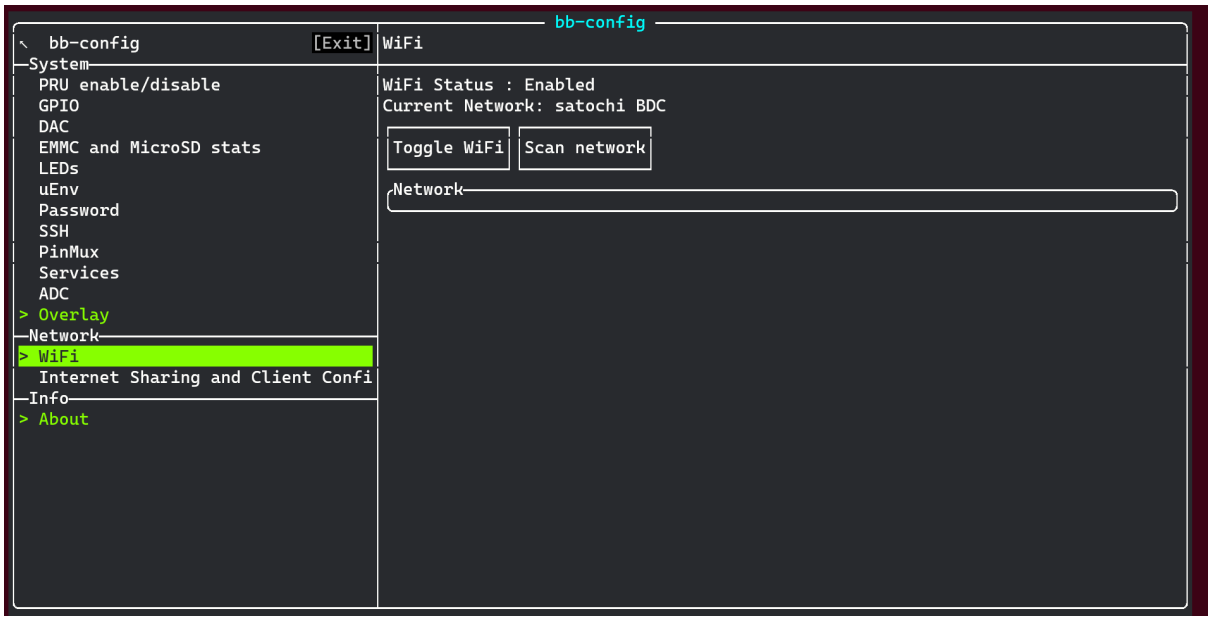
SSH

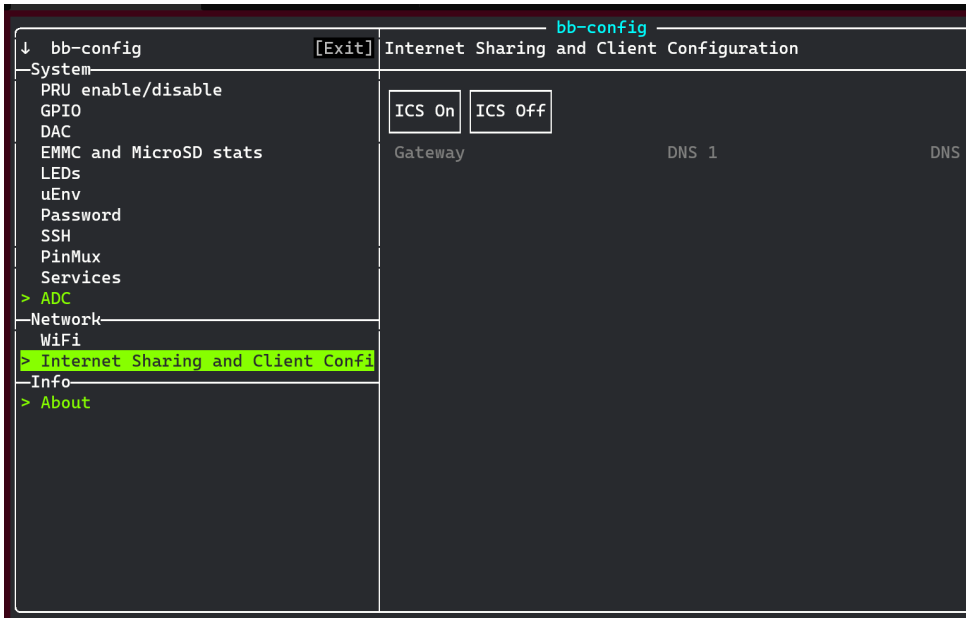
- Enable/Disable SSH



WiFi

- Connect to Wi-Fi

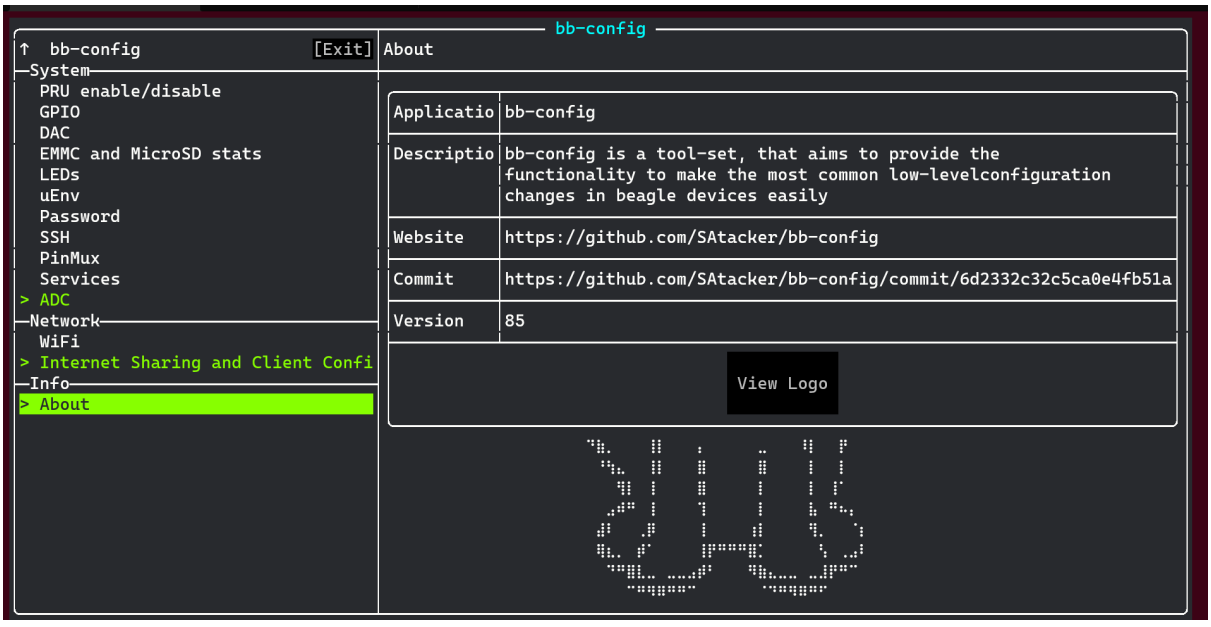




Internet Sharing and Client Config

- Note: You'll have to configure your host Following is an example script:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables --table nat --append POSTROUTING --out-interface wlp4s0 -j␣
↪MASQUERADE
iptables --append FORWARD --in-interface wlp4s0 -j ACCEPT
```

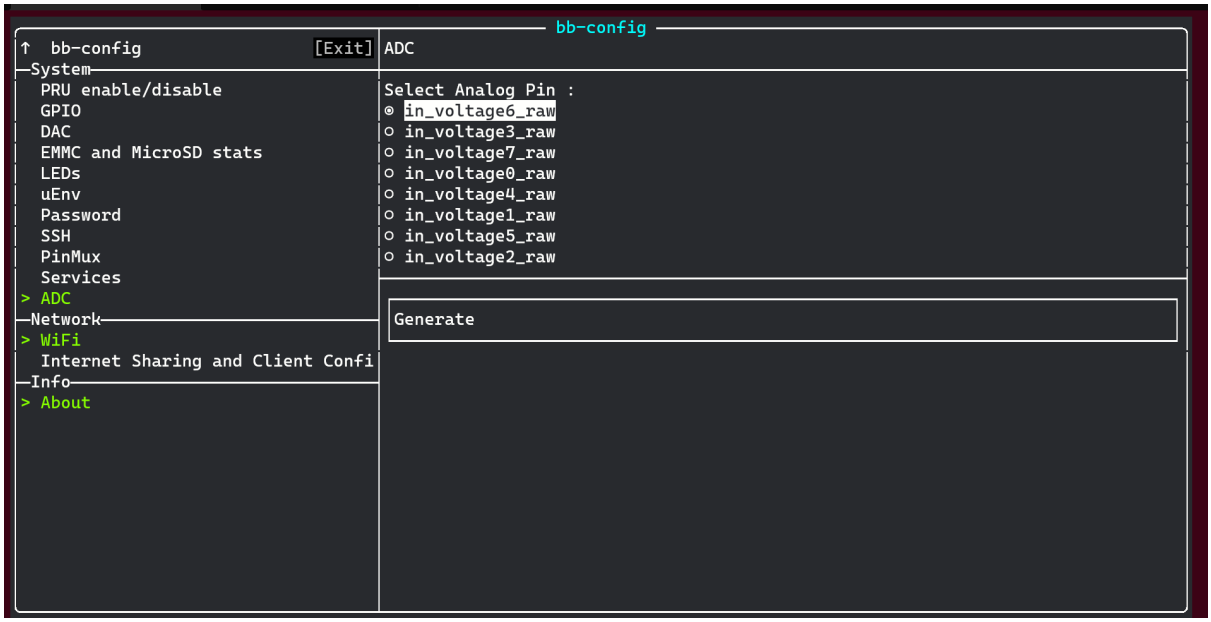


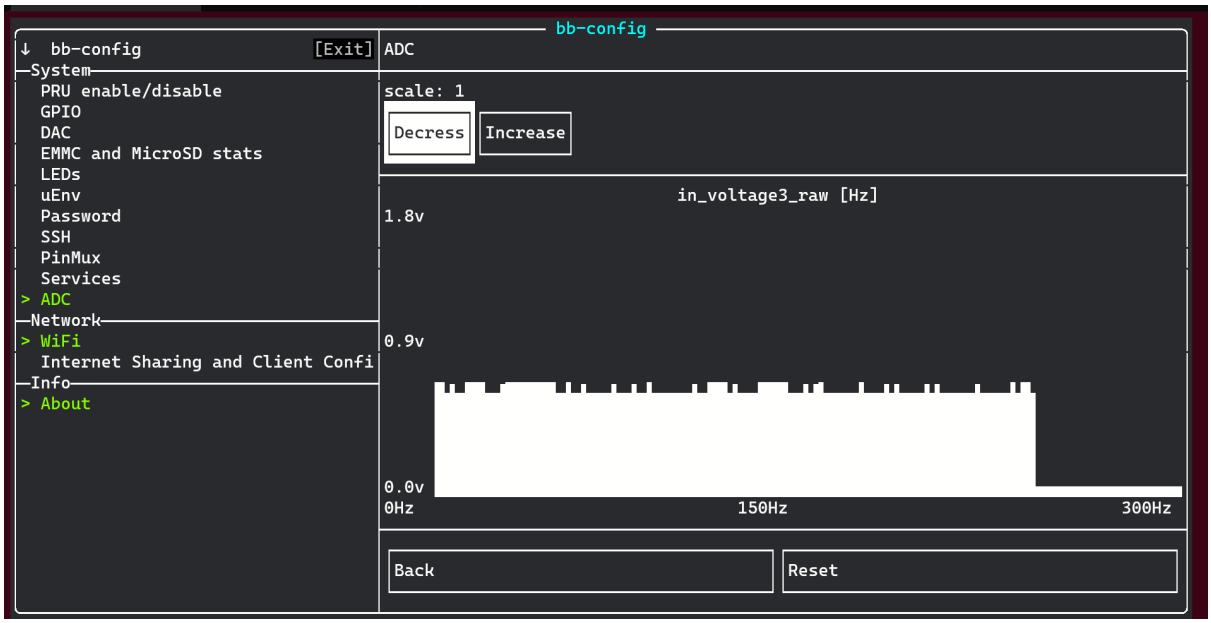
About

BB-Config v2.x

ADC (Graph)

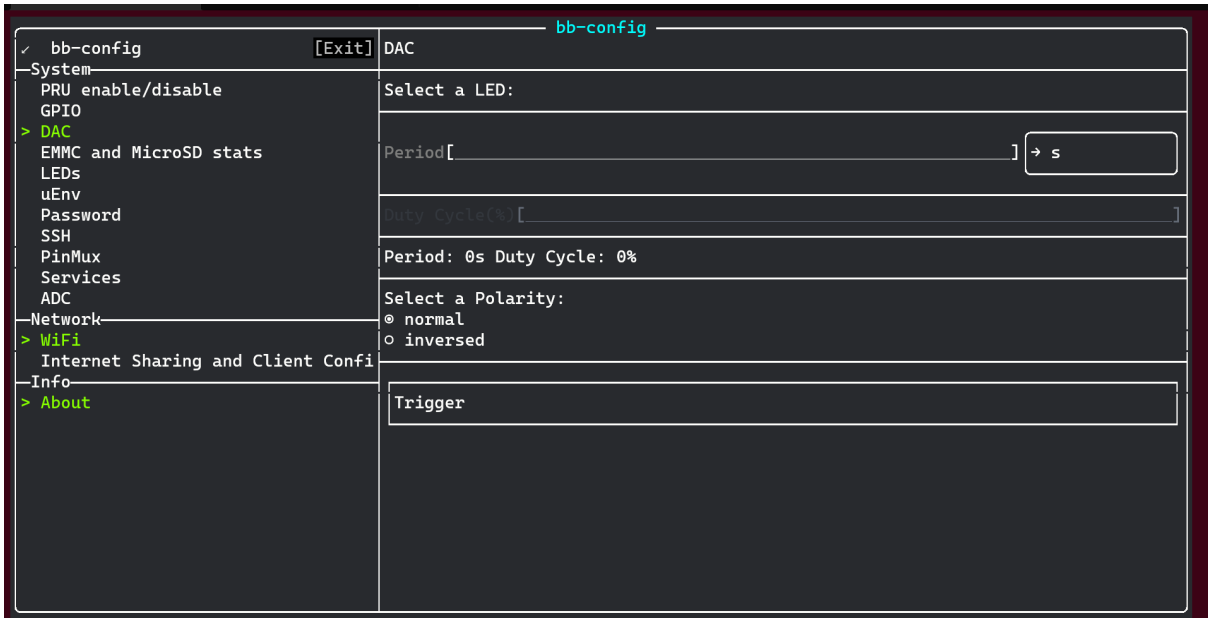
- Plot graph for Analogue pin





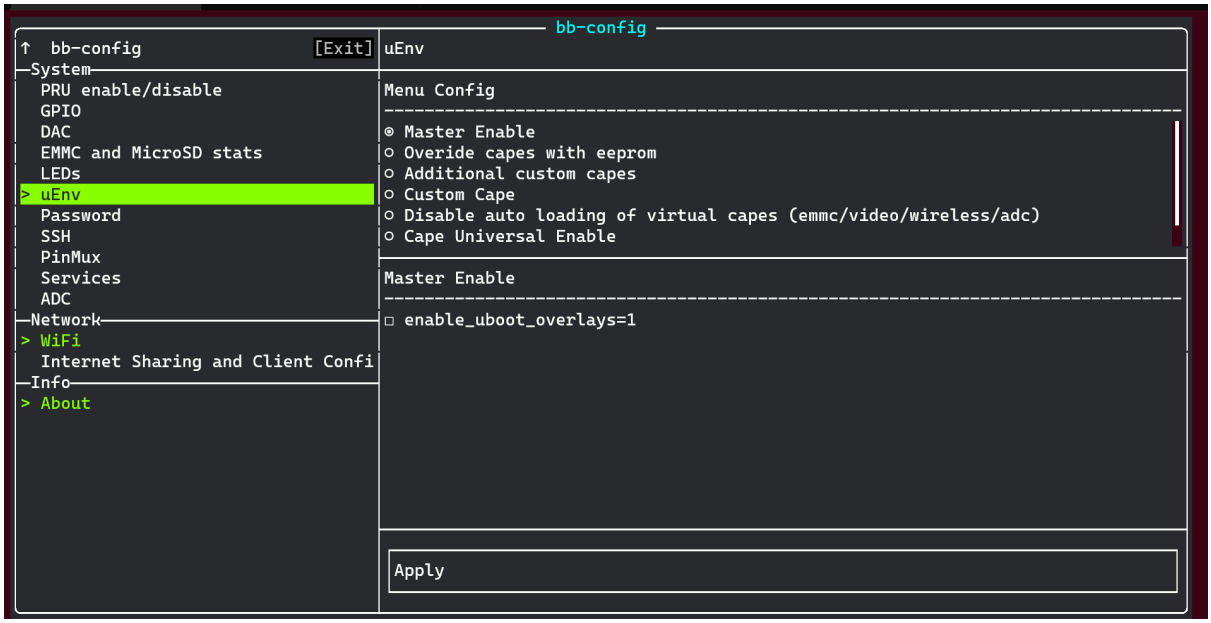
DAC (PWM)

- Generate PWM waveform



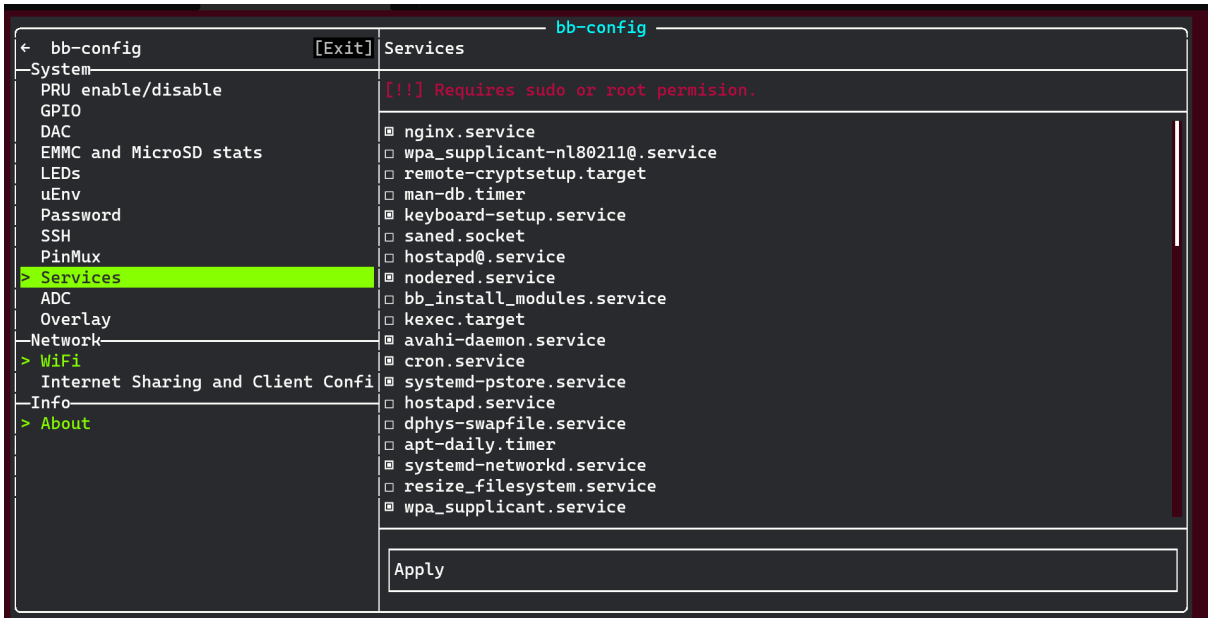
uEnv

- Enable/Disable boot configuration



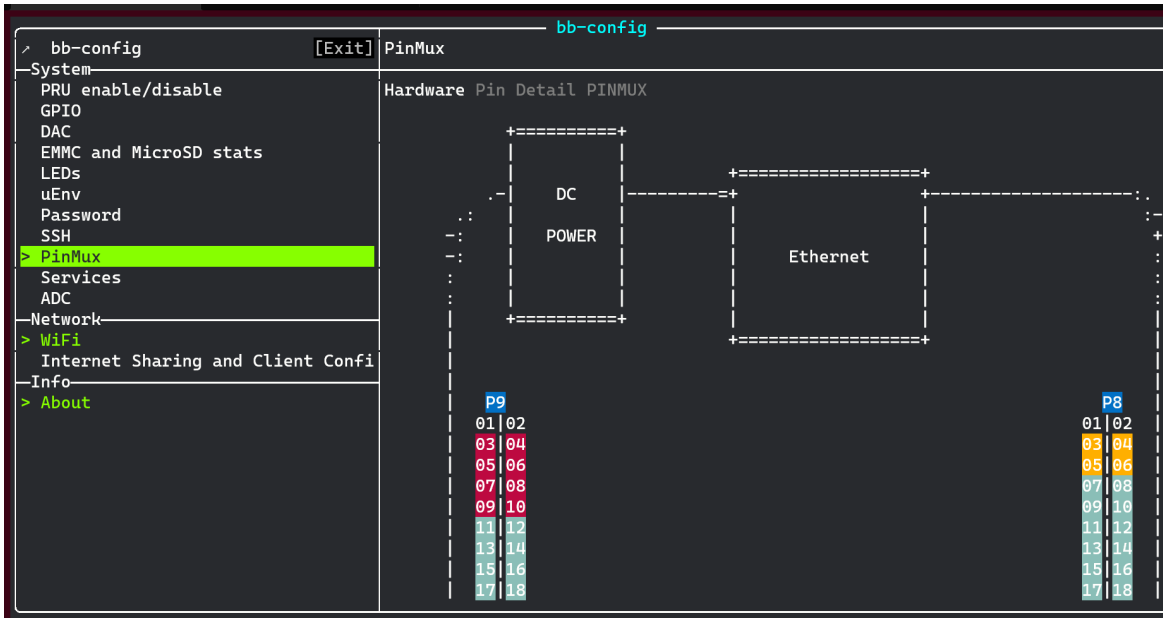
services

- Enable/Disable services startup at boot



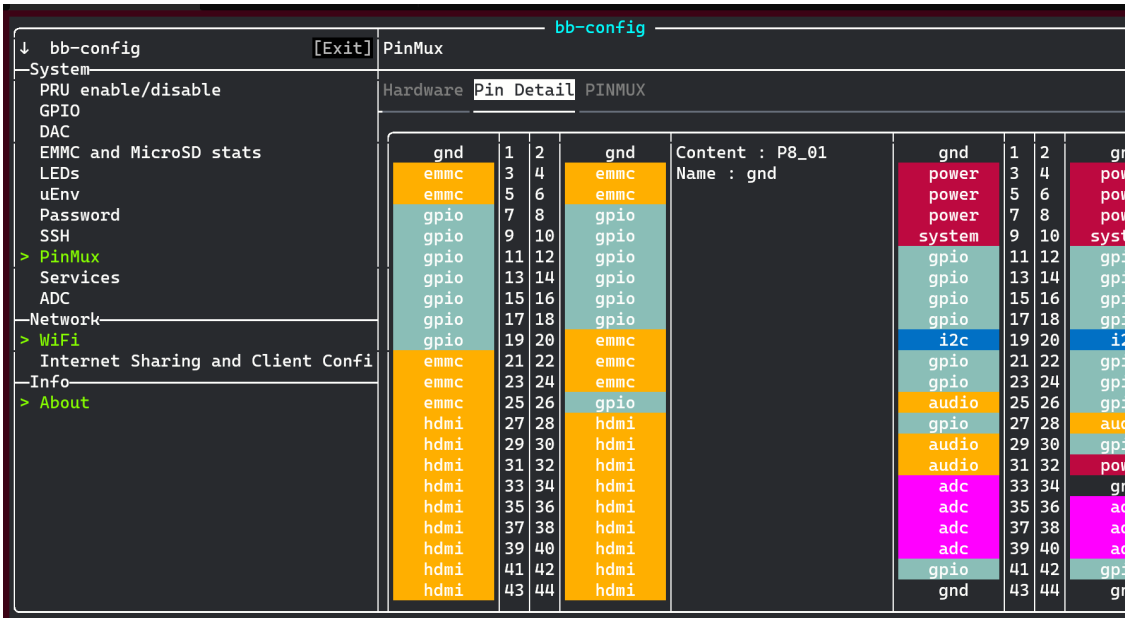
PINMUX

- Display PIN I/O detail
- Config PINMUX

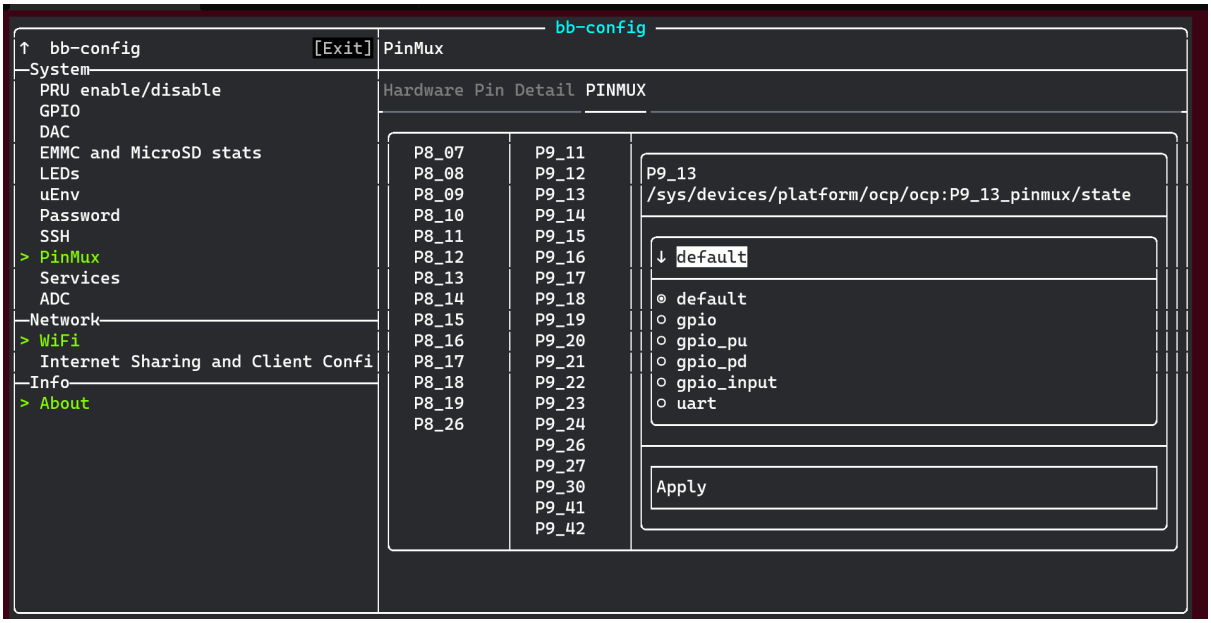


Hardware Display

Pin Table References

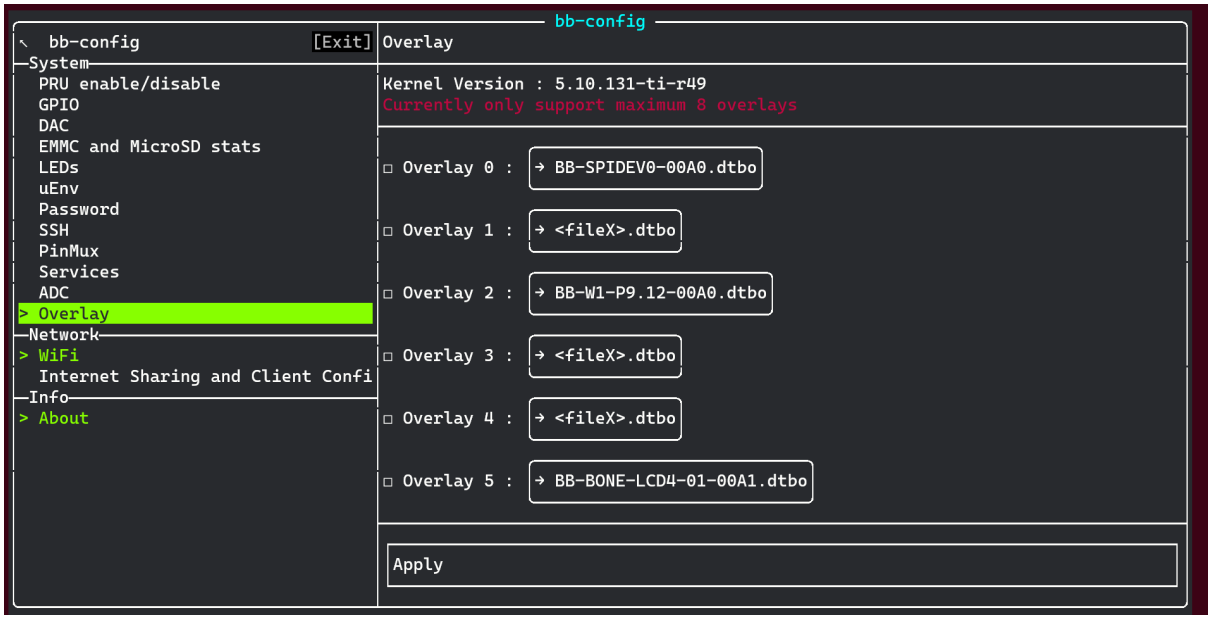


Pin Config



Overlay (dts)

- Enable/Disable Device Tree Overlay in Boot option
- Select dtbo file and automate update in uEnv.txt



WiFi (D-Bus)

- Connect to WiFi with wpa_supplicant
- Support for Debian 11



14.2.4 Version

GSOC@21 BB-Config v1.x

- Name: Shreyas Atre
- Mentors: Arthur Sonzogni, Abhishek Kumar, Deepak Khatri.
- Organization: BeagleBoard.org
- Code: https://github.com/SAtacker/beagle-config
- Project Page: https://summerofcode.withgoogle.com/projects/#6718016412188672

- Progress Log: https://satacker.github.io/gsoc-log/
- Kanban: https://github.com/SAtacker/beagle-config/projects/1
- Initial Video: https://youtu.be/vFUWCzqE6xI

GSOC@22 BB-Config v2.x

- Name: Seak Jian De
- Mentors: Shreyas Atre, Vedant Paranjape, Vaishnav Achath.
- Organization: BeagleBoard.org
- Code: https://git.beagleboard.org/gsoc/bb-config
- Project Page: https://summerofcode.withgoogle.com/programs/2022/projects/2DbiYPlY
- Progress Log: https://forum.beagleboard.org/t/weekly-progress-report-bb-config-improvements-gpio-benchmark/32357/2
- Initial Video: https://youtu.be/V_Euk5uWY1o

14.3 Robotics Control Library

page **index**

This package contains the C library and example/testing programs for the Robot Control project. This project began as a hardware interface for the Robotics Cape and later the BeagleBone Blue and was originally called Robotics_Cape_Installer. It grew to include an extensive math library for discrete time feedback control, as well as a plethora of POSIX-compliant functions for timing, multithreading, program flow, and lots more. Everything is aimed at developing robotics software on embedded computers.

This package ships with official BeagleBone images and is focused on, but not excludive to, the BeagleBoard platform. The library and example programs are primarily written in C, however there also exists the RCPY python interface to this package available at https://github.com/mcdeoliveira/rcpy.

The master branch is always the most current but not necessarily stable. See releases page for older stable versions or install from BeagleBoard.org repositories.

To get started, visit the user manual

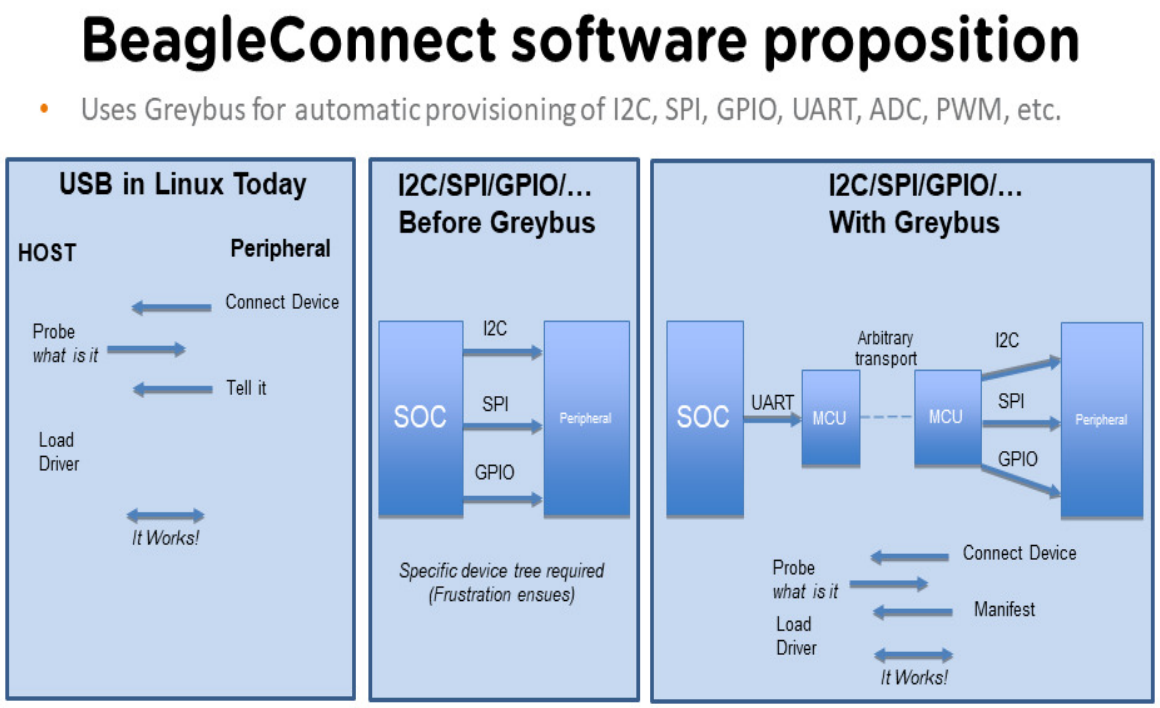
14.4 BeagleConnect Technology

This is the deep-dive introduction to BeagleConnect™ technology and software architecture.

Note: This documentation and the associated software are each a work-in-progress.

BeagleConnect™ is built using Greybus code in the Linux kernel originally designed for mobile phones. To understand a bit more about how the BeagleConnect™ Greybus stack is being built, this section helps describe the development currently in progress and the principles of operation.

14.4.1 Background



BeagleConnect™ uses Greybus and mikroBUS add-on boards with ClickID and BeagleConnect™ mikroBUS manifests to eliminate the need to add and manually configure devices added onto the Linux system.

14.4.2 High-level

- For Linux nerds: Think of BeagleConnect™ as 6LoWPAN over 802.15.4-based Greybus (instead of Unipro as used by Project Ara), where every BeagleConnect™ board shows up as new SPI, I2C, UART, PWM, ADC, and GPIO controllers that can now be probed to load drivers for the sensors or whatever is connected to them. (Proof of concept of Greybus over TCP/IP: https://www.youtube.com/watch?v=7H50pv-4YXw)
- For MCU folks: Think of BeagleConnect™ as a Firmata-style firmware load that exposes the interfaces for remote access over a secured wireless network. However, instead of using host software that knows how to speak the Firmata protocol, the Linux kernel speaks the slightly similar Greybus protocol to the MCU and exposes the device generically to users using a Linux kernel driver. Further, the Greybus protocol is spoken over 6LoWPAN on 802.15.4.

14.4.3 Software architecture

14.4.4 TODO items

- Linux kernel driver (wpanusb and bcfserial still need to be upstreamed)
- Provisioning
- Firmware for host CC13x
- Firmware for device CC13x
- Unify firmware for host/device CC13x
- Click Board drivers and device tree formatted metadata for 100 or so Click Boards

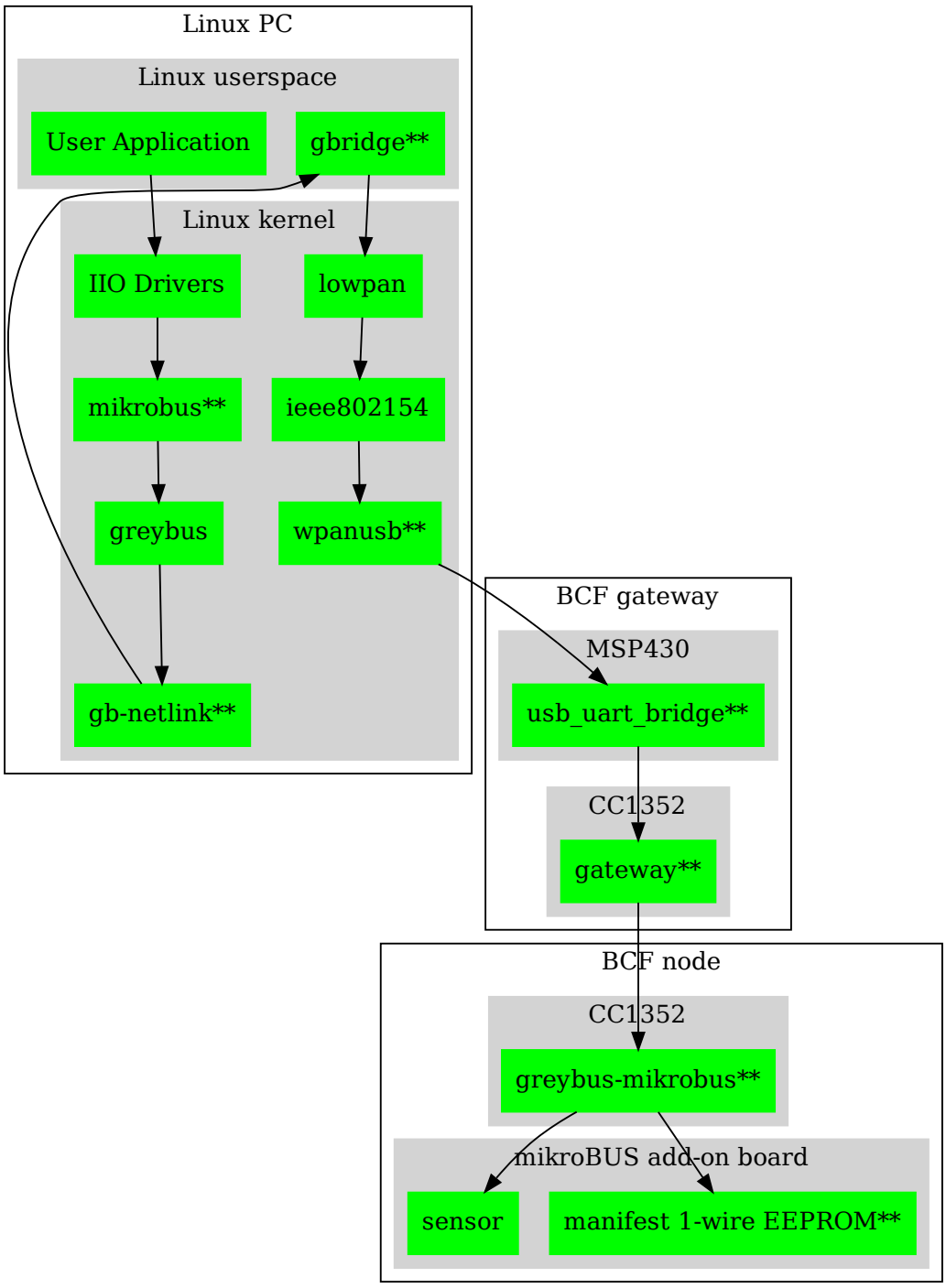


Fig. 14.1: BeagleConnect Software Architecture Diagram

- Click Board plug-ins for node-red for the same 100 or so Click Boards
- BeagleConnect™ Freedom System Reference Manual and FAQs

14.4.5 Associated pre-work

- Click Board support for Node-RED can be executed with native connections on PocketBeagle+TechLab and BeagleBone Black with mikroBUS Cape
- Device tree fragments and driver updates can be provided via https://bbb.io/click
- The Kconfig style provisioning can be implemented for those solutions, which will require a reboot. We need to centralize edits to /boot/uEnv.txt to be programmatic. As I think through this, I don't think BeagleConnect is impacted, because the Greybus-style discovery along with Click EEPROMS will eliminate any need to edit /boot/uEnv.txt.

14.4.6 User experience concerns

- Make sure no reboots are required
- Plugging BeagleConnect into host should trigger host configuration
- Click EEPROMs should trigger loading whatever drivers are needed and provisioning should load any new drivers
- Userspace (spidev, etc.) drivers should unload cleanly when 2nd phase provisioning is completed

14.4.7 BeagleConnect™ Greybus demo using BeagleConnect™ Freedom

BeagleConnect™ Freedom runs a subGHz IEEE 802.15.4 network. This BeagleConnect™ Greybus demo shows how to interact with GPIO, I2C and mikroBUS add-on boards remotely connected over a BeagleConnect™ Freedom.

This section starts with the steps required to use Linux embedded computer (BeagleBone Green Gateway) and the Greybus protocol, over an IEEE 802.15.4 wireless link, to blink an LED on a Zephyr device.

Introduction

Why??

Good question. Blinking an LED is kind of the Hello, World of the hardware community. In this case, we're less interested in the mechanics of switching a GPIO to drive some current through an LED and more interested in how that happens with the Internet of Things (IoT).

There are several existing network and application layers that are driven by corporate heavyweights and industry consortiums, but relatively few that are community driven and, more specifically, even fewer that have the ability to integrate so tightly with the Linux kernel.

The goal here is to provide a community-maintained, developer-friendly, and open-source protocol for the Internet of Things using the Greybus Protocol, and blinking an LED using Greybus is the simplest proof-of-concept for that. All that is required is a reliable transport.

1. Power a BeagleConnect Freedom that has not yet been programmed via a USB power source, not the BeagleBone Green Gateway. You'll hear a click every 1-2 seconds along with seeing 4 of the LEDs turn off and on.
2. In an isolated terminal window, `sudo beagleconnect-start-gateway`
3. `sensortest-rx.py`

Every 1-2 minutes, you should see something like:

```
('fe80::3111:7a22:4b:1200%lowpan0', 52213, 0, 13)  '2l:7.79;'
('fe80::3111:7a22:4b:1200%lowpan0', 52213, 0, 13)  '4h:43.75;4t:23.11;'
```

The value after "2l:" is the amount of light in lux. The value after "4h:" is the relative humidity and after "4t:" is the temperature in Celsius.

Flash BeagleConnect™ Freedom node device with Greybus firmware

#TODO: How can we add a step in here to show the network is connected without needing gbridge to be fully functional?

Do this from the BeagleBone® Green Gateway board that was previously used to program the BeagleConnect™ Freedom gateway device:

1. Disconnect the BeagleConnect™ Freedom **gateway** device
2. Connect a new BeagleConnect™ Freedom board via USB
3. `sudo systemctl stop lowpan.service`
4. `cc2538-bsl.py /usr/share/beagleconnect/cc1352/greybus_mikrobus_beagleconnect.bin /dev/ttyACM0`
5. After it finishes programming successfully, disconnect the BeagleConnect Freedom node device
6. Power the newly programmed BeagleConnect Freedom node device from an alternate USB power source
7. Reconnect the BeagleConnect Freedom **gateway** device to the BeagleBone Green Gateway
8. `sudo systemctl start lowpan.service`
9. `sudo beagleconnect-start-gateway`

```
debian@beaglebone:~$ sudo beagleconnect-start-gateway
[sudo] password for debian:
setting up wpanusb gateway for IEEE 802154 CHANNEL 1(906 Mhz)
ping6: Warning: source address might be selected on device other than␣
↪lowpan0.
PING 2001:db8::1(2001:db8::1) from ::1 lowpan0: 56 data bytes
64 bytes from 2001:db8::1: icmp_seq=2 ttl=64 time=185 ms
64 bytes from 2001:db8::1: icmp_seq=3 ttl=64 time=40.9 ms
64 bytes from 2001:db8::1: icmp_seq=4 ttl=64 time=40.9 ms
64 bytes from 2001:db8::1: icmp_seq=5 ttl=64 time=40.6 ms

--- 2001:db8::1 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 36ms
rtt min/avg/max/mdev = 40.593/76.796/184.799/62.356 ms
debian@beaglebone:~$ iio_info
Library version: 0.19 (git tag: v0.19)
Compiled with backends: local xml ip usb serial
IIO context created with local backend.
Backend version: 0.19 (git tag: v0.19)
Backend description string: Linux beaglebone 5.14.18-bone20 #1buster PREEMPT␣
↪Tue Nov 16 20:47:19 UTC 2021 armv7l
IIO context has 1 attributes:
    local,kernel: 5.14.18-bone20
IIO context has 3 devices:
    iio:device0: TI-am335x-adc.0.auto (buffer capable)
        8 channels found:
            voltage0:  (input, index: 0, format: le:u12/16>>0)
            1 channel-specific attributes found:
                attr  0: raw value: 1412
            voltage1:  (input, index: 1, format: le:u12/16>>0)
            1 channel-specific attributes found:
```

(continues on next page)

(continued from previous page)

```
                attr  0: raw value: 2318
            voltage2:  (input, index: 2, format: le:u12/16>>0)
            1 channel-specific attributes found:
                attr  0: raw value: 2631
            voltage3:  (input, index: 3, format: le:u12/16>>0)
            1 channel-specific attributes found:
                attr  0: raw value: 817
            voltage4:  (input, index: 4, format: le:u12/16>>0)
            1 channel-specific attributes found:
                attr  0: raw value: 881
            voltage5:  (input, index: 5, format: le:u12/16>>0)
            1 channel-specific attributes found:
                attr  0: raw value: 0
            voltage6:  (input, index: 6, format: le:u12/16>>0)
            1 channel-specific attributes found:
                attr  0: raw value: 0
            voltage7:  (input, index: 7, format: le:u12/16>>0)
            1 channel-specific attributes found:
                attr  0: raw value: 1180
        2 buffer-specific attributes found:
                attr  0: data_available value: 0
                attr  1: watermark value: 1
    iio:device1: hdc2010
        3 channels found:
            humidityrelative:  (input)
            3 channel-specific attributes found:
                attr  0: peak_raw value: 52224
                attr  1: raw value: 52234
                attr  2: scale value: 1.525878906
            current:  (output)
            2 channel-specific attributes found:
                attr  0: heater_raw value: 0
                attr  1: heater_raw_available value: 0 1
            temp:  (input)
            4 channel-specific attributes found:
                attr  0: offset value: -15887.515151
                attr  1: peak_raw value: 25600
                attr  2: raw value: 25628
                attr  3: scale value: 2.517700195
    iio:device2: opt3001
        1 channels found:
            illuminance:  (input)
            2 channel-specific attributes found:
                attr  0: input value: 79.040000
                attr  1: integration_time value: 0.800000
        2 device-specific attributes found:
                attr  0: current_timestamp_clock value: realtime

                attr  1: integration_time_available value: 0.1 0.8
debian@beaglebone:~$ dmesg | grep -e mikrobus -e greybus
[  100.491253] greybus 1-2.2: Interface added (greybus)
[  100.491294] greybus 1-2.2: GMP VID=0x00000126, PID=0x00000126
[  100.491306] greybus 1-2.2: DDBL1 Manufacturer=0x00000126,␣
↪Product=0x00000126
[  100.737637] greybus 1-2.2: excess descriptors in interface manifest
[  102.475168] mikrobus:mikrobus_port_gb_register: mikrobus gb_probe , num␣
↪cports= 2, manifest_size 192
[  102.475206] mikrobus:mikrobus_port_gb_register: protocol added 3
[  102.475214] mikrobus:mikrobus_port_gb_register: protocol added 2
[  102.475239] mikrobus:mikrobus_port_register: registering port mikrobus-1
[  102.475400] mikrobus_manifest:mikrobus_state_get: mikrobus descriptor not␣
↪found
```

(continues on next page)

(continued from previous page)

```
[  102.475417] mikrobus_manifest:mikrobus_manifest_attach_device: parsed␣
↪device 1, driver=opt3001, protocol=3, reg=44
[  102.494516] mikrobus_manifest:mikrobus_manifest_attach_device: parsed␣
↪device 2, driver=hdc2010, protocol=3, reg=41
[  102.494567] mikrobus_manifest:mikrobus_manifest_parse:  (null) manifest␣
↪parsed with 2 devices
[  102.494592] mikrobus mikrobus-1: registering device : opt3001
[  102.495096] mikrobus mikrobus-1: registering device : hdc2010
debian@beaglebone:~$
```

#TODO: update the below for the built-in sensors

#TODO: can we also handle the case where these sensors are included and recommend them? Same firmware?

#TODO: the current demo is for the built-in sensors, not the Click boards mentioned below

Currently only a limited number of add-on boards have been tested to work over Greybus, simple add-on boards without interrupt requirement are the ones that work currently. The example is for Air Quality 2 Click and Weather Click attached to the mikroBUS ports on the device side.

/var/log/gbridge will have the gbridge log, and if the mikroBUS port has been instantiated successfully the kernel log will show the devices probe messages:

#TODO: this log needs to be updated

```
greybus 1-2.2: GMP VID=0x00000126, PID=0x00000126
greybus 1-2.2: DDBL1 Manufacturer=0x00000126, Product=0x00000126
greybus 1-2.2: excess descriptors in interface manifest
mikrobus:mikrobus_port_gb_register: mikrobus gb_probe , num cports= 3,␣
↪manifest_size 252
mikrobus:mikrobus_port_gb_register: protocol added 11
mikrobus:mikrobus_port_gb_register: protocol added 3
mikrobus:mikrobus_port_gb_register: protocol added 2
mikrobus:mikrobus_port_register: registering port mikrobus-0
mikrobus_manifest:mikrobus_manifest_attach_device: parsed device 1,␣
↪driver=bme280, protocol=3, reg=76
mikrobus_manifest:mikrobus_manifest_attach_device: parsed device 2,␣
↪driver=ams-iaq-core, protocol=3, reg=5a
mikrobus_manifest:mikrobus_manifest_parse:  Greybus Service Sample␣
↪Application manifest parsed with 2 devices
mikrobus mikrobus-0: registering device : bme280
mikrobus mikrobus-0: registering device : ams-iaq-core
```

#TODO: bring in the GPIO toggle and I2C explorations for greater understanding

Flashing via a Linux Host

If flashing the Freedom board via the BeagleBone fails here's a trick you can try to flash from a Linux host.

Use `sshfs` to mount the Bone's files on the Linux host. This assumes the Bone is plugged in the USB and appears at `192.168.7.2`:

```
host$ cd
host$ sshfs 192.168.7.2:/ bone
host$ cd bone; ls
bin   dev  home    lib         media  opt   root  sbin  sys  usr
boot  etc  ID.txt  lost+found  mnt    proc  run   srv   tmp  var
host$ ls /dev/ttyACM*
/dev/ttyACM1
```

The Bone's files now appear as local files. Notice there is already a `/dev/ACM*` appearing. Now plug the Connect into the Linux host's USB port and run the command again.

```
host$ ls /dev/ttyACM*
/dev/ttyACM0  /dev/ttyACM1
```

The `/dev/ttyACM` that just appeared is the one associated with the Connect. In my case it's `/dev/ttyACM0`. That's what I'll use in this example.

Now change directories to where the binaries are and load:

```
host$ cd ~/bone/usr/share/beagleconnect/cc1352;ls
greybus_mikrobus_beagleconnect.bin     sensortest_beagleconnect.dts
greybus_mikrobus_beagleconnect.config  wpanusb_beagleconnect.bin
greybus_mikrobus_beagleconnect.dts     wpanusb_beagleconnect.config
sensortest_beagleconnect.bin           wpanusb_beagleconnect.dts
sensortest_beagleconnect.config

host$ ~/bone/usr/bin/cc2538-bsl.py sensortest_beagleconnect.bin /dev/ttyACM0
8-bsl.py sensortest_beagleconnect.bin /dev/ttyACM0
Opening port /dev/ttyACM0, baud 50000
Reading data from sensortest_beagleconnect.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
CC1350 PG2.0 (7x7mm): 352KB Flash, 20KB SRAM, CCFG.BL_CONFIG at 0x00057FD8
Primary IEEE Address: 00:12:4B:00:22:7A:10:46
    Performing mass erase
Erasing all main bank flash sectors
    Erase done
Writing 360448 bytes starting at address 0x00000000
Write 104 bytes at 0x00057F988
    Write done
Verifying by comparing CRC32 calculations.
    Verified (match: 0x0f6bdf0f)
```

Now you are ready to continue the instructions above after the cc2528 command.

Trying for different add-on boards See mikroBUS over Greybus for trying out the same example for different mikroBUS add-on boards/ on-board devices.

Observe the node device

Connect BeagleConnect Freedom node device to an Ubuntu laptop to observe the Zephyr console.

Console (`tio`)

In order to see diagnostic messages or to run certain commands on the Zephyr device we will require a terminal open to the device console. In this case, we use tio due how its usage simplifies the instructions.

1. Install `tio` `sudo apt install -y tio`
2. Run `tio` `tio /dev/ttyACM0`

   To exit `tio` (later), enter `ctrl+t, q`.

The Zephyr Shell

After flashing, you should observe the something matching the following output in `tio`.

```
uart:~$ *** Booting Zephyr OS build 9c858c863223  ***
[00:00:00.009,735] <inf> greybus_transport_tcpip: CPort 0 mapped to TCP/IP␣
↪port 4242
[00:00:00.010,131] <inf> greybus_transport_tcpip: CPort 1 mapped to TCP/IP␣
↪port 4243
[00:00:00.010,528] <inf> greybus_transport_tcpip: CPort 2 mapped to TCP/IP␣
↪port 4244
[00:00:00.010,742] <inf> greybus_transport_tcpip: Greybus TCP/IP Transport␣
↪initialized
[00:00:00.010,864] <inf> greybus_manifest: Registering CONTROL greybus␣
↪driver.
[00:00:00.011,230] <inf> greybus_manifest: Registering GPIO greybus driver.
[00:00:00.011,596] <inf> greybus_manifest: Registering I2C greybus driver.
[00:00:00.011,871] <inf> greybus_service: Greybus is active
[00:00:00.026,092] <inf> net_config: Initializing network
[00:00:00.134,063] <inf> net_config: IPv6 address: 2001:db8::1
```

The line beginning with `***` is the Zephyr boot banner.

Lines beginning with a timestamp of the form `[H:m:s.us]` are Zephyr kernel messages.

Lines beginning with `uart:~$` indicates that the Zephyr shell is prompting you to enter a command.

From the informational messages shown, we observe the following.

- Zephyr is configured with the following link-local IPv6 address `fe80::3177:a11c:4b:1200`
- It is listening for (both) TCP and UDP traffic on port 4242

However, what the log messages do not show (which will come into play later), are 2 critical pieces of information:

1. **The RF Channel**: As you may have guessed, IEEE 802.15.4 devices are only able to communicate with each other if they are using the same frequency to transmit and receive data. This information is part of the Physical Layer.
2. The PAN identifier: IEEE 802.15.4 devices are only be able to communicate with one another if they use the same PAN ID. This permits multiple networks (PANs) on the same frequency. This information is part of the Data Link Layer.

If we type `help` in the shell and hit Enter, we're prompted with the following:

```
Please press the <Tab> button to see all available commands.
You can also use the <Tab> button to prompt or auto-complete all commands or␣
↪its subcommands.
You can try to call commands with <-h> or <--help> parameter for more␣
↪information.
Shell supports following meta-keys:

Ctrl+a, Ctrl+b, Ctrl+c, Ctrl+d, Ctrl+e, Ctrl+f, Ctrl+k, Ctrl+l, Ctrl+n,␣
↪Ctrl+p, Ctrl+u, Ctrl+w
Alt+b, Alt+f.
Please refer to shell documentation for more details.
```

So after hitting Tab, we see that there are several interesting commands we can use for additional information.

```
uart:~$
clear       help        history     ieee802154  log         net
resize      sample      shell
```

Zephyr Shell: IEEE 802.15.4 commands Entering `ieee802154 help`, we see

```
uart:~$ ieee802154 help
ieee802154 - IEEE 802.15.4 commands
Subcommands:
ack             :<set/1 | unset/0> Set auto-ack flag
associate       :<pan_id> <PAN coordinator short or long address (EUI-64)>
disassociate    :Disassociate from network
get_chan        :Get currently used channel
get_ext_addr    :Get currently used extended address
get_pan_id      :Get currently used PAN id
get_short_addr  :Get currently used short address
get_tx_power    :Get currently used TX power
scan            :<passive|active> <channels set n[:m:...]:x|all> <per-channel
                duration in ms>
set_chan        :<channel> Set used channel
set_ext_addr    :<long/extended address (EUI-64)> Set extended address
set_pan_id      :<pan_id> Set used PAN id
set_short_addr  :<short address> Set short address
set_tx_power    :<-18/-7/-4/-2/0/1/2/3/5> Set TX power
```

We get the missing Channel number (frequency) with the command `ieee802154 get_chan`.

```
uart:~$ ieee802154 get_chan
Channel 26
```

We get the missing PAN ID with the command `ieee802154 get_pan_id`.

```
uart:~$ ieee802154 get_pan_id
PAN ID 43981 (0xabcd)
```

Zephyr Shell: Network Commands Additionally, we may query the IPv6 information of the Zephyr device.

```
uart:~$ net iface

Interface 0x20002b20 (IEEE 802.15.4) [1]
========================================
Link addr : CD:99:A1:1C:00:4B:12:00
MTU       : 125
IPv6 unicast addresses (max 3):
        fe80::cf99:a11c:4b:1200 autoconf preferred infinite
        2001:db8::1 manual preferred infinite
IPv6 multicast addresses (max 4):
        ff02::1
        ff02::1:ff4b:1200
        ff02::1:ff00:1
IPv6 prefixes (max 2):
        <none>
IPv6 hop limit           : 64
IPv6 base reachable time : 30000
IPv6 reachable time      : 16929
IPv6 retransmit timer    : 0
```

And we see that the static IPv6 address (`2001:db8::1`) from `samples/net/sockets/echo_server/prj.conf` is present and configured. While the statically configured IPv6 address is useful, it isn't 100% necessary.

Rebuilding from source

#TODO: revisit everything below here

Prerequisites

- Zephyr environment is set up according to the Getting Started Guide
  - Please use the Zephyr SDK when installing a toolchain above
- Zephyr SDK is installed at ~/zephyr-sdk-0.11.2 (any later version should be fine as well)
- Zephyr board is connected via USB

Cloning the repository This repository utilizes git submodules to keep track of all of the projects required to reproduce the ongoing work. The instructions here only cover checking out the `demo` branch which should stay in a tested state. ongoing development will be on the `master` branch.

Note: The parent directory ~ is simply used as a placeholder for testing. Please use whatever parent directory you see fit.

Clone specific tag

```
cd ~
git clone --recurse-submodules --branch demo https://github.com/jadonk/
↪beagleconnect
```

Zephyr

Add the Fork For the time being, Greybus must remain outside of the main Zephyr repository. Currently, it is just in a Zephyr fork, but it should be converted to a proper Module (External Project). This is for a number of reasons, but mainly there must be:

- specifications for authentication and encryption
- specifications for joining and rejoining wireless networks
- specifications for discovery

Therefore, in order to reproduce this example, please run the following.

```
cd ~/beagleconnect/sw/zephyrproject/zephyr
west update
```

Build and Flash Zephyr Here, we will build and flash the Zephyr greybus_net sample to our device.

1. Edit the file `~/.zephyrrc` and place the following text inside of it

```
export ZEPHYR_TOOLCHAIN_VARIANT=zephyr
export ZEPHYR_SDK_INSTALL_DIR=~/zephyr-sdk-0.11.2
```

1. Set up the required Zephyr environment variables via

```
source zephyr-env.sh
```

1. Build the project

```
BOARD=cc1352r1_launchxl west build samples/subsys/greybus/net --pristine \
--build-dir build/greybus_launchpad -- -DCONF_FILE="prj.conf overlay-802154.
↪conf"
```

1. Ensure that the last part of the build process looks somewhat like this:

```
...
[221/226] Linking C executable zephyr/zephyr_prebuilt.elf
Memory region         Used Size  Region Size  %age Used
        FLASH:      155760 B     360360 B     43.22%
    FLASH_CCFG:          88 B         88 B    100.00%
            SRAM:       58496 B         80 KB     71.41%
        IDT_LIST:         184 B          2 KB      8.98%
[226/226] Linking C executable zephyr/zephyr.elf
```

1. Flash the firmware to your device using

```
BOARD=cc1352r1_launchxl west flash --build-dir build/greybus_launchpad
```

Linux

Warning: If you aren't comfortable building and installing a Linux kernel on your computer, you should probably just stop here. I'll assume you know the basics of building and installing a Linux kernel from here on out.

Clone, patch, and build the kernel For this demo, I used the 5.8.4 stable kernel. Also, I've applied the `mikrobus` kernel driver, though it isn't strictly required for greybus.

Note: The parent directory ~ is simply used as a placeholder for testing. Please use whatever parent directory you see fit.

TODO: The patches for gb-netlink will eventually be applied here until pushed into mainline.

```
cd ~
git clone --branch v5.8.4 --single-branch git://git.kernel.org/pub/scm/linux/
↪kernel/git/stable/linux.git
cd linux
git checkout -b v5.8.4-greybus
git am ~/beagleconnect/sw/linux/v2-0001-RFC-mikroBUS-driver-for-add-on-
↪boards.patch
git am ~/beagleconnect/sw/linux/0001-mikroBUS-build-fixes.patch
cp /boot/config-`uname -r` .config
yes "" | make oldconfig
./scripts/kconfig/merge_config.sh .config ~/beagleconnect/sw/linux/mikrobus.
↪config
./scripts/kconfig/merge_config.sh .config ~/beagleconnect/sw/linux/atusb.
↪config
make -j`nproc --all`
sudo make modules_install
sudo make install
```

Reboot and select your new kernel.

Probe the IEEE 802.15.4 Device Driver On the Linux machine, make sure the `atusb` driver is loaded. This should happen automatically when the adapter is inserted or when the machine is booted while the adapter is installed.

```
$ dmesg | grep -i ATUSB
[    6.512154] usb 1-1: ATUSB: AT86RF231 version 2
[    6.512492] usb 1-1: Firmware: major: 0, minor: 3, hardware type: ATUSB␣
↪(2)
[    6.525357] usbcore: registered new interface driver atusb
...
```

We should now be able to see the IEEE 802.15.4 network device by entering `ip a show wpan0`.

```
$ ip a show wpan0
36: wpan0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 123 qdisc fq_codel state␣
↪UNKNOWN group default qlen 300
    link/ieee802.15.4 3e:7d:90:4d:8f:00:76:a2 brd ff:ff:ff:ff:ff:ff:ff:ff
```

But wait, that is not an IP address! It's the hardware address of the 802.15.4 device. So, in order to associate it with an IP address, we need to run a couple of other commands (thanks to wpan.cakelab.org).

Set the 802.15.4 Physical and Link-Layer Parameters

1. First, get the phy number for the `wpan0` device

```
$ iwpan list
    wpan_phy phy0
    supported channels:
        page 0: 11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
    current_page: 0
    current_channel: 26,  2480 MHz
    cca_mode: (1) Energy above threshold
    cca_ed_level: -77
    tx_power: 3
    capabilities:
        iftypes: node,monitor
        channels:
            page 0:
                [11]  2405 MHz, [12]  2410 MHz, [13]  2415 MHz,
                [14]  2420 MHz, [15]  2425 MHz, [16]  2430 MHz,
                [17]  2435 MHz, [18]  2440 MHz, [19]  2445 MHz,
                [20]  2450 MHz, [21]  2455 MHz, [22]  2460 MHz,
                [23]  2465 MHz, [24]  2470 MHz, [25]  2475 MHz,
                [26]  2480 MHz
        tx_powers:
                3 dBm, 2.8 dBm, 2.3 dBm, 1.8 dBm, 1.3 dBm, 0.7 dBm,
                0 dBm, -1 dBm, -2 dBm, -3 dBm, -4 dBm, -5 dBm,
                -7 dBm, -9 dBm, -12 dBm, -17 dBm,
        cca_ed_levels:
                -91 dBm, -89 dBm, -87 dBm, -85 dBm, -83 dBm, -81 dBm,
                -79 dBm, -77 dBm, -75 dBm, -73 dBm, -71 dBm, -69 dBm,
                -67 dBm, -65 dBm, -63 dBm, -61 dBm,
        cca_modes:
            (1) Energy above threshold
            (2) Carrier sense only
            (3, cca_opt: 0) Carrier sense with energy above threshold␣
↪(logical operator is 'and')
            (3, cca_opt: 1) Carrier sense with energy above threshold␣
↪(logical operator is 'or')
        min_be: 0,1,2,3,4,5,6,7,8
        max_be: 3,4,5,6,7,8
        csma_backoffs: 0,1,2,3,4,5
        frame_retries: 3
        lbt: false
```

1. Next, set the Channel for the 802.15.4 device on the Linux machine

```
sudo iwpan phy phy0 set channel 0 26
```

1. Then, set the PAN identifier for the 802.15.4 device on the Linux machine `sudo iwpan dev wpan0 set pan_id 0xabcd`
2. Associate the `wpan0` device to a new, 6lowpan network interface

```
sudo ip link add link wpan0 name lowpan0 type lowpan
```

1. Finally, set the links up for both `wpan0` and `lowpan0`

```
sudo ip link set wpan0 up
sudo ip link set lowpan0 up
```

We should observe something like the following when we run `ip a show lowpan0`.

```
ip a show lowpan0
37: lowpan0@wpan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1280 qdisc noqueue␣
↪state UNKNOWN group default qlen 1000
    link/6lowpan 9e:0b:a4:e8:00:d3:45:53 brd ff:ff:ff:ff:ff:ff:ff:ff
    inet6 fe80::9c0b:a4e8:d3:4553/64 scope link
    valid_lft forever preferred_lft forever
```

Ping Pong

Broadcast Ping Now, perform a broadcast ping to see what else is listening on `lowpan0`.

```
$ ping6 -I lowpan0 ff02::1
PING ff02::1(ff02::1) from fe80::9c0b:a4e8:d3:4553%lowpan0 lowpan0: 56 data␣
↪bytes
64 bytes from fe80::9c0b:a4e8:d3:4553%lowpan0: icmp_seq=1 ttl=64 time=0.099␣
↪ms
64 bytes from fe80::9c0b:a4e8:d3:4553%lowpan0: icmp_seq=2 ttl=64 time=0.125␣
↪ms
64 bytes from fe80::cf99:a11c:4b:1200%lowpan0: icmp_seq=2 ttl=64 time=17.3␣
↪ms (DUP!)
64 bytes from fe80::9c0b:a4e8:d3:4553%lowpan0: icmp_seq=3 ttl=64 time=0.126␣
↪ms
64 bytes from fe80::cf99:a11c:4b:1200%lowpan0: icmp_seq=3 ttl=64 time=9.60␣
↪ms (DUP!)
64 bytes from fe80::9c0b:a4e8:d3:4553%lowpan0: icmp_seq=4 ttl=64 time=0.131␣
↪ms
64 bytes from fe80::cf99:a11c:4b:1200%lowpan0: icmp_seq=4 ttl=64 time=14.9␣
↪ms (DUP!)
```

Yay! We have pinged (pung?) the Zephyr device over IEEE 802.15.4 using 6LowPAN!

Ping Zephyr We can ping the Zephyr device directly without a broadcast ping too, of course.

```
$ ping6 -I lowpan0 fe80::cf99:a11c:4b:1200
PING fe80::cf99:a11c:4b:1200(fe80::cf99:a11c:4b:1200) from␣
↪fe80::9c0b:a4e8:d3:4553%lowpan0 lowpan0: 56 data bytes
64 bytes from fe80::cf99:a11c:4b:1200%lowpan0: icmp_seq=1 ttl=64 time=16.0 ms
64 bytes from fe80::cf99:a11c:4b:1200%lowpan0: icmp_seq=2 ttl=64 time=13.8 ms
64 bytes from fe80::cf99:a11c:4b:1200%lowpan0: icmp_seq=3 ttl=64 time=9.77 ms
64 bytes from fe80::cf99:a11c:4b:1200%lowpan0: icmp_seq=5 ttl=64 time=11.5 ms
```

Ping Linux Similarly, we can ping the Linux host from the Zephyr shell.

```
uart:~$ net ping --help
ping - Ping a network host.
Subcommands:
--help  :'net ping [-c count] [-i interval ms] <host>' Send ICMPv4 or ICMPv6
        Echo-Request to a network host.
$ net ping -c 5 fe80::9c0b:a4e8:d3:4553
```

(continues on next page)

(continued from previous page)

```
PING fe80::9c0b:a4e8:d3:4553
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::cf99:a11c:4b:1200: icmp_seq=0␣
↪ttl=64 rssi=110 time=11 ms
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::cf99:a11c:4b:1200: icmp_seq=1␣
↪ttl=64 rssi=126 time=9 ms
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::cf99:a11c:4b:1200: icmp_seq=2␣
↪ttl=64 rssi=128 time=13 ms
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::cf99:a11c:4b:1200: icmp_seq=3␣
↪ttl=64 rssi=126 time=10 ms
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::cf99:a11c:4b:1200: icmp_seq=4␣
↪ttl=64 rssi=126 time=7 ms
```

Assign a Static Address So far, we have been using IPv6 Link-Local addressing. However, the Zephyr application is configured to use a statically configured IPv6 address as well which is, namely `2001:db8::1`.

If we add a similar static IPv6 address to our Linux IEEE 802.15.4 network interface, `lowpan0`, then we should expect to be able to reach that as well.

In Linux, run the following

```
sudo ip -6 addr add 2001:db8::2/64 dev lowpan0
```

We can verify that the address has been set by examining the `lowpan0` network interface again.

```
$ ip a show lowpan0
37: lowpan0@wpan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1280 qdisc noqueue␣
↪state UNKNOWN group default qlen 1000
    link/6lowpan 9e:0b:a4:e8:00:d3:45:53 brd ff:ff:ff:ff:ff:ff:ff:ff
    inet6 2001:db8::2/64 scope global
    valid_lft forever preferred_lft forever
    inet6 fe80::9c0b:a4e8:d3:4553/64 scope link
    valid_lft forever preferred_lft forever
```

Lastly, ping the statically configured IPv6 address of the Zephyr device.

```
$ ping6 2001:db8::1
PING 2001:db8::1(2001:db8::1) 56 data bytes
64 bytes from 2001:db8::1: icmp_seq=2 ttl=64 time=53.7 ms
64 bytes from 2001:db8::1: icmp_seq=3 ttl=64 time=13.1 ms
64 bytes from 2001:db8::1: icmp_seq=4 ttl=64 time=22.0 ms
64 bytes from 2001:db8::1: icmp_seq=5 ttl=64 time=22.7 ms
64 bytes from 2001:db8::1: icmp_seq=6 ttl=64 time=18.4 ms
```

Now that we have set up a reliable transport, let's move on to the application layer.

Greybus

Hopefully the videos listed earlier provide a sufficient foundation to understand what will happen shortly. However, there is still a bit more preparation required.

Build and probe Greybus Kernel Modules Greybus was originally intended to work exclusively on the UniPro physical layer. However, we're using RF as our physical layer and TCP/IP as our transport. As such, there was need to be able to communicate with the Linux Greybus facilities through userspace, and out of that need arose gb-netlink. The Netlink Greybus module actually does not care about the physical layer, but is happy to usher Greybus messages back and forth between the kernel and userspace.

Build and probe the gb-netlink modules (as well as the other Greybus modules) with the following:

```
cd ${WORKSPACE}/sw/greybus
make -j`nproc --all`
sudo make install
../load_gb_modules.sh
```

Build and Run Gbridge The gbridge utility was created as a proof of concept to abstract the Greybus Netlink datapath among several reliable transports. For the purposes of this tutorial, we'll be using it as a TCP/IP bridge.

To run `gbridge`, perform the following:

```
sudo apt install -y libnl-3-dev libnl-genl-3-dev libbluetooth-dev libavahi-
↪client-dev
cd gbridge
autoreconf -vfi
GBNETLINKDIR=${PWD}/../greybus \
./configure --enable-uart --enable-tcpip --disable-gbsim --enable-netlink --
↪disable-bluetooth
make -j`nproc --all`
sudo make install
gbridge
```

Blinky!

Now that we have set up a reliable TCP transport, and set up the Greybus modules in the Linux kernel, and used Gbridge to connect a Greybus node to the Linux kernel via TCP/IP, we can now get to the heart of the demonstration!

First, save the following script as `blinky.sh`.

```
#!/bin/bash

# Blinky Demo for CC1352R SensorTag

# /dev/gpiochipN that Greybus created
CHIP="$(gpiodetect | grep greybus_gpio | head -n 1 | awk '{print $1}')"

# red, green, blue LED pins
RED=6
GREEN=7
BLUE=21

# Bash array for pins and values
PINS=($RED $GREEN $BLUE)
NPINS=${#PINS[@]}

for ((;;)); do
    for i in ${!PINS[@]}; do
        # turn off previous pin
        if [ $i -eq 0 ]; then
            PREV=2
        else
            PREV=$((i-1))
        fi
        gpioset $CHIP ${PINS[$PREV]}=0

        # turn on current pin
        gpioset $CHIP ${PINS[$i]}=1

        # wait a sec
        sleep 1
```

(continues on next page)

(continued from previous page)

```
    done
done
```

Second, run the script with root privileges: `sudo bash blinky.sh`

The output of your minicom session should resemble the following.

```
$ *** Booting Zephyr OS build zephyr-v2.3.0-1435-g40c0ed940d71  ***
[00:00:00.011,932] <inf> net_config: Initializing network
[00:00:00.111,938] <inf> net_config: IPv6 address: fe80::6c42:bc1c:4b:1200
[00:00:00.112,121] <dbg> greybus_service.greybus_service_init: Greybus␣
↪initializing..
[00:00:00.112,426] <dbg> greybus_transport_tcpip.gb_transport_backend_init:␣
↪Greybus TCP/IP Transport initializing..
[00:00:00.112,579] <dbg> greybus_transport_tcpip.netsetup: created server␣
↪socket 0 for cport 0
[00:00:00.112,579] <dbg> greybus_transport_tcpip.netsetup: setting socket␣
↪options for socket 0
[00:00:00.112,609] <dbg> greybus_transport_tcpip.netsetup: binding socket 0␣
↪(cport 0) to port 4242
[00:00:00.112,640] <dbg> greybus_transport_tcpip.netsetup: listening on␣
↪socket 0 (cport 0)
[00:00:00.112,823] <dbg> greybus_transport_tcpip.netsetup: created server␣
↪socket 1 for cport 1
[00:00:00.112,823] <dbg> greybus_transport_tcpip.netsetup: setting socket␣
↪options for socket 1
[00:00:00.112,854] <dbg> greybus_transport_tcpip.netsetup: binding socket 1␣
↪(cport 1) to port 4243
[00:00:00.112,854] <dbg> greybus_transport_tcpip.netsetup: listening on␣
↪socket 1 (cport 1)
[00:00:00.113,037] <inf> net_config: IPv6 address: fe80::6c42:bc1c:4b:1200
[00:00:00.113,250] <dbg> greybus_transport_tcpip.netsetup: created server␣
↪socket 2 for cport 2
[00:00:00.113,250] <dbg> greybus_transport_tcpip.netsetup: setting socket␣
↪options for socket 2
[00:00:00.113,281] <dbg> greybus_transport_tcpip.netsetup: binding socket 2␣
↪(cport 2) to port 4244
[00:00:00.113,311] <dbg> greybus_transport_tcpip.netsetup: listening on␣
↪socket 2 (cport 2)
[00:00:00.113,494] <dbg> greybus_transport_tcpip.netsetup: created server␣
↪socket 3 for cport 3
[00:00:00.113,494] <dbg> greybus_transport_tcpip.netsetup: setting socket␣
↪options for socket 3
[00:00:00.113,525] <dbg> greybus_transport_tcpip.netsetup: binding socket 3␣
↪(cport 3) to port 4245
[00:00:00.113,555] <dbg> greybus_transport_tcpip.netsetup: listening on␣
↪socket 3 (cport 3)
[00:00:00.113,861] <inf> greybus_transport_tcpip: Greybus TCP/IP Transport␣
↪initialized
[00:00:00.116,149] <inf> greybus_service: Greybus is active
[00:00:00.116,546] <dbg> greybus_transport_tcpip.accept_loop: calling poll
[00:45:08.397,399] <dbg> greybus_transport_tcpip.accept_loop: poll returned 1
[00:45:08.397,399] <dbg> greybus_transport_tcpip.accept_loop: socket 0␣
↪(cport 0) has traffic
[00:45:08.397,491] <dbg> greybus_transport_tcpip.accept_loop: accepted␣
↪connection from [2001:db8::2]:39638 as fd 4
[00:45:08.397,491] <dbg> greybus_transport_tcpip.accept_loop: spawning␣
↪client thread..
[00:45:08.397,735] <dbg> greybus_transport_tcpip.accept_loop: calling poll
[00:45:08.491,363] <dbg> greybus_transport_tcpip.accept_loop: poll returned 1
[00:45:08.491,363] <dbg> greybus_transport_tcpip.accept_loop: socket 3␣
↪(cport 3) has traffic
```

(continues on next page)

(continued from previous page)

```
[00:45:08.491,455] <dbg> greybus_transport_tcpip.accept_loop: accepted␣
↪connection from [2001:db8::2]:39890 as fd 5
[00:45:08.491,455] <dbg> greybus_transport_tcpip.accept_loop: spawning␣
↪client thread..
[00:45:08.491,699] <dbg> greybus_transport_tcpip.accept_loop: calling poll
[00:45:08.620,056] <dbg> greybus_transport_tcpip.accept_loop: poll returned 1
[00:45:08.620,086] <dbg> greybus_transport_tcpip.accept_loop: socket 2␣
↪(cport 2) has traffic
[00:45:08.620,147] <dbg> greybus_transport_tcpip.accept_loop: accepted␣
↪connection from [2001:db8::2]:42422 as fd 6
[00:45:08.620,147] <dbg> greybus_transport_tcpip.accept_loop: spawning␣
↪client thread..
[00:45:08.620,422] <dbg> greybus_transport_tcpip.accept_loop: calling poll
[00:45:08.679,504] <dbg> greybus_transport_tcpip.accept_loop: poll returned 1
[00:45:08.679,534] <dbg> greybus_transport_tcpip.accept_loop: socket 1␣
↪(cport 1) has traffic
[00:45:08.679,595] <dbg> greybus_transport_tcpip.accept_loop: accepted␣
↪connection from [2001:db8::2]:48286 as fd 7
[00:45:08.679,595] <dbg> greybus_transport_tcpip.accept_loop: spawning␣
↪client thread..
[00:45:08.679,870] <dbg> greybus_transport_tcpip.accept_loop: calling poll
...
```

Read I2C Registers The SensorTag comes with an opt3001 ambient light sensor as well as an hdc2080 temperature & humidity sensor.

First, find which i2c device corresponds to the SensorTag:

```
ls -la /sys/bus/i2c/devices/* | grep "greybus"
lrwxrwxrwx 1 root root 0 Aug 15 11:24 /sys/bus/i2c/devices/i2c-8 -> ../../../
↪devices/virtual/gb_nl/gn_nl/greybus1/1-2/1-2.2/1-2.2.2/gbphy2/i2c-8
```

On my machine, the i2c device node that Greybus creates is `/dev/i2c-8`.

Read the ID registers (at the i2c register address 0x7e) of the opt3001 sensor (at i2c bus address 0x44) as shown below:

```
i2cget -y 8 0x44 0x7e w
0x4954
```

Read the ID registers (at the i2c register address 0xfc) of the hdc2080 sensor (at i2c bus address 0x41) as shown below:

```
i2cget -y 8 0x41 0xfc w
0x5449
```

Conclusion

The blinking LED can and poking i2c registers can be a somewhat anticlimactic, but hopefully it illustrates the potential for Greybus as an IoT application layer protocol.

What is nice about this demo, is that we're using Device Tree to describe our Greybus Peripheral declaratively, they Greybus Manifest is automatically generated, and the Greybus Service is automatically started in Zephyr.

In other words, all that is required to replicate this for other IoT devices is simply an appropriate Device Tree overlay file.

The proof-of-concept involving Linux, Zephyr, and IEEE 802.15.4 was actually fairly straight forward and was accomplished with mostly already-upstream source.

For Greybus in Zephyr, there is still a considerable amount of integration work to be done, including * converting the fork to a proper Zephyr module * adding security and authentication * automatic detection, joining, and rejoining of devices.

Thanks for reading, and we hope you've enjoyed this tutorial.

Chapter 15

Books

This is a collection of open-source books written to help Beagle developers.

BeagleBone Cookbook is a great introduction to programming a BeagleBone using Linux from userspace, mostly using Python or JavaScript.

PRU Cookbook provides numerous examples on using the incredible ultra-low-latency microcontrollers inside the processors used on BeagleBone boards that are a big part of what has made BeagleBone such a popular platform.

Links to additional books available for purchase can be found on the Beagle books page.

15.1 BeagleBone Cookbook

Contributors

- Author: Mark A. Yoder
- Book revision: v2.1 beta

A cookbook for programming Beagles

15.1.1 Basics

When you buy BeagleBone Black, pretty much everything you need to get going comes with it. You can just plug it into the USB of a host computer, and it works. The goal of this chapter is to show what you can do with your Bone, right out of the box. It has enough information to carry through the next three chapters on sensors (*Sensors*), displays (*Displays and Other Outputs*), and motors (*Motors*).

Picking Your Beagle

Problem There are many different BeagleBoards. How do you pick which one to use?

Solution Current list of boards: https://git.beagleboard.org/explore/projects/topics/boards

Discussion

Getting Started, Out of the Box

Problem You just got your Bone, and you want to know what to do with it.

Solution Fortunately, you have all you need to get running: your Bone and a USB cable. Plug the USB cable into your host computer (Mac, Windows, or Linux) and plug the mini-USB connector side into the USB connector near the Ethernet connector on the Bone, as shown in *Plugging BeagleBone Black into a USB port*.

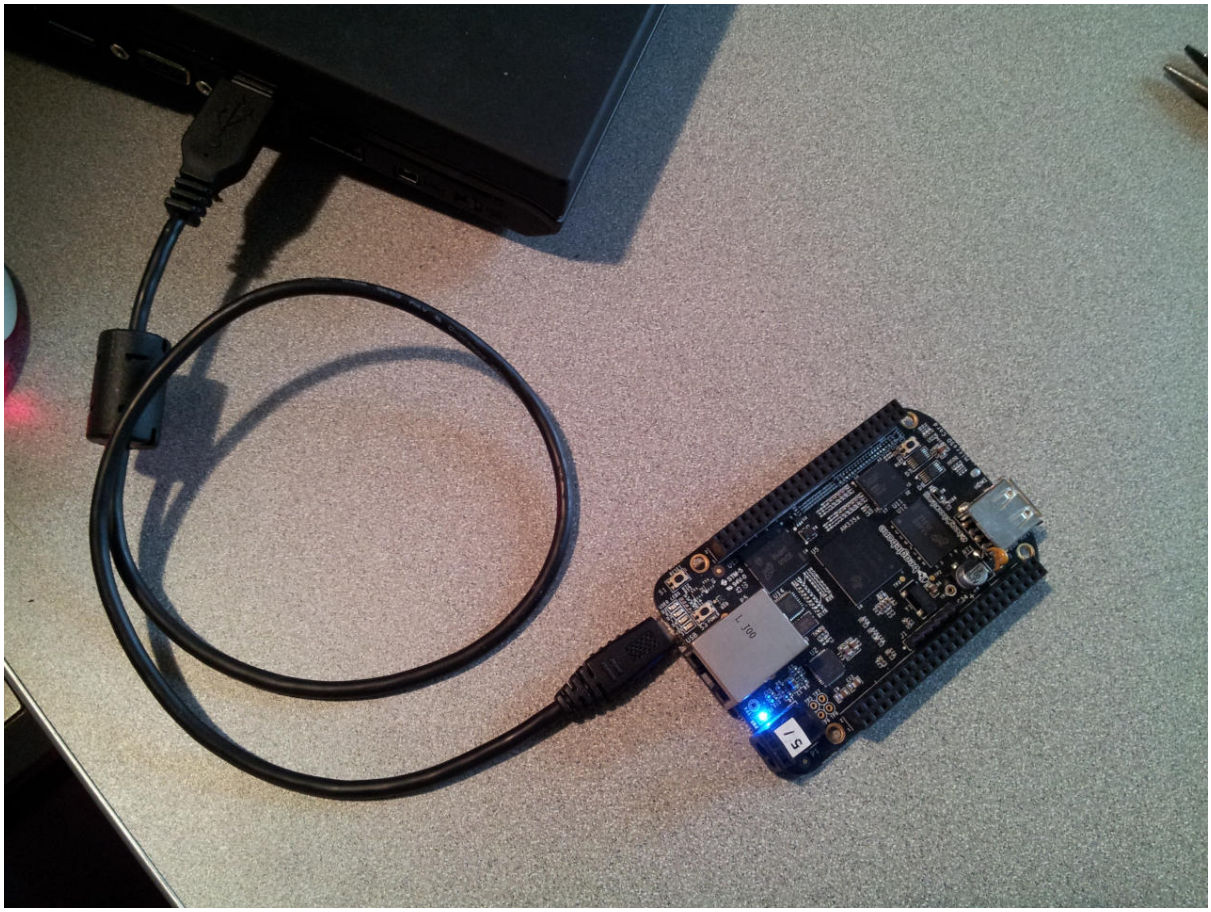


Fig. 15.1: Plugging BeagleBone Black into a USB port

The four blue **USER LEDs** will begin to blink, and in 10 or 15 seconds, you'll see a new USB drive appear on your host computer. *The Bone appears as a USB drive* shows how it will appear on a Windows host, and Linux and Mac hosts will look similar. The Bone acting like a USB drive and the files you see are located on the Bone.

Browse to http://192.168.7.2:3000 from your host computer (*Visual Studio Code*). If the page is not found, run the following:

```
bone$ sudo systemctl start bb-code-server.service
```

Wait a minute and try the URL again.

Here, you'll find *Visual Studio Code*, a web-based integrated development environment (IDE) that lets you edit and run code on your Bone! See :ref: *basics_vsc* for more details.

Warning:

Make sure you turn off your Bone properly. It's best to run the *halt* command:

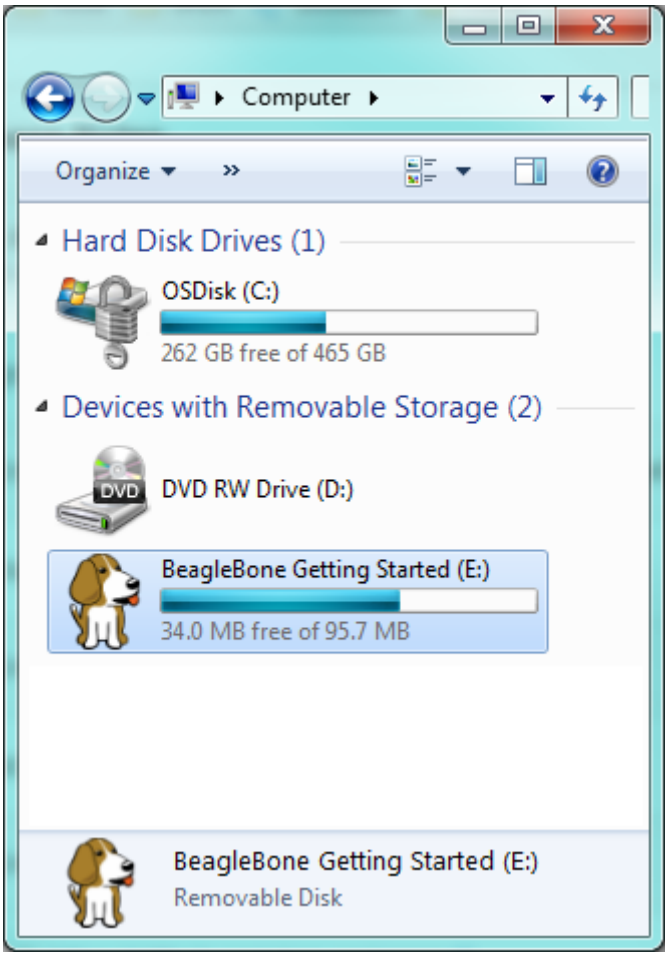


Fig. 15.2: The Bone appears as a USB drive

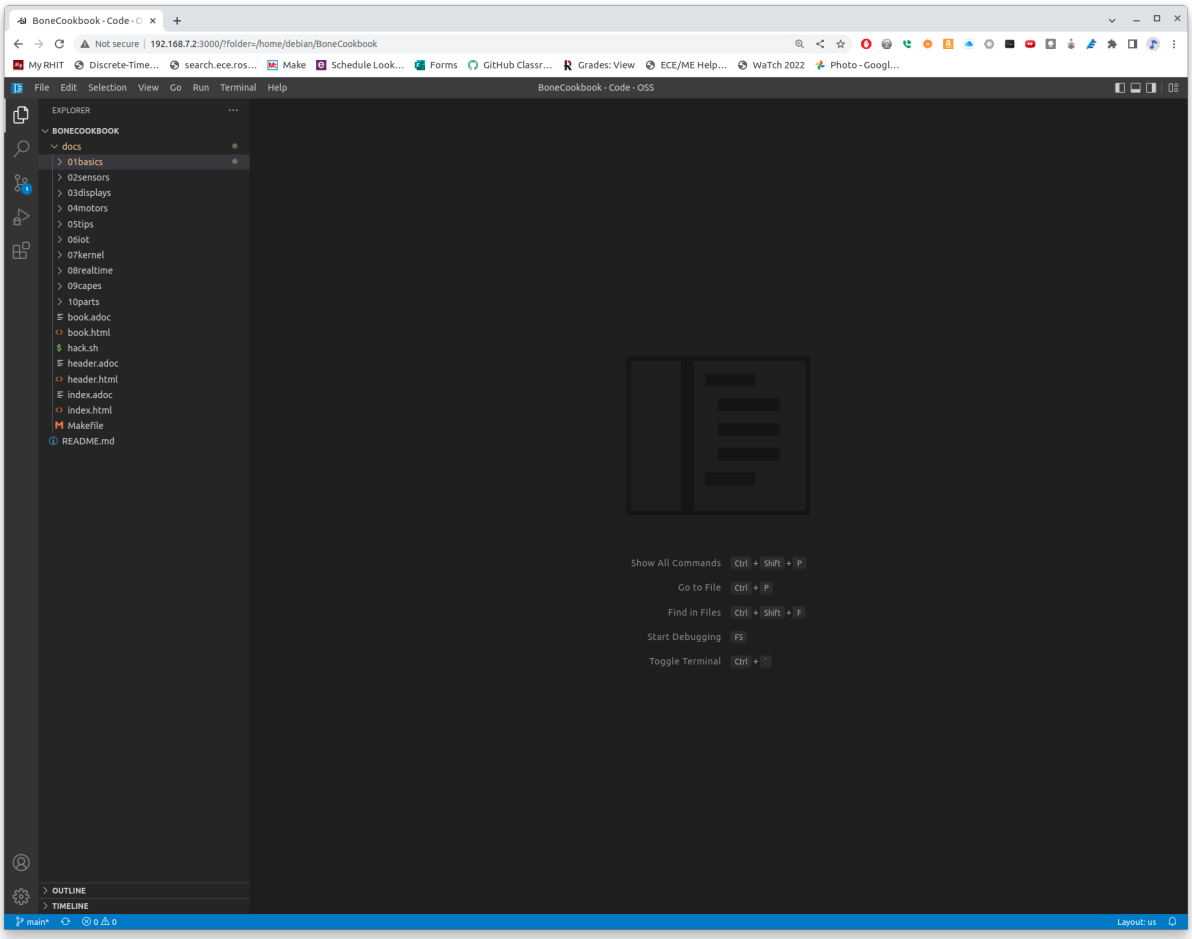


Fig. 15.3: Visual Studio Code

```
bone$ sudo halt

The system is going down for system halt NOW! (pts/0)
```

This will ensure that the Bone shuts down correctly. If you just pull the power, it is possible that open files would not close properly and might become corrupt.

Discussion The rest of this book goes into the details behind this quick out-of-the-box demo. Explore your Bone and then start exploring the book.

Verifying You Have the Latest Version of the OS on Your Bone

Problem You just got BeagleBone Black, and you want to know which version of the operating system it's running.

Solution This book uses Debian, the Linux distribution that currently ships on the Bone. However this book is based on a newer version (BeagleBoard.org Debian Bullseye IoT Image 2023-06-03) than what is shipping at the time of this writing. You can see which version your Bone is running by following the instructions in *Getting Started, Out of the Box* to log into the Bone. Then run:

```
bone$ cat /etc/dogtag
BeagleBoard.org Debian Bullseye IoT Image 2023-06-03
```

I'm running the 2023-06-03 version.

Running the Python and JavaScript Examples

Problem You'd like to learn Python or JavaScript interact with the Bone to perform physical computing tasks without first learning Linux.

Solution Plug your board into the USB of your host computer and browse to http://192.168.7.2:3000 using Google Chrome or Firefox (as shown in *Getting Started, Out of the Box*). In the left column, click on *examples*, then *BeagleBone* and then *Black*. Several sample scripts will appear. Go and explore them.

Tip: Explore the various demonstrations of Python and JavaScript. These are what come with the Bone. In *Cloning the Cookbook Repository* you see how to load the examples for the Cookbook.

Cloning the Cookbook Repository

Problem You want to run the Cookbook examples.

Solution Connect your Bone to the Internet and log into it. From the command line run:

```
bone$ git clone https://git.beagleboard.org/beagleboard/beaglebone-cookbook-
↪code
bone$ cd beaglebone-cookbook-code
bone$ ls
```

You can look around from the command line, or explore from Visual Sudio Code. If you are using VSC, go to the *File* menu and select *Open Folder ...* and select beaglebone-cookbook-code. Then explore.

Wiring a Breadboard

Problem You would like to use a breadboard to wire things to the Bone.

Solution Many of the projects in this book involve interfacing things to the Bone. Some plug in directly, like the USB port. Others need to be wired. If it's simple, you might be able to plug the wires directly into the *P8* or *P9* headers. Nevertheless, many require a breadboard for the fastest and simplest wiring.

To make this recipe, you will need:

- Breadboard and jumper wires

The *Breadboard wired to BeagleBone Black* shows a breadboard wired to the Bone. All the diagrams in this book assume that the ground pin (*P9_1* on the Bone) is wired to the negative rail and 3.3 V (*P9_3*) is wired to the positive rail.

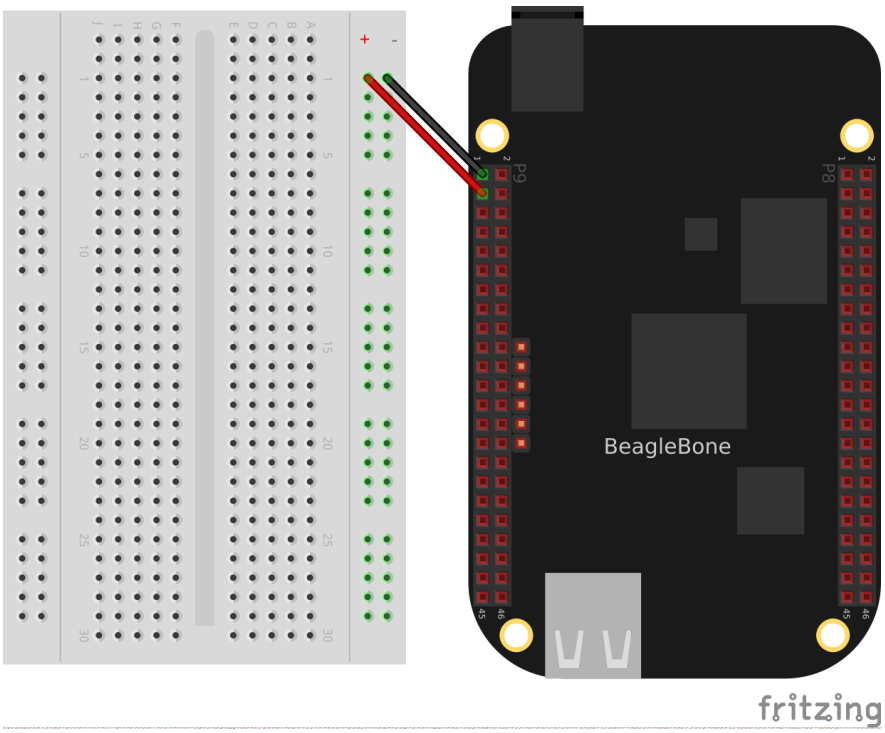


Fig. 15.4: Breadboard wired to BeagleBone Black

Breadboard wired to BeagleBone Black

Editing Code Using Visual Studio Code

Problem You want to edit and debug files on the Bone.

Solution Plug your Bone into a host computer via the USB cable. Open a browser (either Google Chrome or FireFox will work) on your host computer (as shown in *Getting Started, Out of the Box*). After the Bone has booted up, browse to http://192.168.7.2:3000 on your host. You will see something like *Visual Studio Code*.

Click the *examples* folder on the left and then click *BeagleBoard* and then *Black*, finally double-click `seqLEDs.py`. You can now edit the file.

Note: If you edit lines 33 and 37 of the `seqLEDs.py` file (time.sleep(0.25)), changing *0.25* to *0.1*, the LEDs next to the Ethernet port on your Bone will flash roughly twice as fast.

Running Python and JavaScript Applications from Visual Studio Code

Problem You have a file edited in VS Code, and you want to run it.

Solution VS Code has a *bash* command window built in at the bottom of the window. If it's not there, hit Ctrl-Shift-P and then type *terminal create new* then hit *Enter*. The terminal will appear at the bottom of the screen. You can run your code from this window. To do so, add `#!/usr/bin/env python` at the top of the file that you want to run and save.

Tip: If you are running JavaScript, replace the word **python** in the line with **node**.

At the bottom of the VS Code window are a series of tabs (*Visual Studio Code showing bash terminal*). Click the *TERMINAL* tab. Here, you have a command prompt.

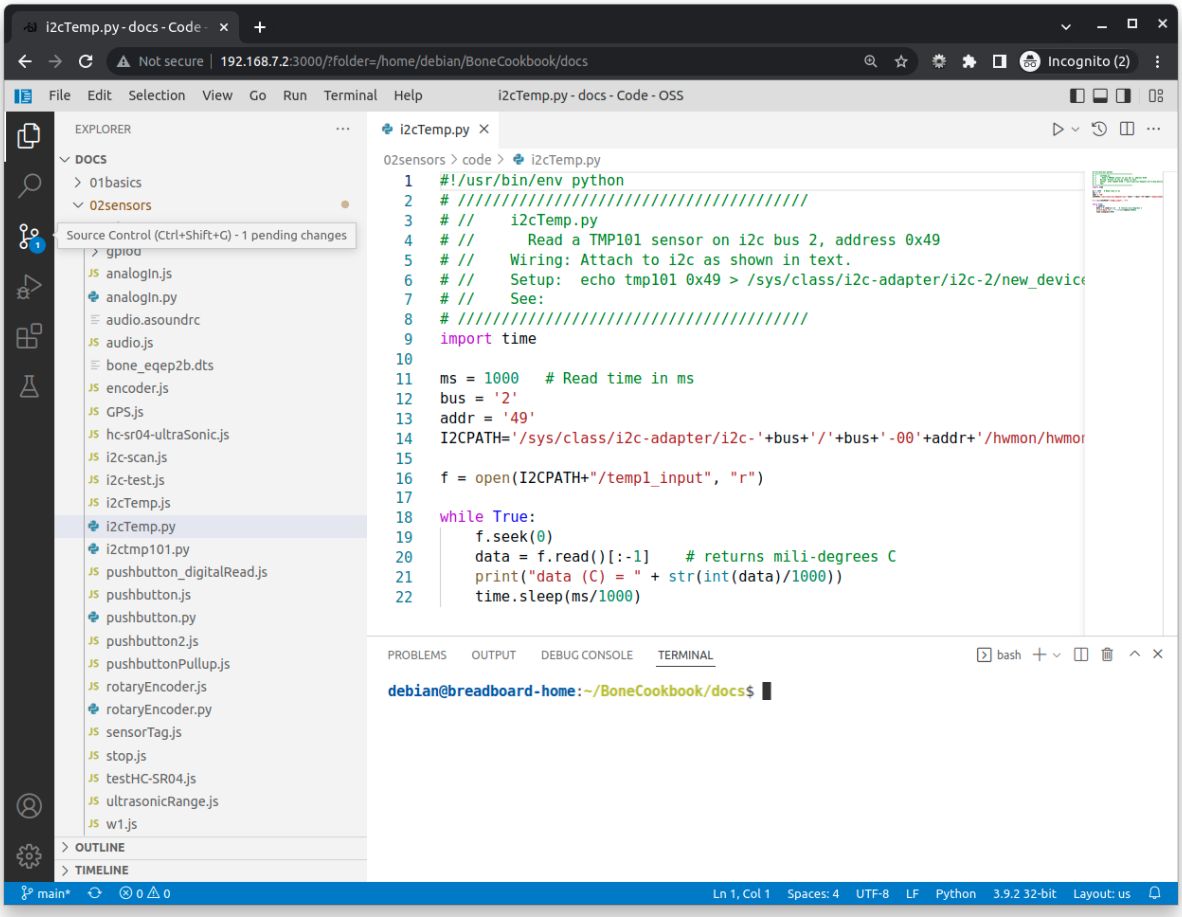


Fig. 15.5: Visual Studio Code showing bash terminal

Change to the directory that contains your file, make it executable, and then run it:

```
bone$ cd ~/examples/BeagleBone/Black/
bone$ ./seqLEDs.py
```

The *cd* is the change directory command. After you *cd*, you are in a new directory. Finally, *./seqLEDs.py* instructs the python script to run. You will need to press ^C (Ctrl-C) to stop your program.

Finding the Latest Version of the OS for Your Bone

Problem You want to find out the latest version of Debian that is available for your Bone.

Solution On your host computer, open a browser and go to https://forum.beagleboard.org/tag/latest-images This shows you a list of dates of the most recent Debian images (*Latest Debian images*).

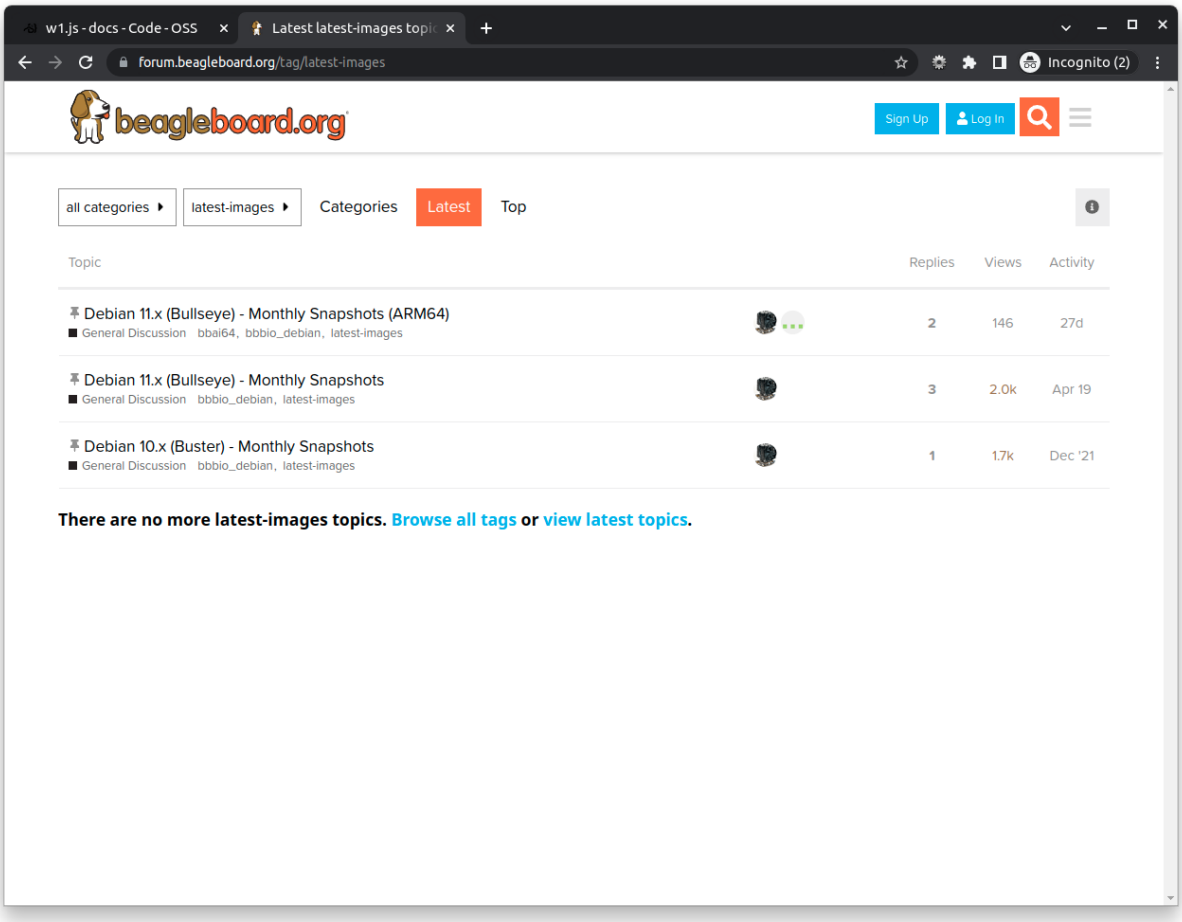


Fig. 15.6: Latest Debian images

At the time of writing, we are using the *Bullseye* image. Click on its link. Scrolling up you'll find *Latest Debian images*. There are three types of snapshots, Minimal, IoT and Xfce Desktop. IoT is the one we are running.

These are the images you want to use if you are flashing a Rev C BeagleBone Black onboard flash, or flashing a 4 GB or bigger miscroSD card. The image beginning with *am335x-debian-11.3-iot-* is used for the non-AI boards. The one beginning with *am57xx-debian-* is for programming the Beagle AI's.

Note: The onboard flash is often called the *eMMC* memory. We just call it *onboard flash*, but you'll often see *eMMC* appearing in filenames of images used to update the onboard flash.

Click the image you want to use and it will download. The images are some 500M, so it might take a while.

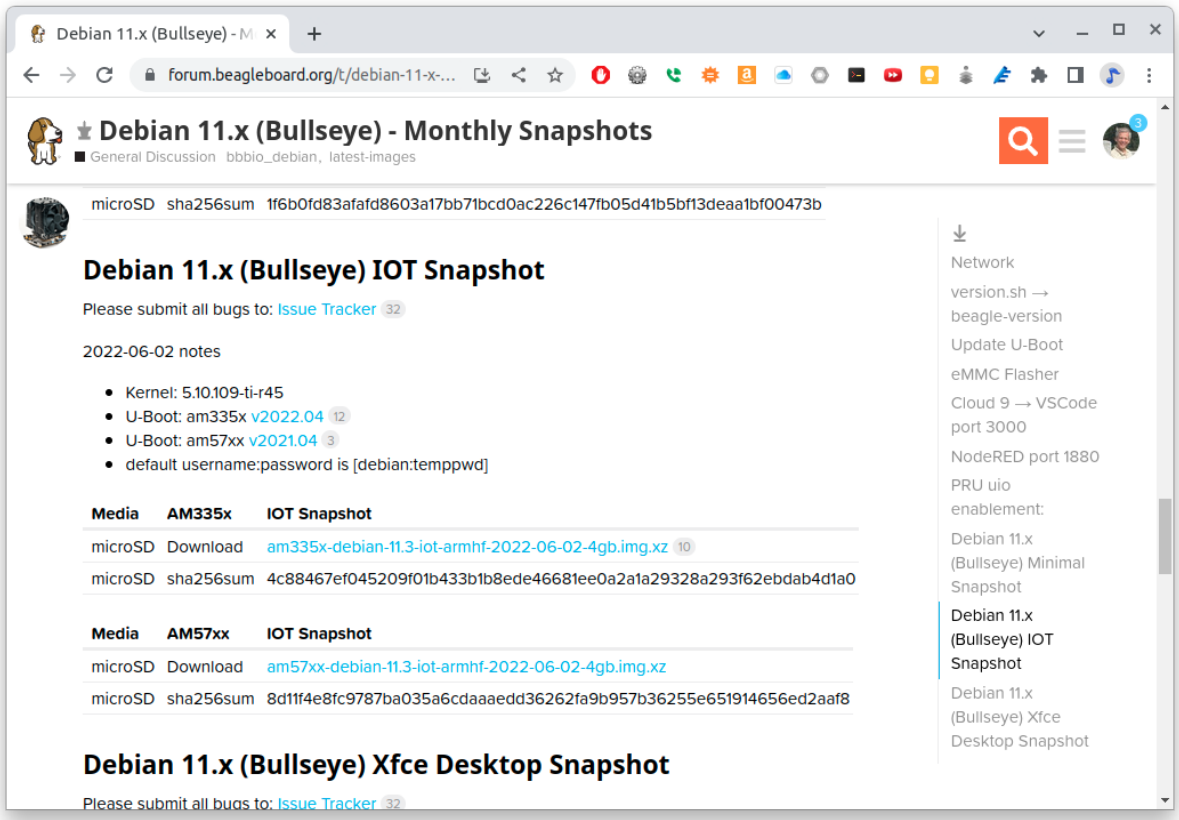


Fig. 15.7: Latest Debian images

Running the Latest Version of the OS on Your Bone

Problem You want to run the latest version of the operating system on your Bone without changing the onboard flash.

Solution This solution is to flash an external microSD card and run the Bone from it. If you boot the Bone with a microSD card inserted with a valid boot image, it will boot from the microSD card. If you boot without the microSD card installed, it will boot from the onboard flash.

Tip: If you want to reflash the onboard flash memory, see *Updating the Onboard Flash*.

Note: I instruct my students to use the microSD for booting. I suggest they keep an extra microSD flashed with the current OS. If they mess up the one on the Bone, it takes only a moment to swap in the extra microSD, boot up, and continue running. If they are running off the onboard flash, it will take much longer to reflash and boot from it.

Download the image you found in *Finding the Latest Version of the OS for Your Bone*. It's more than 500 MB, so be sure to have a fast Internet connection. Then go to http://beagleboard.org/getting-started#update and follow the instructions there to install the image you downloaded.

Updating the OS on Your Bone

Problem You've installed the latest version of Debian on your Bone (*Running the Latest Version of the OS on Your Bone*), and you want to be sure it's up-to-date.

Solution Ensure that your Bone is on the network and then run the following command on the Bone:

```
bone$ sudo apt update
bone$ sudo apt upgrade
```

If there are any new updates, they will be installed.

Note: If you get the error *The following signatures were invalid: KEYEXPIRED 1418840246*, see eLinux support page for advice on how to fix it.

Discussion After you have a current image running on the Bone, it's not at all difficult to keep it upgraded.

Backing Up the Onboard Flash

Problem You've modified the state of your Bone in a way that you'd like to preserve or share.

Solution The eLinux wiki page on BeagleBone Black Extracting eMMC contents provides some simple steps for copying the contents of the onboard flash to a file on a microSD card:

- Get a 4 GB or larger microSD card that is FAT formatted.
- If you create a FAT-formatted microSD card, you must edit the partition and ensure that it is a bootable partition.
- Download beagleboneblack-save-emmc.zip and uncompress and copy the contents onto your microSD card.
- Eject the microSD card from your computer, insert it into the powered-off BeagleBone Black, and apply power to your board.
- You'll notice *USER0* (the LED closest to the S1 button in the corner) will (after about 20 seconds) begin to blink steadily, rather than the double-pulse "heartbeat" pattern that is typical when your BeagleBone Black is running the standard Linux kernel configuration.
- It will run for a bit under 10 minutes and then *USER0* will stay on steady. That's your cue to remove power, remove the microSD card, and put it back into your computer.
- You will see a file called *BeagleBoneBlack-eMMC-image-XXXXX.img*, where *XXXXX* is a set of random numbers. Save this file to use for restoring your image later.

Note: Because the date won't be set on your board, you might want to adjust the date on the file to remember when you made it. For storage on your computer, these images will typically compress very well, so use your favorite compression tool.

Tip: The eLinux wiki is the definitive place for the BeagleBoard.org community to share information about the Beagles. Spend some time looking around for other helpful information.

Updating the Onboard Flash

Problem You want to copy the microSD card to the onboard flash.

Solution If you want to update the onboard flash with the contents of the microSD card,

- Repeat the steps in *Running the Latest Version of the OS on Your Bone* to update the OS.
- Attach to an external 5 V source. *you must be powered from an external 5 V source*. The flashing process requires more current than what typically can be pulled from USB.
- Boot from the microSD card.
- Log on to the bone and edit */boot/uEnv.txt*.
- Uncomment out the last line *cmdline=init=/usr/sbin/init-beagle-flasher*.
- Save the file and reboot.
- The USR LEDs will flash back and forth for a few minutes.
- When they stop flashing, remove the SD card and reboot.
- You are now running from the newly flashed onboard flash.

Warning: If you write the onboard flash, **be sure to power the Bone from an external 5 V source**. The USB might not supply enough current.

When you boot from the microSD card, it will copy the image to the onboard flash. When all four *USER* LEDs turn off (in some versions, they all turn on), you can power down the Bone and remove the microSD card. The next time you power up, the Bone will boot from the onboard flash.

15.1.2 Sensors

In this chapter, you will learn how to sense the physical world with BeagleBone Black. Various types of electronic sensors, such as cameras and microphones, can be connected to the Bone using one or more interfaces provided by the standard USB 2.0 host port, as shown in *The USB 2.0 host port*.

Note: All the examples in the book assume you have cloned the Cookbook repository on git.beagleboard.org. Go here *Cloning the Cookbook Repository* for instructions.

The two 46-pin cape headers (called *P8* and *P9*) along the long edges of the board (*Cape Headers P8 and P9*) provide connections for cape add-on boards, digital and analog sensors, and more.

The simplest kind of sensor provides a single digital status, such as off or on, and can be handled by an *input mode* of one of the Bone's 65 general-purpose input/output (GPIO) pins. More complex sensors can be connected by using one of the Bone's seven analog-to-digital converter (ADC) inputs or several I^2C buses.

Displays and Other Outputs discusses some of the *output mode* usages of the GPIO pins.

All these examples assume that you know how to edit a file (*Editing Code Using Visual Studio Code*) and run it, either within the Visual Studio Code (VSC) integrated development environment (IDE) or from the command line (*Getting to the Command Shell via SSH*).

Choosing a Method to Connect Your Sensor

Problem You want to acquire and attach a sensor and need to understand your basic options.

Solution *Some of the many sensor connection options on the Bone* shows many of the possibilities for connecting a sensor.

Choosing the simplest solution available enables you to move on quickly to addressing other system aspects. By exploring each connection type, you can make more informed decisions as you seek to optimize and troubleshoot your design.

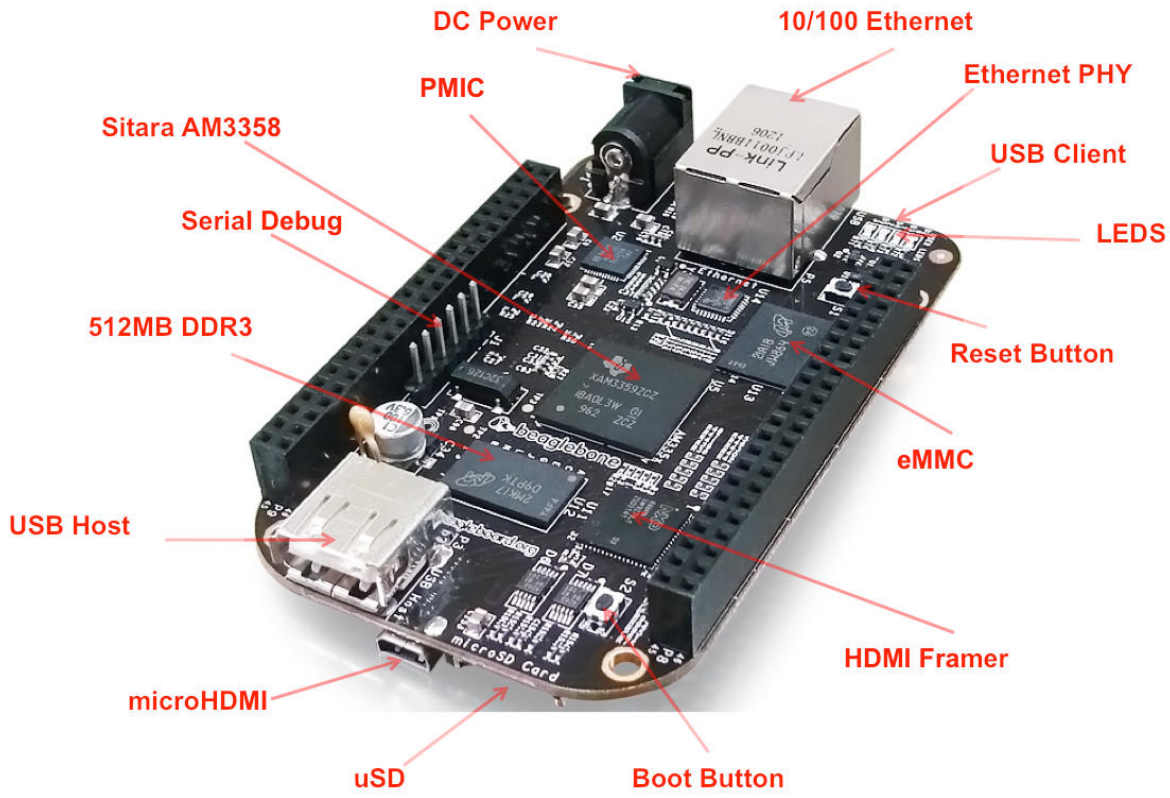


Fig. 15.8: The USB 2.0 host port

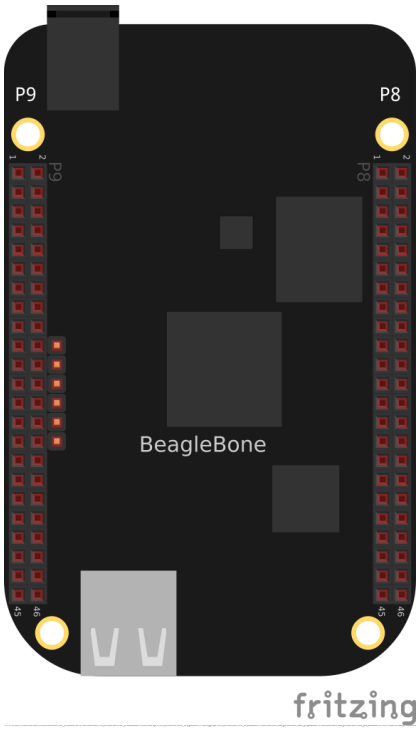


Fig. 15.9: Cape Headers P8 and P9

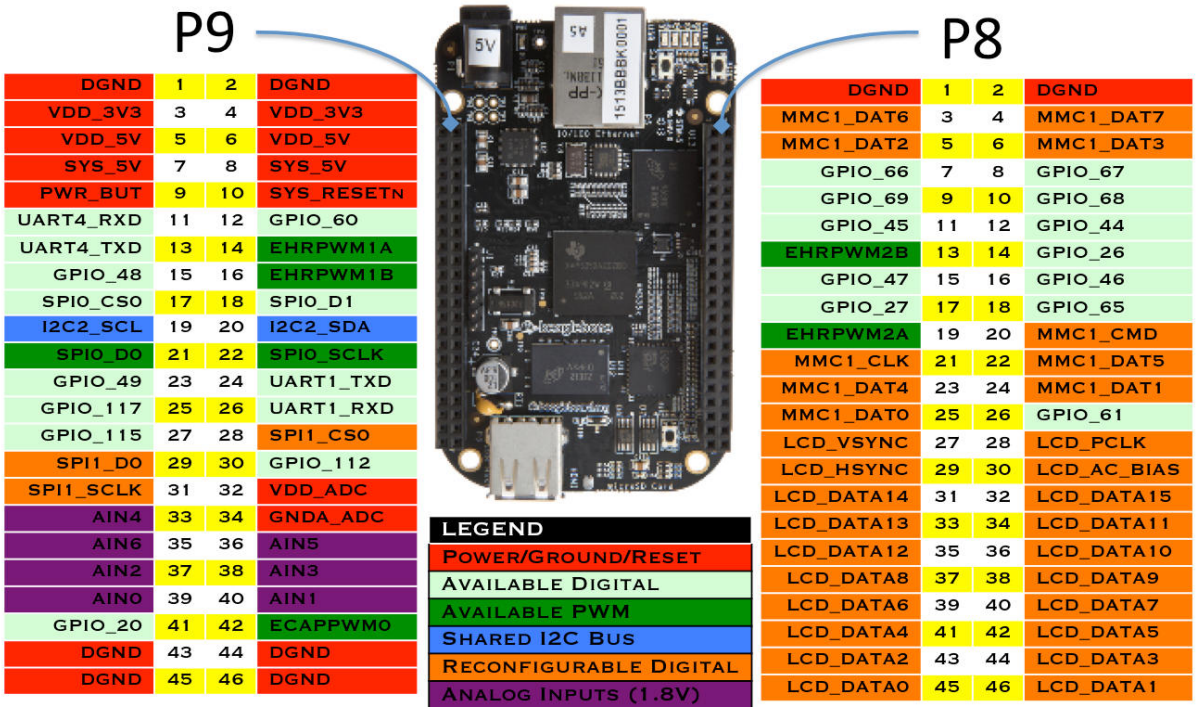


Fig. 15.10: Some of the many sensor connection options on the Bone

Input and Run a Python or JavaScript Application for Talking to Sensors

Problem You have your sensors all wired up and your Bone booted up, and you need to know how to enter and run your code.

Solution You are just a few simple steps from running any of the recipes in this book.

- Plug your Bone into a host computer via the USB cable (*Getting Started, Out of the Box*).
- Start Visual Studio Code (*Editing Code Using Visual Studio Code*).
- In the *bash* tab (as shown in *Entering commands in the VSC bash tab*), run the following commands:

```
bone$ cd
bone$ cd beaglebone-cookbook-code/02sensors
```

Here, we issued the *change directory* (*cd*) command without specifying a target directory. By default, it takes you to your home directory. Notice that the prompt has changed to reflect the change.

Note: If you log in as *debian*, your home is */home/debian*. If you were to create a new user called *newuser*, that user's home would be */home/newuser*. By default, all non-root (non-superuser) users have their home directories in */home*.

Note: All the examples in the book assume you have cloned the Cookbook repository on git.beagleboard.org. Go here *Cloning the Cookbook Repository* for instructions.

- Double-click the *pushbutton.py* file to open it.
- Press ^S (Ctrl-S) to save the file. (You can also go to the File menu in VSC and select Save to save the file, but Ctrl-S is easier.) Even easier, VSC can be configured to autosave every so many seconds.
- In the *bash* tab, enter the following commands:

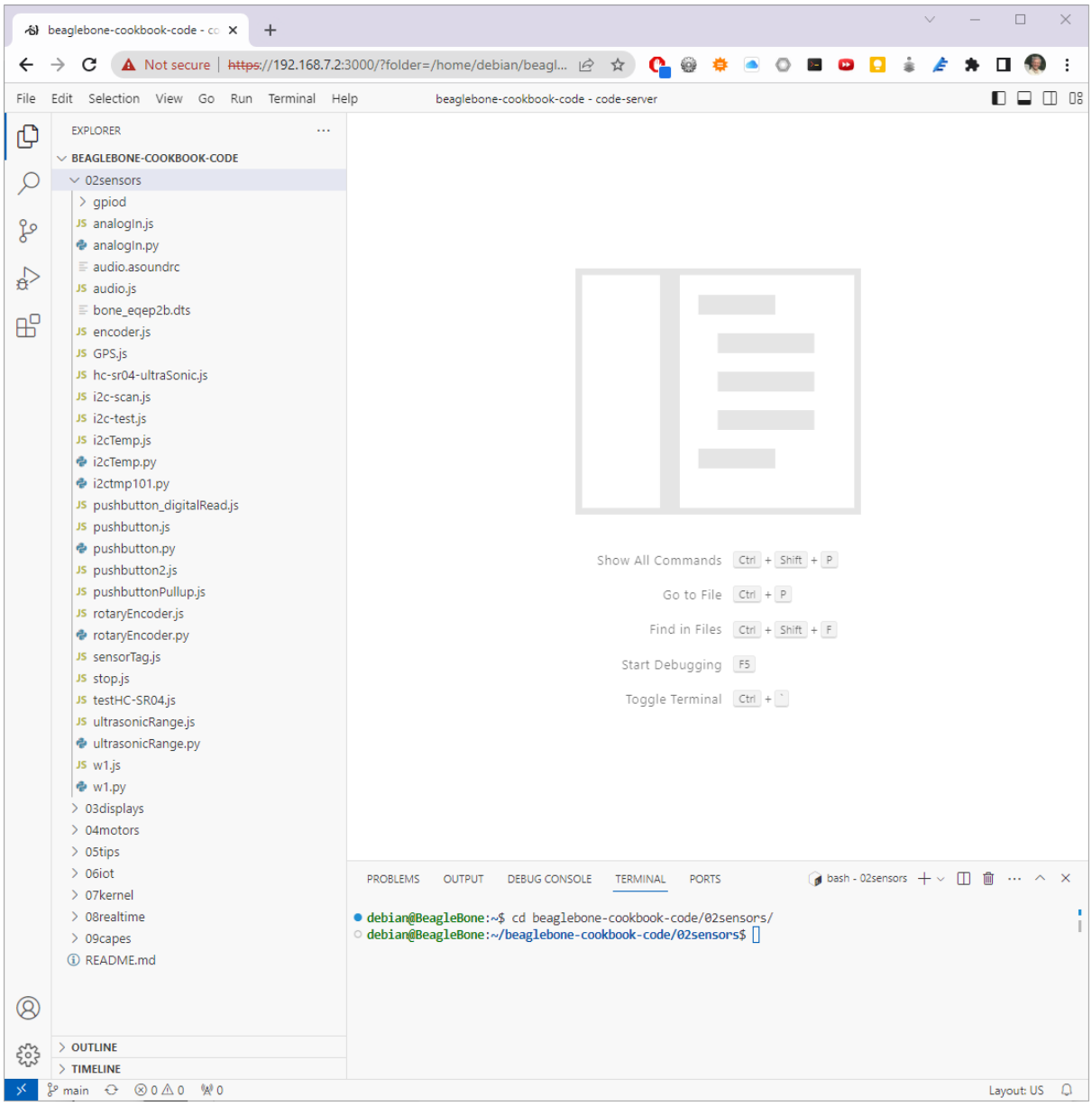


Fig. 15.11: Entering commands in the VSC bash tab

```
debian@beaglebone:beaglebone-cookbook/code/02sensors$ ./pushbutton.py
data= 0
data= 0
data= 1
data= 1
^C
```

This process will work for any script in this book.

Reading the Status of a Pushbutton or Magnetic Switch (Passive On/Off Sensor)

Problem You want to read a pushbutton, a magnetic switch, or other sensor that is electrically open or closed.

Solution Connect the switch to a GPIO pin and read from the proper place in */sys/class/gpio*.

To make this recipe, you will need:

- Breadboard and jumper wires.
- Pushbutton switch.
- Magnetic reed switch. (optional)

You can wire up either a pushbutton, a magnetic reed switch, or both on the Bone, as shown in *Diagram for wiring a pushbutton and magnetic reed switch input*.

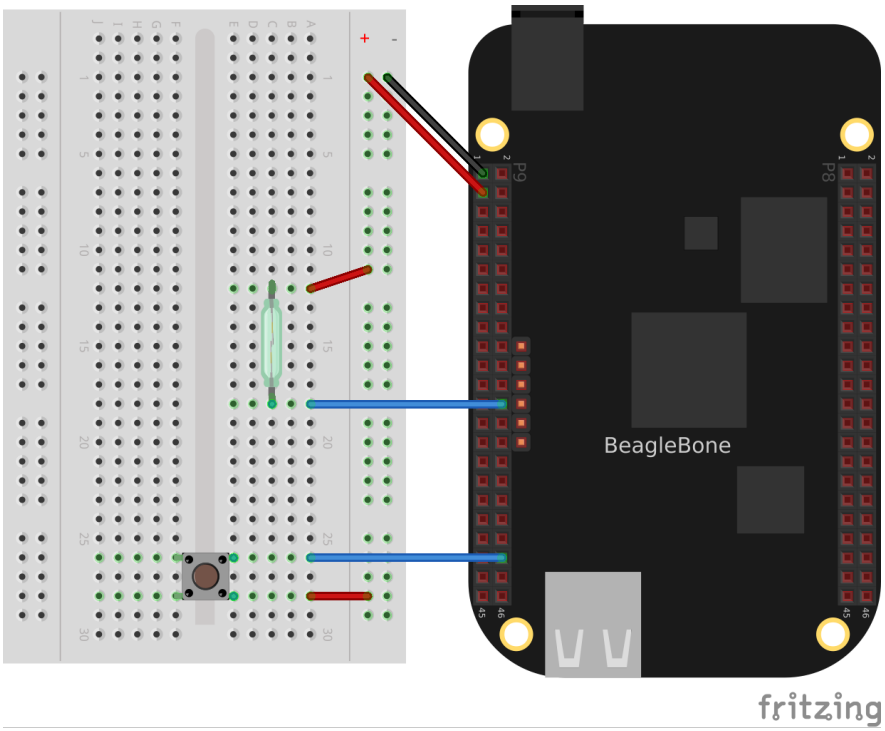


Fig. 15.12: Diagram for wiring a pushbutton and magnetic reed switch input

The code in *Monitoring a pushbutton (pushbutton.py)* reads GPIO port *P9_42*, which is attached to the pushbutton.

Python

c

Listing 15.1: Monitoring a pushbutton (pushbutton.py)

```
#!/usr/bin/env python
# ////////////////////////////////////
# //        pushbutton.py
# //      Reads P9_42 and prints its value.
# //        Wiring:        Connect a switch between P9_42 and 3.3V
# //        Setup:
# //        See:
# ////////////////////////////////////
import time
import gpiod
import os

ms = 100    # Read time in ms
CHIP = 'gpiochip0'
LINE_OFFSET = [7] #  P9_42 is gpio 7
chip = gpiod.Chip(CHIP)
lines = chip.get_lines(LINE_OFFSET)
lines.request(consumer='pushbutton.py', type=gpiod.LINE_REQ_DIR_IN)

while True:
    data = lines.get_values()
    print('data = ' + str(data[0]))
    time.sleep(ms/1000)
```

pushbutton.py

Listing 15.2: Monitoring a pushbutton (pushbutton.c)

```
////////////////////////////////////
//        pushbutton.c
//      Reads P9_42 and prints its value.
//        Wiring:        Connect a switch between P9_42 and 3.3V
//        Setup:
//        See:
////////////////////////////////////
#include <gpiod.h>
#include <stdio.h>
#include <unistd.h>

#define CONSUMER        "pushbutton.c"

int main(int argc, char **argv)
{
        int chipnumber = 0;
        unsigned int line_num = 7;
        struct gpiod_line *line;
        struct gpiod_chip *chip;
        int i, ret;

        chip = gpiod_chip_open_by_number(chipnumber);
        line = gpiod_chip_get_line(chip, line_num);
        ret = gpiod_line_request_input(line, CONSUMER);

        /* Get */
        while(1) {
                printf("%d\r", gpiod_line_get_value(line));
                usleep(100);
        }
}
```

pushbutton.c

Put this code in a file called *pushbutton.py* following the steps in *Input and Run a Python or JavaScript Application for Talking to Sensors*. In the VSC *bash* tab, run it by using the following commands:

```
bone$ ./pushbutton.py
data = 0
data = 0
data = 1
data = 1
^C
```

The command runs it. Try pushing the button. The code reads the pin and prints its current value.

You will have to press ^C (Ctrl-C) to stop the code.

If you want to run the C version do:

```
bone$ gcc -o pushbutton pushbutton.c -lgpiod
bone$ ./pushbutton
data = 0
data = 0
data = 1
data = 1
^C
```

If you want to use the magnetic reed switch wired as shown in *Diagram for wiring a pushbutton and magnetic reed switch input*, change *P9_42* to *P9_26* which is gpio *14*.

Mapping Header Numbers to gpio Numbers

Problem You have a sensor attached to the P8 or P9 header and need to know which gpio pin it is using.

Solution The *gpioinfo* command displays information about all the P8 and P9 header pins. To see the info for just one pin, use *grep*.

```
bone$ gpioinfo | grep -e chip -e P9.42
gpiochip0 - 32 lines:
        line   7: "P8_42A [ecappwm0]" "P9_42" input active-high [used]
gpiochip1 - 32 lines:
gpiochip2 - 32 lines:
gpiochip3 - 32 lines:
```

This shows P9_42 is on chip 0 and pin 7. To find the gpio number multiply the chip number by 32 and add it to the pin number. This gives 0*32+7=7.

For P9_26 you get:

```
bone$ gpioinfo | grep -e chip -e P9.26
gpiochip0 - 32 lines:
        line  14: "P9_26 [uart1_rxd]" "P9_26" input active-high [used]
gpiochip1 - 32 lines:
gpiochip2 - 32 lines:
gpiochip3 - 32 lines:
```

0*32+14=14, so the P9_26 pin is gpio 14.

Reading a Position, Light, or Force Sensor (Variable Resistance Sensor)

Problem You have a variable resistor, force-sensitive resistor, flex sensor, or any of a number of other sensors that output their value as a variable resistance, and you want to read their value with the Bone.

Solution Use the Bone's analog-to-digital converters (ADCs) and a resistor divider circuit to detect the resistance in the sensor.

The Bone has seven built-in analog inputs that can easily read a resistive value. *Seven analog inputs on P9 header* shows them on the lower part of the *P9* header.

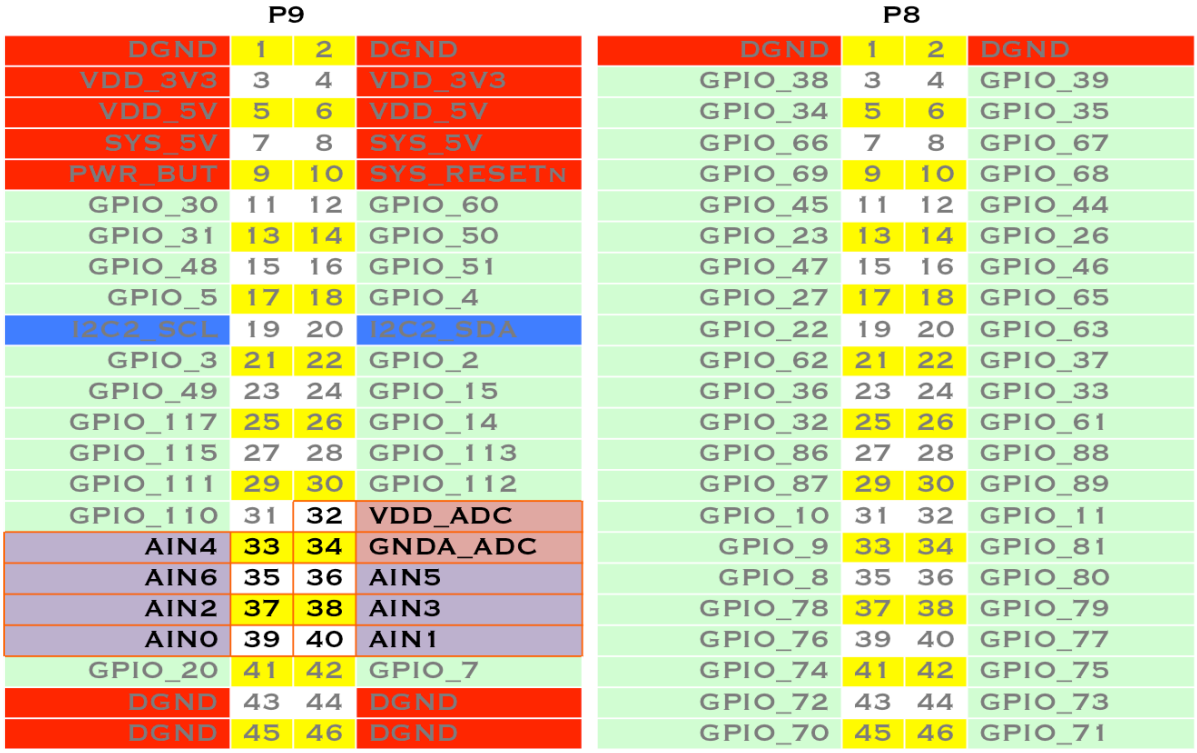


Fig. 15.13: Seven analog inputs on P9 header

To make this recipe, you will need:

- Breadboard and jumper wires.
- 10k trimpot or
- Flex resistor (optional)
- 22 kΩ resistor

A variable resistor with three terminals *Wiring a 10 kΩ variable resistor (trimpot) to an ADC port* shows a simple variable resistor (trimpot) wired to the Bone. One end terminal is wired to the ADC 1.8 V power supply on pin *P9_32*, and the other end terminal is attached to the ADC ground (*P9_34*). The middle terminal is wired to one of the seven analog-in ports (*P9_36*).

Reading an analog voltage (analogIn.py) shows the code used to read the variable resistor. Add the code to a file called *analogIn.py* and run it; then change the resistor and run it again. The voltage read will change.

Python

JavaScript

Listing 15.3: Reading an analog voltage (analogIn.py)

```
#!/usr/bin/env python3
#////////////////////////////////////
#         analogin.py
#          Reads the analog value of the light sensor.
#////////////////////////////////////
```

(continues on next page)

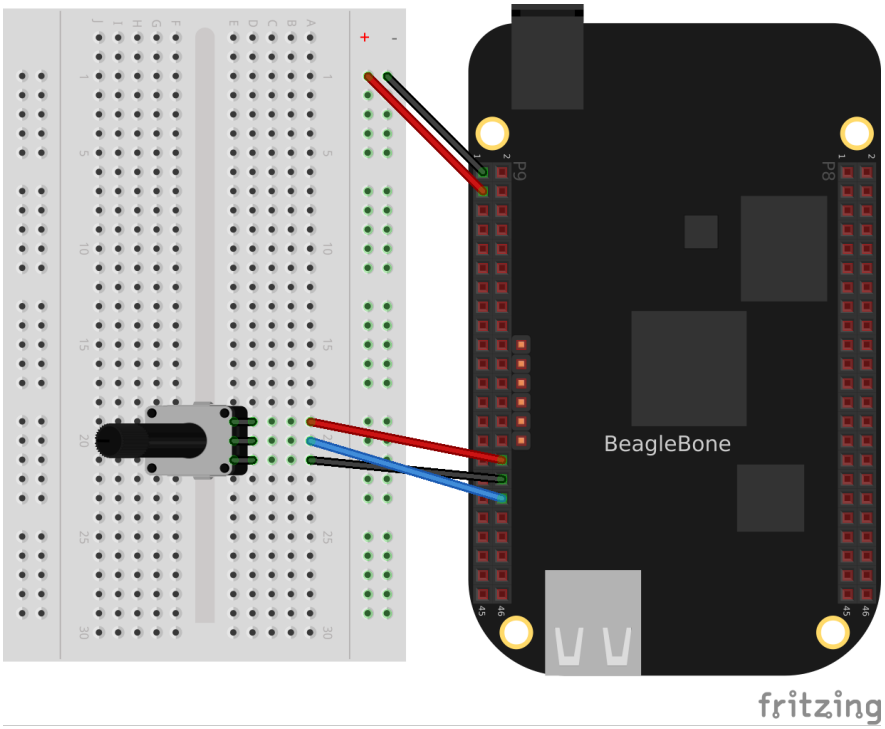


Fig. 15.14: Wiring a 10 kΩ variable resistor (trimpot) to an ADC port

(continued from previous page)

```
import time
import os

pin = "2"         # light sensor, A2, P9_37

IIOPATH='/sys/bus/iio/devices/iio:device0/in_voltage'+pin+'_raw'

print('Hit ^C to stop')

f = open(IIOPATH, "r")

while True:
    f.seek(0)
    x = float(f.read())/4096
    print('{}: {:.1f}%, {:.3f} V'.format(pin, 100*x, 1.8*x), end = '\r')
    time.sleep(0.1)

# // Bone   | Pocket | AIN
# // ------ | ------ | ---
# // P9_39 | P1_19  | 0
# // P9_40 | P1_21  | 1
# // P9_37 | P1_23  | 2
# // P9_38 | P1_25  | 3
# // P9_33 | P1_27  | 4
# // P9_36 | P2_35  | 5
# // P9_35 | P1_02  | 6
```

analogIn.py

Listing 15.4: Reading an analog voltage (analogIn.js)

```
#!/usr/bin/env node
////////////////////////////////////
```

(continues on next page)

(continued from previous page)

```
//        analogin.js
//         Reads the analog value of the light sensor.
////////////////////////////////////
const fs = require("fs");
const ms = 500;  // Time in milliseconds

const pin = "2";         // light sensor, A2, P9_37

const IIOPATH='/sys/bus/iio/devices/iio:device0/in_voltage'+pin+'_raw';

console.log('Hit ^C to stop');

// Read every 500ms
setInterval(readPin, ms);

function readPin() {
    var data = fs.readFileSync(IIOPATH).slice(0, -1);
    console.log('data = ' + data);
 }
// Bone  | Pocket | AIN
// ----- | ------ | ---
// P9_39 | P1_19  | 0
// P9_40 | P1_21  | 1
// P9_37 | P1_23  | 2
// P9_38 | P1_25  | 3
// P9_33 | P1_27  | 4
// P9_36 | P2_35  | 5
// P9_35 | P1_02  | 6
```

analogIn.js

Note: The code in *Reading an analog voltage (analogIn.js)* outputs a value between 0 and 4096.

A variable resistor with two terminals Some resistive sensors have only two terminals, such as the flex sensor in *Reading a two-terminal flex resistor* The resistance between its two terminals changes when it is flexed. In this case, we need to add a fixed resistor in series with the flex sensor. *Reading a two-terminal flex resistor* shows how to wire in a 22 kΩ resistor to give a voltage to measure across the flex sensor.

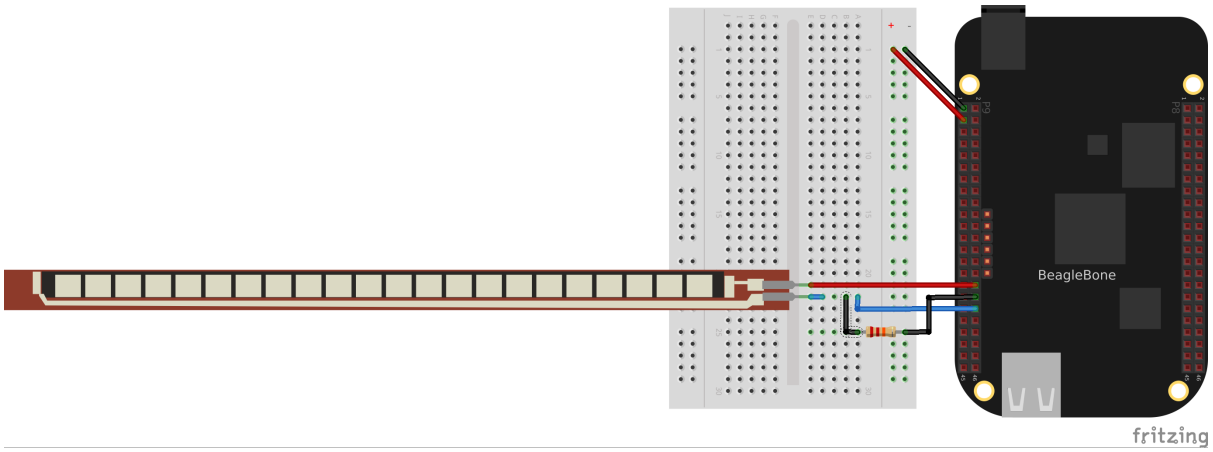


Fig. 15.15: Reading a two-terminal flex resistor

The code in *Reading an analog voltage (analogIn.py)* and *Reading an analog voltage (analogIn.js)* also works for this setup.

Reading a Distance Sensor (Analog or Variable Voltage Sensor)

Problem You want to measure distance with a LV-MaxSonar-EZ1 Sonar Range Finder, which outputs a voltage in proportion to the distance.

Solution To make this recipe, you will need:

- Breadboard and jumper wires.
- LV-MaxSonar-EZ1 Sonar Range Finder

All you have to do is wire the EZ1 to one of the Bone's *analog-in* pins, as shown in *Wiring the LV-MaxSonar-EZ1 Sonar Range Finder to the P9_33 analog-in port*. The device outputs ~6.4 mV/in when powered from 3.3 V.

Warning: Make sure not to apply more than 1.8 V to the Bone's *analog-in* pins, or you will likely damage them. In practice, this circuit should follow that rule.

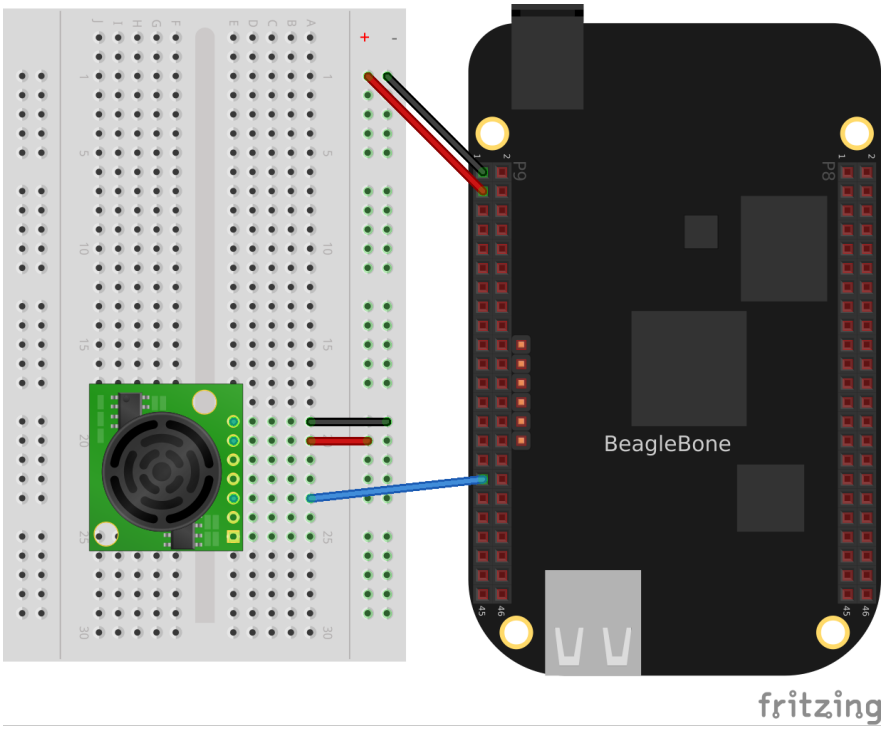


Fig. 15.16: Wiring the LV-MaxSonar-EZ1 Sonar Range Finder to the *P9_33* analog-in port

Reading an analog voltage (ultrasonicRange.py) shows the code that reads the sensor at a fixed interval.

Python

JavaScript

Listing 15.5: Reading an analog voltage (ultrasonicRange.py)

```
#!/usr/bin/env python
# ////////////////////////////////////
# //         ultrasonicRange.js
# //          Reads the analog value of the sensor.
# ////////////////////////////////////
import time
ms = 250;   # Time in milliseconds

```

(continues on next page)

(continued from previous page)

```
pin = "0"         # sensor, A0, P9_39

IIOPATH='/sys/bus/iio/devices/iio:device0/in_voltage'+pin+'_raw'

print('Hit ^C to stop');

f = open(IIOPATH, "r")
while True:
    f.seek(0)
    data = f.read()[:-1]
    print('data= ' + data)
    time.sleep(ms/1000)

# // Bone  | Pocket | AIN
# // ----- | ------ | ---
# // P9_39 | P1_19  | 0
# // P9_40 | P1_21  | 1
# // P9_37 | P1_23  | 2
# // P9_38 | P1_25  | 3
# // P9_33 | P1_27  | 4
# // P9_36 | P2_35  | 5
# // P9_35 | P1_02  | 6
```

ultrasonicRange.py

Listing 15.6: Reading an analog voltage (ultrasonicRange.js)

```
#!/usr/bin/env node
////////////////////////////////////
//        ultrasonicRange.js
//         Reads the analog value of the sensor.
////////////////////////////////////
const fs = require("fs");
const ms = 250;  // Time in milliseconds

const pin = "0";         // sensor, A0, P9_39

const IIOPATH='/sys/bus/iio/devices/iio:device0/in_voltage'+pin+'_raw';

console.log('Hit ^C to stop');

// Read every ms
setInterval(readPin, ms);

function readPin() {
    var data = fs.readFileSync(IIOPATH);
    console.log('data= ' + data);
 }
// Bone  | Pocket | AIN
// ----- | ------ | ---
// P9_39 | P1_19  | 0
// P9_40 | P1_21  | 1
// P9_37 | P1_23  | 2
// P9_38 | P1_25  | 3
// P9_33 | P1_27  | 4
// P9_36 | P2_35  | 5
// P9_35 | P1_02  | 6
```

ultrasonicRange.js

Reading a Distance Sensor (Variable Pulse Width Sensor)

Problem You want to use a HC-SR04 Ultrasonic Range Sensor with BeagleBone Black.

Solution The HC-SR04 Ultrasonic Range Sensor (shown in *HC-SR04 Ultrasonic range sensor*) works by sending a trigger pulse to the *Trigger* input and then measuring the pulse width on the *Echo* output. The width of the pulse tells you the distance.

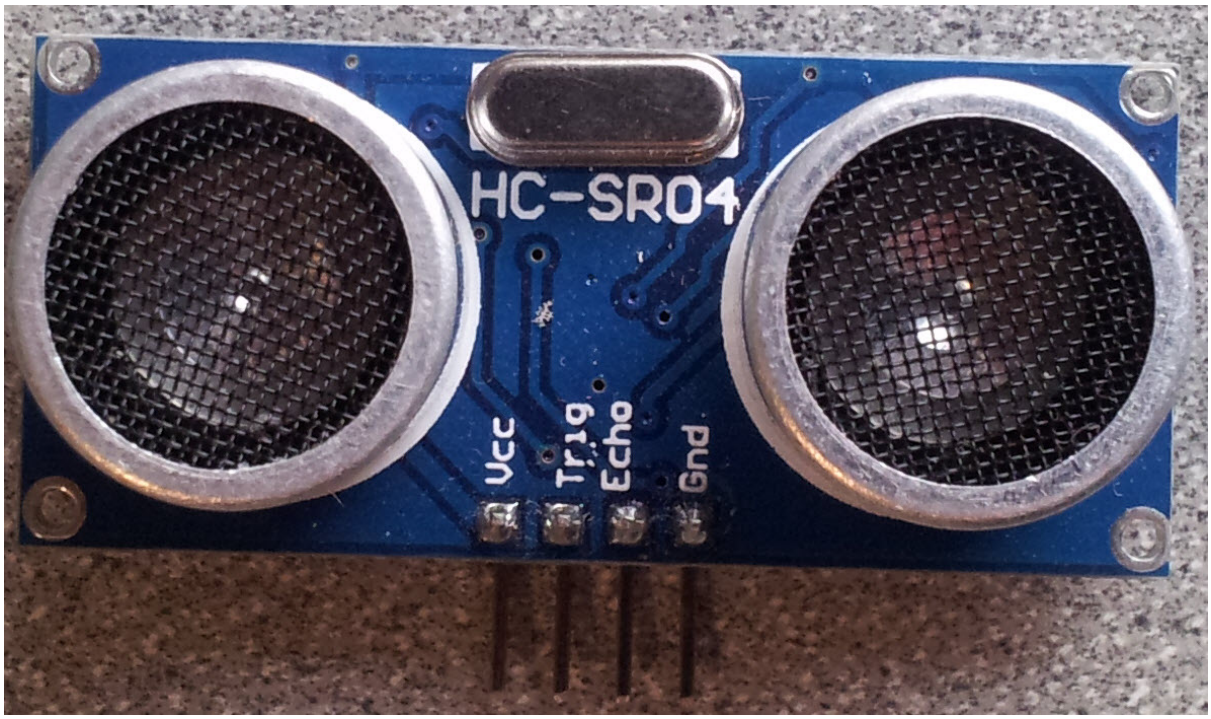


Fig. 15.17: HC-SR04 Ultrasonic range sensor

To make this recipe, you will need:

- Breadboard and jumper wires.
- 10 kΩ and 20 kΩ resistors
- HC-SR04 Ultrsonic Range Sensor.

Wire the sensor as shown in *Wiring an HC-SR04 Ultrasonic Sensor*. Note that the HC-SR04 is a 5 V device, so the *banded* wire (running from *P9_7* on the Bone to VCC on the range finder) attaches the HC-SR04 to the Bone's 5 V power supply.

Driving a HC-SR04 ultrasound sensor (hc-sr04-ultraSonic.js) shows BoneScript code used to drive the HC-SR04.

Listing 15.7: Driving a HC-SR04 ultrasound sensor (hc-sr04-ultraSonic.js)

```
#!/usr/bin/env node

// This is an example of reading HC-SR04 Ultrasonic Range Finder
// This version measures from the fall of the Trigger pulse
//   to the end of the Echo pulse

var b = require('bonescript');

var trigger = 'P9_16',  // Pin to trigger the ultrasonic pulse
    echo    = 'P9_41',  // Pin to measure to pulse width related to the␣
↪distance
```

(continues on next page)

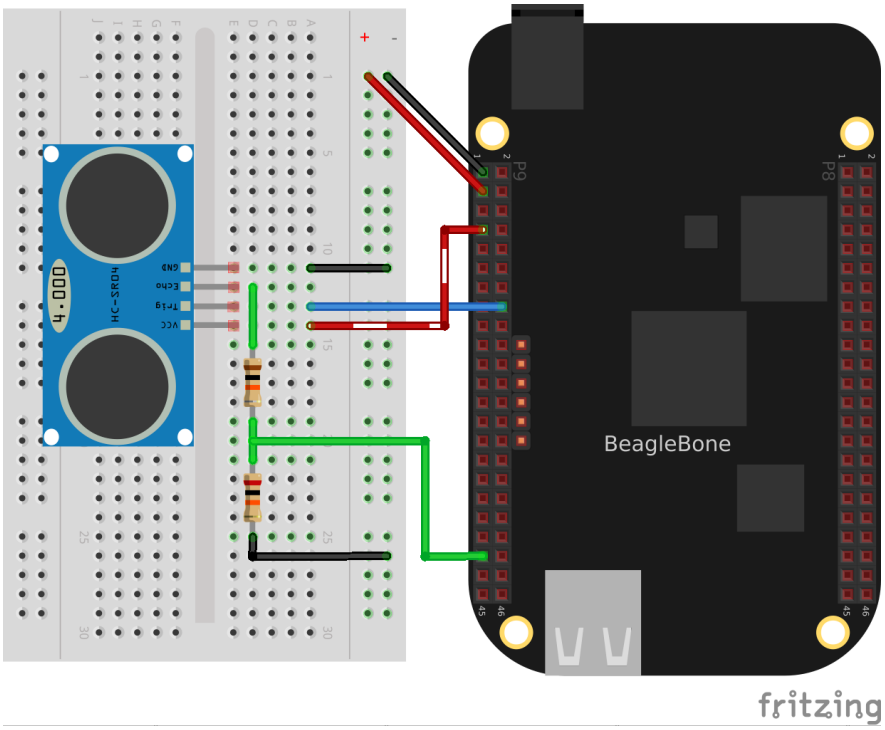


Fig. 15.18: Wiring an HC-SR04 Ultrasonic Sensor

(continued from previous page)

```
    ms = 250;           // Trigger period in ms

var startTime, pulseTime;

b.pinMode(echo,   b.INPUT, 7, 'pulldown', 'fast', doAttach);
function doAttach(x) {
    if(x.err) {
        console.log('x.err = ' + x.err);
        return;
    }
    // Call pingEnd when the pulse ends
    b.attachInterrupt(echo, true, b.FALLING, pingEnd);
}

b.pinMode(trigger, b.OUTPUT);

b.digitalWrite(trigger, 1);     // Unit triggers on a falling edge.
                                // Set trigger to high so we call pull it␣
↪low later

// Pull the trigger low at a regular interval.
setInterval(ping, ms);

// Pull trigger low and start timing.
function ping() {
    // console.log('ping');
    b.digitalWrite(trigger, 0);
    startTime = process.hrtime();
}

// Compute the total time and get ready to trigger again.
function pingEnd(x) {
    if(x.attached) {
```

(continues on next page)

(continued from previous page)

```
        console.log("Interrupt handler attached");
        return;
    }
    if(startTime) {
        pulseTime = process.hrtime(startTime);
        b.digitalWrite(trigger, 1);
        console.log('pulseTime = ' + (pulseTime[1]/1000000-0.8).toFixed(3));
    }
}
```

hc-sr04-ultraSonic.js

This code is more complex than others in this chapter, because we have to tell the device when to start measuring and time the return pulse.

Accurately Reading the Position of a Motor or Dial

Problem You have a motor or dial and want to detect rotation using a rotary encoder.

Solution Use a rotary encoder (also called a *quadrature encoder*) connected to one of the Bone's eQEP ports, as shown in *Wiring a rotary encoder using eQEP2*.

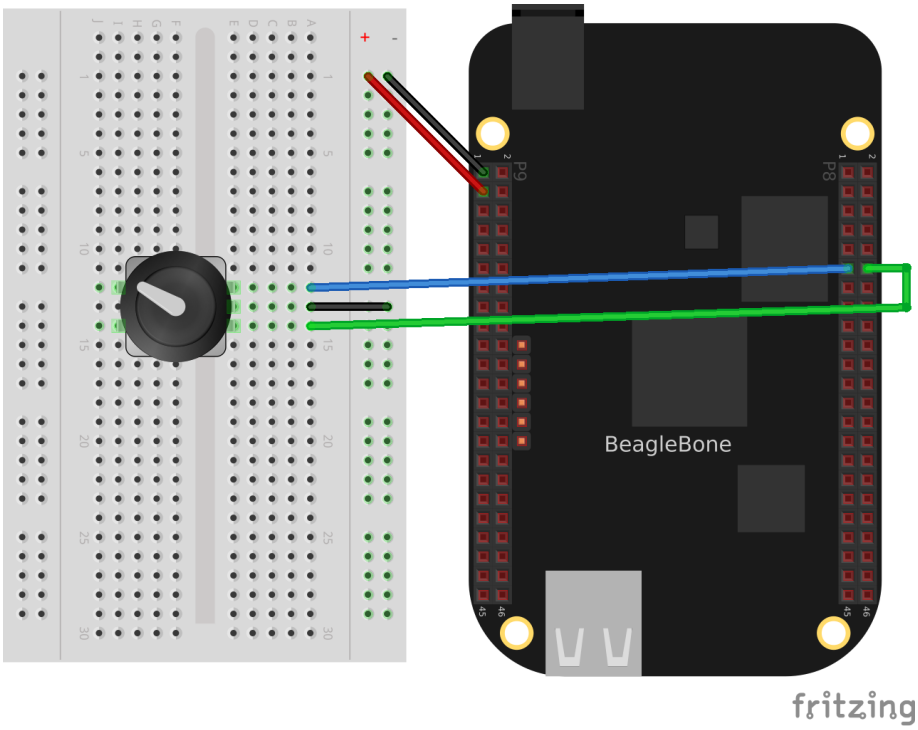


Fig. 15.19: Wiring a rotary encoder using eQEP2

Table 15.1: On the BeagleBone and PocketBeage the three encoders are:

eQEP0	P9.27 and P9.42 OR P1_33 and P2_34
eQEP1	P9.33 and P9.35
eQEP2	P8.11 and P8.12 OR P2_24 and P2_33

Table 15.2: On the AI it's:

eQEP1	P8.33 and P8.35
eQEP2	P8.11 and P8.12 or P9.19 and P9.41
eQEP3	P8.24 and P8.25 or P9.27 and P9.42

To make this recipe, you will need:

- Breadboard and jumper wires.
- Rotary encoder.

We are using a quadrature rotary encoder, which has two switches inside that open and close in such a manner that you can tell which way the shaft is turning. In this particular encoder, the two switches have a common lead, which is wired to ground. It also has a pushbutton switch wired to the other side of the device, which we aren't using.

Wire the encoder to *P8_11* and *P8_12*, as shown in *Wiring a rotary encoder using eQEP2*.

BeagleBone Black has built-in hardware for reading up to three encoders. Here, we'll use the *eQEP2* encoder via the Linux *count* subsystem.

Then run the following commands:

```
bone$ config-pin P8_11 qep
bone$ config-pin P8_12 qep
bone$ show-pins | grep qep
P8.12        12 fast rx  up  4 qep 2 in A    ocp/P8_12_pinmux (pinmux_P8_12_
↪qep_pin)
P8.11        13 fast rx  up  4 qep 2 in B    ocp/P8_11_pinmux (pinmux_P8_11_
↪qep_pin)
```

This will enable *eQEP2* on pins *P8_11* and *P8_12*. The *2* after the *qep* returned by *show-pins* shows it's *eQEP2*.

Finally, add the code in *Reading a rotary encoder (rotaryEncoder.py)* to a file named *rotaryEncoder.py* and run it.

Python

JavaScript

Listing 15.8: Reading a rotary encoder (rotaryEncoder.py)

```
#!/usr/bin/env python
# // This uses the eQEP hardware to read a rotary encoder
# // bone$ config-pin P8_11 eqep
# // bone$ config-pin P8_12 eqep
import time

eQEP = '2'
COUNTERPATH = '/dev/bone/counter/counter'+eQEP+'/count0'

ms = 100          # Time between samples in ms
maxCount = '1000000'

# Set the eEQP maximum count
f = open(COUNTERPATH+'/ceiling', 'w')
f.write(maxCount)
f.close()

# Enable
f = open(COUNTERPATH+'/enable', 'w')
f.write('1')
f.close()

f = open(COUNTERPATH+'/count', 'r')
```

(continues on next page)

(continued from previous page)

```

olddata = -1
while True:
        f.seek(0)
        data = f.read()[:-1]
        # Print only if data changes
        if data != olddata:
                olddata = data
                print("data = " + data)
        time.sleep(ms/1000)

# Black OR Pocket
# eQEP0:        P9.27 and P9.42 OR P1_33 and P2_34
# eQEP1:        P9.33 and P9.35
# eQEP2:        P8.11 and P8.12 OR P2_24 and P2_33

# AI
# eQEP1:        P8.33 and P8.35
# eQEP2:        P8.11 and P8.12 or P9.19 and P9.41
# eQEP3:        P8.24 and P8.25 or P9.27 and P9.42
```

rotaryEncoder.py

Listing 15.9: Reading a rotary encoder (rotaryEncoder.js)

```
#!/usr/bin/env node
// This uses the eQEP hardware to read a rotary encoder
// bone$ config-pin P8_11 eqep
// bone$ config-pin P8_12 eqep
const fs = require("fs");

const eQEP = "2";
const COUNTERPATH = '/dev/bone/counter/counter'+eQEP+'/count0';

const ms = 100;          // Time between samples in ms
const maxCount = '1000000';

// Set the eEQP maximum count
fs.writeFileSync(COUNTERPATH+'/ceiling', maxCount);

// Enable
fs.writeFileSync(COUNTERPATH+'/enable', '1');

setInterval(readEncoder, ms);    // Check state every ms

var olddata = -1;
function readEncoder() {
        var data = parseInt(fs.readFileSync(COUNTERPATH+'/count'));
        if(data != olddata) {
                // Print only if data changes
                console.log('data = ' + data);
                olddata = data;
        }
}

// Black OR Pocket
// eQEP0:        P9.27 and P9.42 OR P1_33 and P2_34
// eQEP1:        P9.33 and P9.35
// eQEP2:        P8.11 and P8.12 OR P2_24 and P2_33

// AI
// eQEP1:        P8.33 and P8.35
```

(continues on next page)

(continued from previous page)

```
// eQEP2:         P8.11 and P8.12 or P9.19 and P9.41
// eQEP3:         P8.24 and P8.25 or P9.27 and P9.42
```

`rotaryEncoder.js`

Try rotating the encoder clockwise and counter-clockwise. You'll see an output like this:

```
data = 32
data = 40
data = 44
data = 48
data = 39
data = 22
data = 0
data = 999989
data = 999973
data = 999972
^C
```

The values you get for *data* will depend on which way you are turning the device and how quickly. You will need to press ^C (Ctrl-C) to end.

See Also You can also measure rotation by using a variable resistor (see *Wiring a 10 kΩ variable resistor (trimpot) to an ADC port*).

Acquiring Data by Using a Smart Sensor over a Serial Connection

Problem You want to connect a smart sensor that uses a built-in microcontroller to stream data, such as a global positioning system (GPS), to the Bone and read the data from it.

Solution The Bone has several serial ports (UARTs) that you can use to read data from an external microcontroller included in smart sensors, such as a GPS. Just wire one up, and you'll soon be gathering useful data, such as your own location.

Here's what you'll need:

- Breadboard and jumper wires.
- GPS receiver

Wire your GPS, as shown in *Wiring a GPS to UART 4*.

The GPS will produce raw National Marine Electronics Association (NMEA) data that's easy for a computer to read, but not for a human. There are many utilities to help convert such sensor data into a human-readable form. For this GPS, run the following command to load a NMEA parser:

```
bone$ npm install -g nmea
```

Running the code in *Talking to a GPS with UART 4 (GPS.js)* will print the current location every time the GPS outputs it.

Listing 15.10: Talking to a GPS with UART 4 (GPS.js)

```
#!/usr/bin/env node
// Install with: npm install nmea

// Need to add exports.serialParsers = m.module.parsers;
// to the end of /usr/local/lib/node_modules/bonescript/serial.js

var b = require('bonescript');
```

(continues on next page)

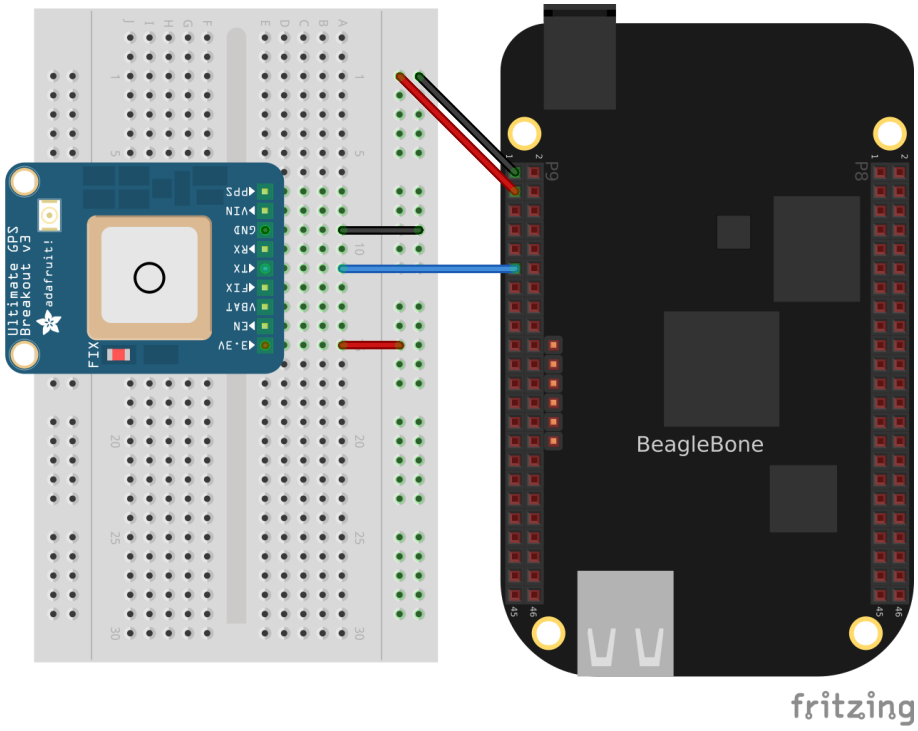


Fig. 15.20: Wiring a GPS to UART 4

(continued from previous page)

```
var nmea = require('nmea');

var port = '/dev/ttyO4';
var options = {
    baudrate: 9600,
    parser: b.serialParsers.readline("\n")
};

b.serialOpen(port, options, onSerial);

function onSerial(x) {
    if (x.err) {
        console.log('***ERROR*** ' + JSON.stringify(x));
    }
    if (x.event == 'open') {
       console.log('***OPENED***');
    }
    if (x.event == 'data') {
        console.log(String(x.data));
        console.log(nmea.parse(x.data));
    }
}
```

GPS.js

If you don't need the NMEA formatting, you can skip the *npm* part and remove the lines in the code that refer to it.

Note: If you get an error like this *TypeError: Cannot call method 'readline' of undefined*

add this line to the end of file `/usr/local/lib/node_modules/bonescript/serial.js`:

exports.serialParsers = m.module.parsers;

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
UART4_RXD	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
UART4_TXD	13	14	GPIO_50	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_5	17	18	GPIO_4	GPIO_27	17	18	GPIO_65
UART1_RTSN	19	20	UART1_CTSN	GPIO_22	19	20	GPIO_63
UART2_TXD	21	22	UART2_RXD	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	UART1_TXD	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	UART1_RXD	GPIO_32	25	26	GPIO_61
GPIO_115	27	28	GPIO_113	GPIO_86	27	28	GPIO_88
GPIO_111	29	30	GPIO_112	GPIO_87	29	30	GPIO_89
GPIO_110	31	32	VDD_ADC	UART5_CTSN+	31	32	UART5_RTSN
AIN4	33	34	GNDA_ADC	UART4_RTSN	33	34	UART3_RTSN
AIN6	35	36	AIN5	UART4_CTSN	35	36	UART3_CTSN
AIN2	37	38	AIN3	UARR5_TXD+	37	38	UART5_RXD+
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	UART3_TXD	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Fig. 15.21: Table of UART outputs

Measuring a Temperature

Problem You want to measure a temperature using a digital temperature sensor.

Solution The TMP101 sensor is a common digital temperature sensor that uses a standard I^{2}C-based serial protocol.

To make this recipe, you will need:

- Breadboard and jumper wires.
- Two 4.7 kΩ resistors.
- TMP101 temperature sensor.

Wire the TMP101, as shown in *Wiring an I2C TMP101 temperature sensor*.

There are two I^{2}C buses brought out to the headers. *Table of I2C outputs* shows that you have wired your device to I^{2}C bus *2*.

Once the I^{2}C device is wired up, you can use a couple handy I^{2}C tools to test the device. Because these are Linux command-line tools, you have to use *2* as the bus number. *i2cdetect*, shown in *I2C tools*, shows which I^{2}C devices are on the bus. The *-r* flag indicates which bus to use. Our TMP101 is appearing at address *0x49*. You can use the *i2cget* command to read the value. It returns the temperature in hexadecimal and degrees C. In this example, 0x18 = 24{deg}C, which is 75.2{deg}F. (Hmmm, the office is a bit warm today.) Try warming up the TMP101 with your finger and running *i2cget* again.

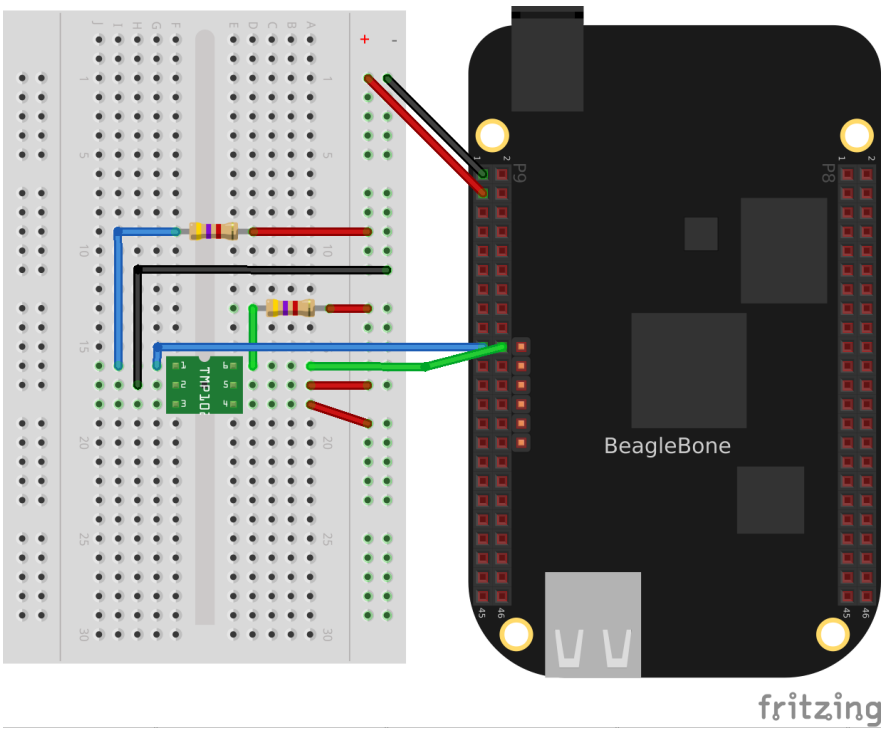


Fig. 15.22: Wiring an I^{2}C TMP101 temperature sensor

2 I2C ports

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_50	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
I2C1_SCL	17	18	I2C1_SDA	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
I2C2_SCL	21	22	I2C2_SDA	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	I2C1_SCL	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	I2C1_SDA	GPIO_32	25	26	GPIO_61
GPIO_115	27	28	GPIO_113	GPIO_86	27	28	GPIO_88
GPIO_111	29	30	GPIO_112	GPIO_87	29	30	GPIO_89
GPIO_110	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Fig. 15.23: Table of I^{2}C outputs

I²C tools

```
bone$ i2cdetect -y -r 2
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 49 -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --

bone$ i2cget -y 2 0x49
  0x18
```

Reading the temperature via the kernel driver

The cleanest way to read the temperature from at TMP101 sensor is to use the kernel driver.

Assuming the TMP101 is on bus 2 (the last digit is the bus number)

I²C TMP101 via Kernel

```
bone$ cd /sys/class/i2c-adapter/
bone$ ls
i2c-0  i2c-1  i2c-2                    # Three i2c buses (bus 0 is internal)
bone$ cd i2c-2        # Pick bus 2
bone$ ls -ls
0 --w--w---- 1 root gpio 4096 Jul  1 09:24 delete_device
0 lrwxrwxrwx 1 root gpio    0 Jun 30 16:25 device -> ../../4819c000.i2c
0 drwxrwxr-x 3 root gpio    0 Dec 31  1999 i2c-dev
0 -r--r--r-- 1 root gpio 4096 Dec 31  1999 name
0 --w--w---- 1 root gpio 4096 Jul  1 09:24 new_device
0 lrwxrwxrwx 1 root gpio    0 Jun 30 16:25 of_node -> ../../../../../../../..
↪/firmware/devicetree/base/ocp/interconnect@48000000/segment@100000/target-
↪module@9c000/i2c@0
0 drwxrwxr-x 2 root gpio    0 Dec 31  1999 power
0 lrwxrwxrwx 1 root gpio    0 Jun 30 16:25 subsystem -> ../../../../../../../
↪../bus/i2c
0 -rw-rw-r-- 1 root gpio 4096 Dec 31  1999 uevent
```

Assuming the TMP101 is at address 0x49

```
bone$ echo tmp101 0x49 > new_device
```

This tells the kernel you have a TMP101 sensor at address 0x49. Check the log to be sure.

```
bone$ dmesg -H | tail -3
[ +13.571823] i2c i2c-2: new_device: Instantiated device tmp101 at 0x49
[  +0.043362] lm75 2-0049: supply vs not found, using dummy regulator
[  +0.009976] lm75 2-0049: hwmon0: sensor 'tmp101'
```

Yes, it's there, now see what happened.

```
bone$ ls
2-0049  delete_device  device  i2c-dev  name new_device  of_node  power ␣
↪subsystem  uevent
```

Notice a new directory has appeared. It's for i2c bus 2, address 0x49. Look into it.

```
bone$ cd 2-0049/hwmon/hwmon0
bone$ ls -F
device@  name  power/  subsystem@  temp1_input  temp1_max temp1_max_hyst ␣
↪uevent  update_interval
bone$ cat temp1_input
24250
```

There is the temperature in milli-degrees C.

Other i2c devices are supported by the kernel. You can try the Linux Kernel Driver Database, https://cateee.net/lkddb/ to see them.

Once the driver is in place, you can read it via code. *Reading an I2C device (i2cTemp.py)* shows how to read the TMP101.

Python

JavaScript

Listing 15.11: Reading an I^{2}C device (i2cTemp.py)

```
#!/usr/bin/env python
# ////////////////////////////////////
# //        i2cTemp.py
# //      Read a TMP101 sensor on i2c bus 2, address 0x49
# //        Wiring:        Attach to i2c as shown in text.
# //        Setup:        echo tmp101 0x49 > /sys/class/i2c-adapter/i2c-2/
↪new_device
# //        See:
# ////////////////////////////////////
import time

ms = 1000   # Read time in ms
bus = '2'
addr = '49'
I2CPATH='/sys/class/i2c-adapter/i2c-'+bus+'/'+bus+'-00'+addr+'/hwmon/hwmon0';

f = open(I2CPATH+"/temp1_input", "r")

while True:
    f.seek(0)
    data = f.read()[:-1]    # returns mili-degrees C
    print("data (C) = " + str(int(data)/1000))
    time.sleep(ms/1000)
```

i2cTemp.py

Listing 15.12: Reading an I^{2}C device (i2cTemp.js)

```
#!/usr/bin/env node
////////////////////////////////////
//        i2cTemp.js
//      Read at TMP101 sensor on i2c bus 2, address 0x49
//        Wiring:        Attach to i2c as shown in text.
//        Setup:        echo tmp101 0x49 > /sys/class/i2c-adapter/i2c-2/new_
↪device
//        See:
////////////////////////////////////
const fs = require("fs");

const ms = 1000;   // Read time in ms
const bus = '2';
const addr = '49';
I2CPATH='/sys/class/i2c-adapter/i2c-'+bus+'/'+bus+'-00'+addr+'/hwmon/hwmon0';
```

(continues on next page)

(continued from previous page)

```
15
16 // Read every ms
17 setInterval(readTMP, ms);
18
19 function readTMP() {
20     var data = fs.readFileSync(I2CPATH+"/temp1_input").slice(0, -1);
21     console.log('data (C) = ' + data/1000);
22  }
```

i2cTemp.js

Run the code by using the following command:

```
bone$ ./i2cTemp.js
data (C) = 25.625
data (C) = 27.312
data (C) = 28.187
data (C) = 28.375
^C
```

Notice using the kernel interface gets you more digits of accuracy.

Reading i2c device directly

The TMP102 sensor can be read directly with i2c commands rather than using the kernel driver. First you need to install the i2c module.

```
bone$ pip install smbus
```

Listing 15.13: Reading an I^{2}C device (i2cTemp.py)

```
1  #!/usr/bin/env python
2  # ////////////////////////////////////////
3  # //        i2ctmp101.py
4  # //      Read at TMP101 sensor on i2c bus 2, address 0x49
5  # //        Wiring:        Attach to i2c as shown in text.
6  # //        Setup:        pip install smbus
7  # //        See:
8  # ////////////////////////////////////////
9  import smbus
10 import time
11
12 ms = 1000               # Read time in ms
13 bus = smbus.SMBus(2)    # Using i2c bus 2
14 addr = 0x49             # TMP101 is at address 0x49
15
16 while True:
17     data = bus.read_byte_data(addr, 0)
18     print("temp (C) = " + str(data))
19     time.sleep(ms/1000)
```

i2ctmp101.py

This gets only 8 bits for the temperature. See the TMP101 datasheet (https://www.ti.com/product/TMP101) for details on how to get up to 12 bits.

Reading Temperature via a Dallas 1-Wire Device

Problem You want to measure a temperature using a Dallas Semiconductor DS18B20 temperature sensor.

Solution The DS18B20 is an interesting temperature sensor that uses Dallas Semiconductor's 1-wire interface. The data communication requires only one wire! (However, you still need wires from ground and 3.3 V.) You can wire it to any GPIO port.

To make this recipe, you will need:

- Breadboard and jumper wires.
- 4.7 kΩ resistor
- DS18B20 1-wire temperature sensor.

Wire up as shown in *Wiring a Dallas 1-Wire temperature sensor*.

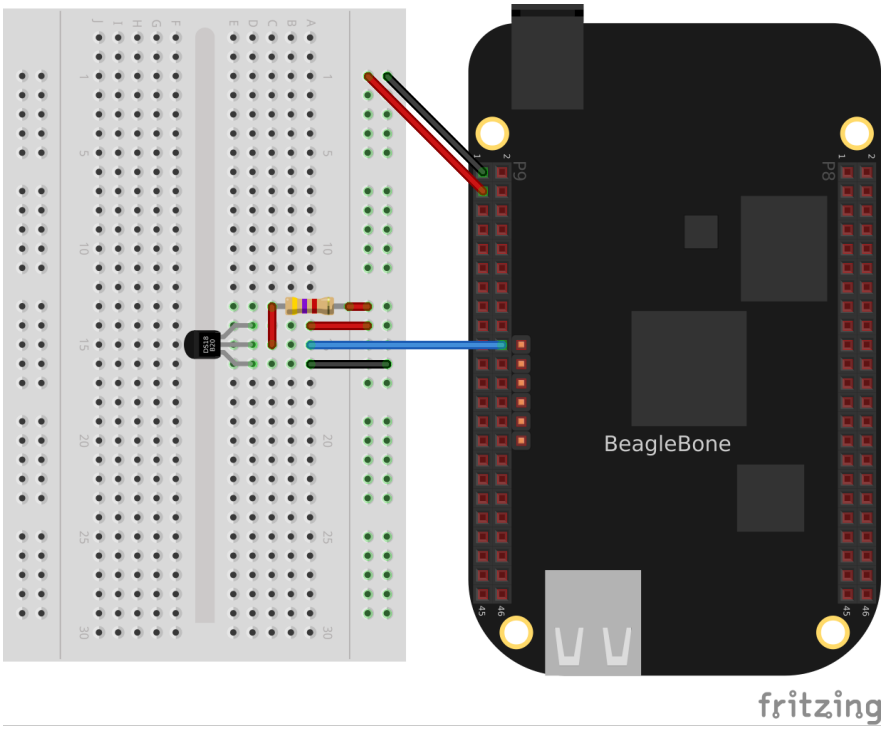


Fig. 15.24: Wiring a Dallas 1-Wire temperature sensor

Edit the file */boot/uEnt.txt*. Go to about line 19 and edit as shown:

```
17 ###
18 ###Additional custom capes
19 uboot_overlay_addr4=BB-W1-P9.12-00A0.dtbo
20 #uboot_overlay_addr5=<file5>.dtbo
```

Be sure to remove the # at the beginning of the line.

Reboot the bone:

```
bone$ reboot
```

Now run the following command to discover the serial number on your device:

```
bone$ ls /sys/bus/w1/devices/
28-00000114ef1b  28-00000128197d  w1_bus_master1
```

I have two devices wired in parallel on the same P9_12 input. This shows the serial numbers for all the devices.

Finally, add the code in *Reading a temperature with a DS18B20 (w1.py)* in to a file named *w1.py*, edit the path assigned to *w1* so that the path points to your device, and then run it.

Python

JavaScript

Listing 15.14: Reading a temperature with a DS18B20 (w1.py)

```
#!/usr/bin/env python
# ////////////////////////////////////
# //        w1.js
# //      Read a Dallas 1-wire device on P9_12
# //        Wiring:        Attach gnd and 3.3V and data to P9_12
# //        Setup:        Edit /boot/uEnv.txt to include:
# //              uboot_overlay_addr4=BB-W1-P9.12-00A0.dtbo
# //        See:
# ////////////////////////////////////
import time

ms = 500   # Read time in ms
#  Do ls /sys/bus/w1/devices and find the address of your device
addr = '28-00000d459c2c' # Must be changed for your device.
W1PATH ='/sys/bus/w1/devices/' + addr

f = open(W1PATH+'/temperature')

while True:
    f.seek(0)
    data = f.read()[:-1]
    print("temp (C) = " + str(int(data)/1000))
    time.sleep(ms/1000)
```

w1.py

Listing 15.15: Reading a temperature with a DS18B20 (w1.js)

```
#!/usr/bin/env node
////////////////////////////////////
//        w1.js
//      Read a Dallas 1-wire device on P9_12
//        Wiring:        Attach gnd and 3.3V and data to P9_12
//        Setup:        Edit /boot/uEnv.txt to include:
//              uboot_overlay_addr4=BB-W1-P9.12-00A0.dtbo
//        See:
////////////////////////////////////
const fs = require("fs");

const ms = 500   // Read time in ms
// Do ls /sys/bus/w1/devices and find the address of your device
const addr = '28-00000d459c2c'; // Must be changed for your device.
const W1PATH ='/sys/bus/w1/devices/' + addr;

// Read every ms
setInterval(readW1, ms);

function readW1() {
    var data = fs.readFileSync(W1PATH+'/temperature').slice(0, -1);
    console.log('temp (C) = ' + data/1000);
 }
```

w1.js

```
bone$ ./w1.js
temp (C) = 28.625
temp (C) = 29.625
temp (C) = 30.5
temp (C) = 31.0
^C
```

Each temperature sensor has a unique serial number, so you can have several all sharing the same data line.

Playing and Recording Audio

Problem BeagleBone doesn't have audio built in, but you want to play and record files.

Solution One approach is to buy an audio cape, but another, possibly cheaper approach is to buy a USB audio adapter, such as the one shown in *A USB audio dongle*.



Fig. 15.25: A USB audio dongle

Drivers for the Advanced Linux Sound Architecture (ALSA) are already installed on the Bone. You can list the recording and playing devices on your Bone by using *aplay* and *arecord*, as shown in *Listing the ALSA audio output and input devices on the Bone*. BeagleBone Black has audio-out on the HDMI interface. It's listed as *card 0* in *Listing the ALSA audio output and input devices on the Bone*. *card 1* is my USB audio adapter's audio out.

Listing the ALSA audio output and input devices on the Bone

```
bone$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: Black [TI BeagleBone Black], device 0: HDMI nxp-hdmi-hifi-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: Device [C-Media USB Audio Device], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

(continues on next page)

(continued from previous page)

```
bone$ arecord -l
**** List of CAPTURE Hardware Devices ****
card 1: Device [C-Media USB Audio Device], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

In the *aplay* output shown in *Listing the ALSA audio output and input devices on the Bone*, you can see the USB adapter's audio out. By default, the Bone will send audio to the HDMI. You can change that default by creating a file in your home directory called *~/.asoundrc* and adding the code in *Change the default audio out by putting this in ~/.asoundrc (audio.asoundrc)* to it.

Listing 15.16: Change the default audio out by putting this in ~/.asoundrc (audio.asoundrc)

```
pcm.!default {
  type plug
  slave {
    pcm "hw:1,0"
  }
}
ctl.!default {
  type hw
  card 1
}
```

`audio.asoundrc`

You can easily play `.wav` files with *aplay*:

```
bone$ aplay test.wav
```

You can play other files in other formats by installing *mplayer*:

```
bone$ sudo apt update
bone$ sudo apt install mplayer
bone$ mplayer test.mp3
```

Discussion Adding the simple USB audio adapter opens up a world of audio I/O on the Bone.

15.1.3 Displays and Other Outputs

In this chapter, you will learn how to control physical hardware via BeagleBone Black's general-purpose input/output (GPIO) pins. The Bone has 65 GPIO pins that are brought out on two 46-pin headers, called *P8* and *P9*, as shown in *The P8 and P9 GPIO headers*.

Note: All the examples in the book assume you have cloned the Cookbook repository on git.beagleboard.org. Go here *Cloning the Cookbook Repository* for instructions.

The purpose of this chapter is to give simple examples that show how to use various methods of output. Most solutions require a breadboard and some jumper wires.

All these examples assume that you know how to edit a file (*Editing Code Using Visual Studio Code*) and run it, either within Visual Studio Code (VSC) integrated development environment (IDE) or from the command line (*Getting to the Command Shell via SSH*).

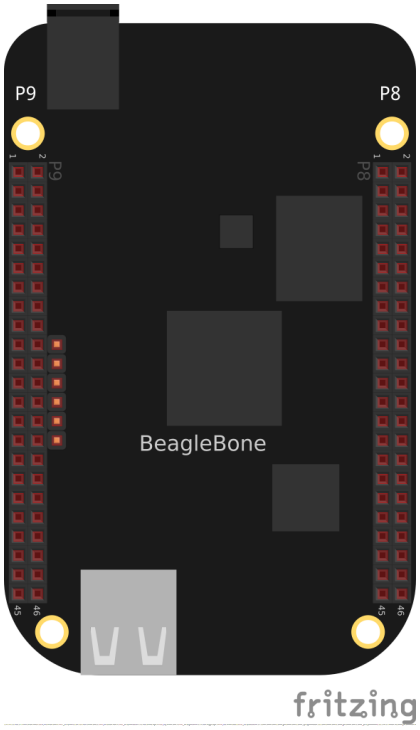


Fig. 15.26: The P8 and P9 GPIO headers

Toggling an Onboard LED

Problem You want to know how to flash the four LEDs that are next to the Ethernet port on the Bone.

Solution Locate the four onboard LEDs shown in *The four USER LEDs*. They are labeled *USR0* through *USR3*, but we'll refer to them as the *USER* LEDs.

Place the code shown in *Using an internal LED (internLED.py)* in a file called `internLED.py`. You can do this using VSC to edit files (as shown in *Editing Code Using Visual Studio Code*) or with a more traditional editor (as shown in *Editing a Text File from the GNU/Linux Command Shell*).

Python

C

Listing 15.17: Using an internal LED (internLED.py)

```
#!/usr/bin/env python
# ////////////////////////////////////
#          internLED.py
#          Blinks A USR LED.
#          Wiring:
#          Setup:
#          See:
# ////////////////////////////////////
import gpiod
import time

LED_CHIP = 'gpiochip1'
LED_LINE_OFFSET = [21]  # USR0 run: gpioinfo | grep -i -e chip -e usr

chip = gpiod.Chip(LED_CHIP)

lines = chip.get_lines(LED_LINE_OFFSET)
```

(continues on next page)

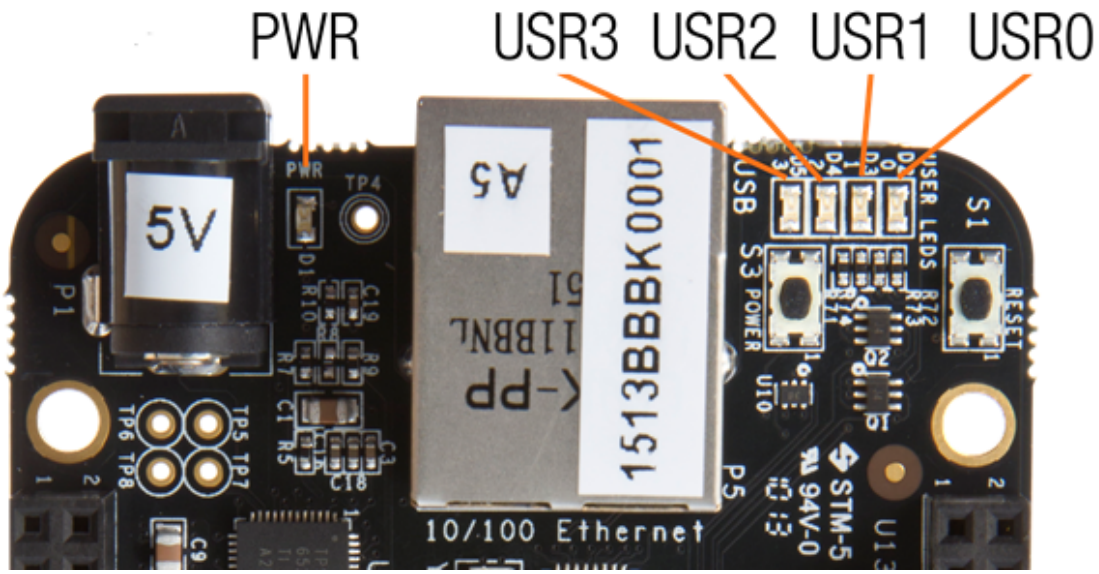


Fig. 15.27: The four *USER* LEDs

(continued from previous page)

```
lines.request(consumer='internLED.py', type=gpiod.LINE_REQ_DIR_OUT)

state =  0       # Start with LED off
while True:
    lines.set_values([state])
    state = ~state      # Toggle the state
    time.sleep(0.25)
```

internLED.py

Listing 15.18: Using an internal LED (internLED.c)

```
// // ////////////////////////////////////
// #          internLED.c
// #          Blinks A USR LED.
// #          Wiring:
// #          Setup:
// #          See:
// // ////////////////////////////////////
#include <gpiod.h>
#include <stdio.h>
#include <unistd.h>

#define         CONSUMER        "internLED.c"

int main(int argc, char **argv)
{
        int chipnumber = 1;
        unsigned int line_num = 21;         // usr0 LED, run: gpioinfo | grep -
↪i -e chip -e usr
        unsigned int val;
        struct gpiod_chip *chip;
        struct gpiod_line *line;
        int i, ret;

        chip = gpiod_chip_open_by_number(chipnumber);
        line = gpiod_chip_get_line(chip, line_num);
        ret = gpiod_line_request_output(line, CONSUMER, 0);
```

(continues on next page)

(continued from previous page)

```
26
27          /* Blink */
28          val = 0;
29          while(1) {
30                  ret = gpiod_line_set_value(line, val);
31                  // printf("Output %u on line #%u\n", val, line_num);
32                  usleep(100000);                 // Number of microseconds to␣
 ↪sleep
33                  val = !val;
34          }
35  }
```

internLED.c

In the *bash* command window, enter the following commands:

```
bone$ cd ~/beaglebone-cookbook-code/03displays
bone$ ./internLED.py
```

The *USER0* LED should now be flashing.

Toggling an External LED

Problem You want to connect your own external LED to the Bone.

Solution Connect an LED to one of the GPIO pins using a series resistor to limit the current. To make this recipe, you will need:

- Breadboard and jumper wires.
- 220 Ω to 470 Ω resistor.
- LED

Warning: The value of the current limiting resistor depends on the LED you are using. The Bone can drive only 4 to 6 mA, so you might need a larger resistor to keep from pulling too much current. A 330 Ω or 470 Ω resistor might be better.

Diagram for using an external LED shows how you can wire the LED to pin 14 of the *P9* header (*P9_14*). Every circuit in this book (*Wiring a Breadboard*) assumes you have already wired the rightmost bus to ground (*P9_1*) and the next bus to the left to the 3.3 V (*P9_3*) pins on the header. Be sure to get the polarity right on the LED. The _short_ lead always goes to ground.

After you've wired it, start VSC (see *Editing Code Using Visual Studio Code*) and find the code shown in *Code for using an external LED (externLED.py)*. Notice that it looks very similar to the *internLED* code, in fact it only differs in the line number (18 instead of 21). The built-in LEDs use the same GPIO interface as the GPIO pins.

Python

C

Listing 15.19: Code for using an external LED (externLED.py)

```
1  #!/usr/bin/env python
2  # ////////////////////////////////////
3  #          externLED.py
4  #          Blinks an external LED wired to P9_14.
5  #          Wiring: P9_14 connects to the plus lead of an LED.  The negative␣
 ↪lead of the
6  #                  LED goes to a 220 Ohm resistor.  The other lead of the␣
 ↪resistor goes
```

(continues on next page)

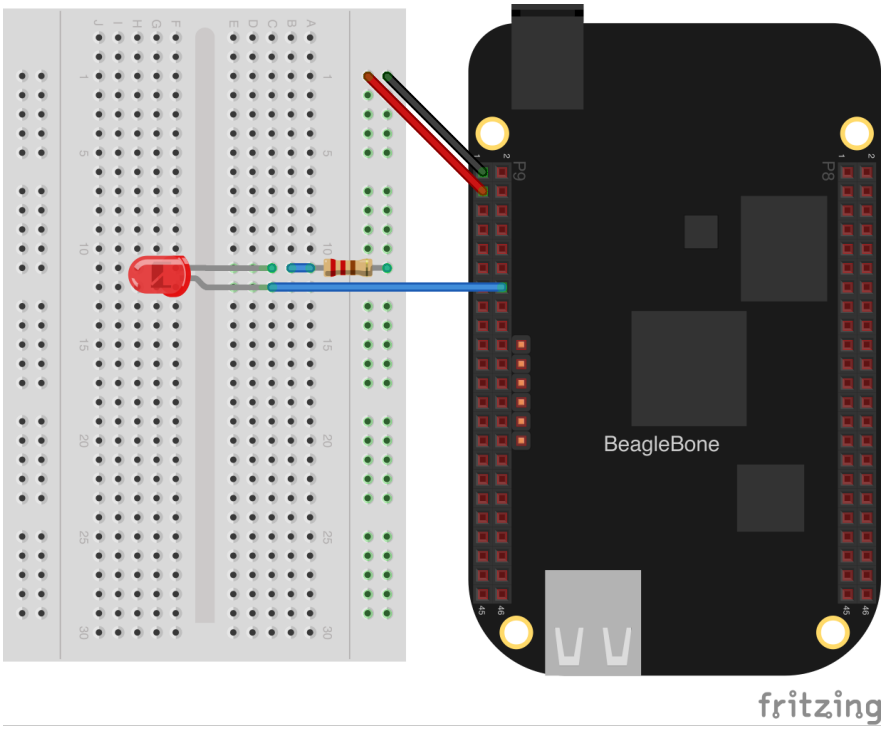


Fig. 15.28: Diagram for using an external LED

(continued from previous page)

```
#                to ground
#         Setup:
#         See:
# ////////////////////////////////////
import gpiod
import time

LED_CHIP = 'gpiochip1'
LED_LINE_OFFSET = [18]  # P9_14 run: gpioinfo | grep -i -e chip -e P9_14

chip = gpiod.Chip(LED_CHIP)

lines = chip.get_lines(LED_LINE_OFFSET)
lines.request(consumer='internLED.py', type=gpiod.LINE_REQ_DIR_OUT)

state =  0      # Start with LED off
while True:
    lines.set_values([state])
    state = ~state      # Toggle the state
    time.sleep(0.25)
```

externLED.py

Listing 15.20: Code for using an external LED (externLED.c)

```
// // ////////////////////////////////////
// #         externLED.c
// Blinks an external LED wired to P9_14.
// Wiring: P9_14 connects to the plus lead of an LED.  The negative lead of␣
↪the
//                 LED goes to a 220 Ohm resistor.  The other lead of the␣
↪resistor goes
//                 to ground
```

(continues on next page)

(continued from previous page)

```
//          Setup:
//          See:
// // ////////////////////////////////////
#include <gpiod.h>
#include <stdio.h>
#include <unistd.h>

#define         CONSUMER        "internLED.c"

int main(int argc, char **argv)
{
        int chipnumber = 1;
        unsigned int line_num = 18;         // P9_14, run: gpioinfo | grep -i -e chip -e P9_14
        unsigned int val;
        struct gpiod_chip *chip;
        struct gpiod_line *line;
        int i, ret;

        chip = gpiod_chip_open_by_number(chipnumber);
        line = gpiod_chip_get_line(chip, line_num);
        ret = gpiod_line_request_output(line, CONSUMER, 0);

        /* Blink */
        val = 0;
        while(1) {
                ret = gpiod_line_set_value(line, val);
                // printf("Output %u on line #%u\n", val, line_num);
                usleep(100000);                 // Number of microseconds to sleep
                val = !val;
        }
}
```

externLED.c

Save your file and run the code as before (*Toggling an Onboard LED*).

Toggling a High-Voltage External Device

Problem You want to control a device that runs at 120 V.

Solution Working with 120 V can be tricky –even dangerous– if you aren't careful. Here's a safe way to do it.

To make this recipe, you will need:

- PowerSwitch Tail II

Diagram for wiring PowerSwitch Tail II shows how you can wire the PowerSwitch Tail II to pin *P9_14*.

After you've wired it, because this uses the same output pin as *Toggling an External LED*, you can run the same code (*Code for using an external LED (externLED.py)*).

Fading an External LED

Problem You want to change the brightness of an LED from the Bone.

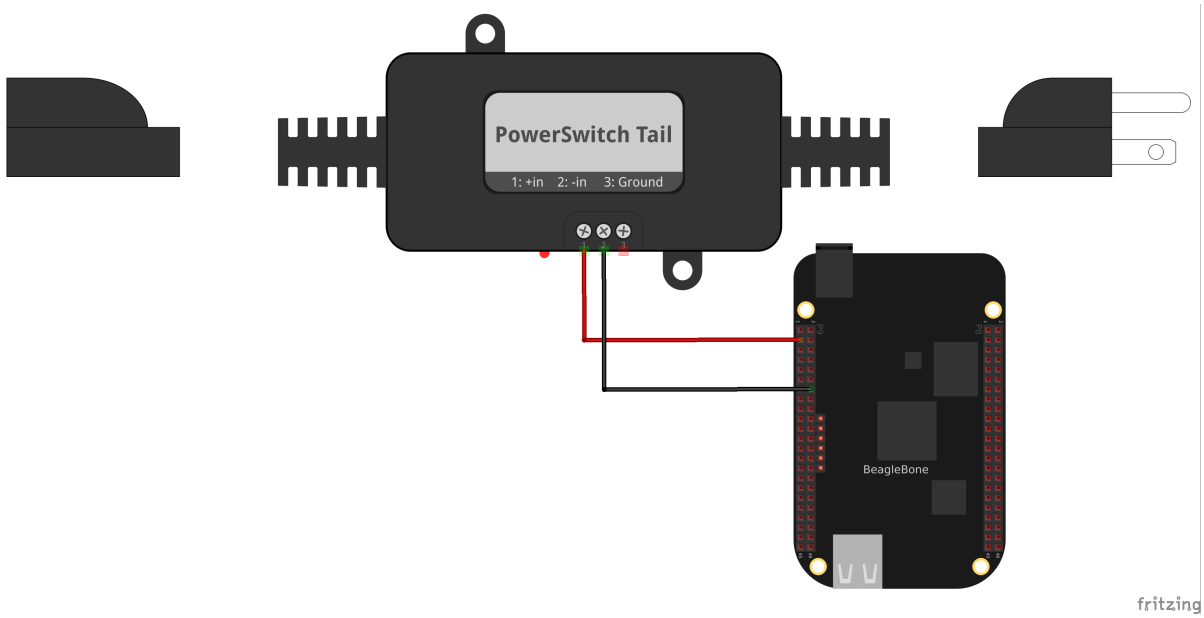


Fig. 15.29: Diagram for wiring PowerSwitch Tail II

Solution Use the Bone's pulse width modulation (PWM) hardware to fade an LED. We'll use the same circuit as before (*Diagram for using an external LED*). Find the code in *Code for using an external LED (fadeLED.py)* Next configure the pins. We are using P9_14 so run:

```
bone$ config-pin P9_14 pwm
```

Then run it as before.

Python

JavaScript

Listing 15.21: Code for using an external LED (fadeLED.py)

```
#!/usr/bin/env python
# ////////////////////////////////////
# //        fadeLED.py
# //        Blinks the P9_14 pin
# //        Wiring:
# //        Setup:  config-pin P9_14 pwm
# //        See:
# ////////////////////////////////////
import time
ms = 20;    # Fade time in ms

pwmPeriod = 1000000    # Period in ns
pwm     = '1'  # pwm to use
channel = 'a'  # channel to use
PWMPATH='/dev/bone/pwm/'+pwm+'/'+channel
step = 0.02    # Step size
min = 0.02     # dimmest value
max = 1        # brightest value
brightness = min # Current brightness

f = open(PWMPATH+'/period', 'w')
f.write(str(pwmPeriod))
f.close()

f = open(PWMPATH+'/enable', 'w')
```

(continues on next page)

(continued from previous page)

```
f.write('1')
f.close()

f = open(PWMPATH+'/duty_cycle', 'w')
while True:
    f.seek(0)
    f.write(str(round(pwmPeriod*brightness)))
    brightness  += step
    if(brightness >= max or brightness <= min):
        step = -1 * step
    time.sleep(ms/1000)

# | Pin   | pwm | channel
# | P9_31 | 0   | a
# | P9_29 | 0   | b
# | P9_14 | 1   | a
# | P9_16 | 1   | b
# | P8_19 | 2   | a
# | P8_13 | 2   | b
```

fadeLED.py

Listing 15.22: Code for using an external LED (fadeLED.js)

```
#!/usr/bin/env node
////////////////////////////////////////
//        fadeLED.js
//        Blinks the P9_14 pin
//        Wiring:
//        Setup:  config-pin P9_14 pwm
//        See:
////////////////////////////////////////
const fs = require("fs");
const ms = '20';    // Fade time in ms

const pwmPeriod = '1000000';    // Period in ns
const pwm     = '1';  // pwm to use
const channel = 'a';  // channel to use
const PWMPATH='/dev/bone/pwm/'+pwm+'/'+channel;
var   step = 0.02;  // Step size
const min = 0.02,     // dimmest value
    max = 1;        // brightest value
var brightness = min; // Current brightness;


// Set the period in ns
fs.writeFileSync(PWMPATH+'/period', pwmPeriod);
fs.writeFileSync(PWMPATH+'/duty_cycle', pwmPeriod/2);
fs.writeFileSync(PWMPATH+'/enable', '1');

setInterval(fade, ms);      // Step every  ms

function fade() {
     fs.writeFileSync(PWMPATH+'/duty_cycle',
        parseInt(pwmPeriod*brightness));
    brightness += step;
    if(brightness >= max || brightness <= min) {
        step = -1 * step;
    }
}

// | Pin   | pwm | channel
```

(continues on next page)

(continued from previous page)

```
// | P9_31 | 0   | a
// | P9_29 | 0   | b
// | P9_14 | 1   | a
// | P9_16 | 1   | b
// | P8_19 | 2   | a
// | P8_13 | 2   | b
```

`fadeLED.js`

The Bone has several outputs that can be use as pwm's as shown in *Table of PWM outputs*. There are three *EHRPWM's* which each has a pair of pwm channels. Each pair must have the same period.

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
EHRPWM0B	21	22	EHRPWM0A
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	ECAPPWM2
EHRPWM0B	29	30	GPIO_112
EHRPWM0A	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPPWM0
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
TIMER4	7	8	TIMER7
TIMER5	9	10	TIMER6
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	EHRPWM1B
GPIO_8	35	36	EHRPWM1A
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
EHRPWM2A	45	46	EHRPWM2B

Fig. 15.30: Table of PWM outputs

The pwm's are accessed through */dev/bone/pwm*

```
bone$ cd /dev/bone/pwm
bone$ ls
0  1  2
```

Here we see three pwmchips that can be used, each has two channels. Explore one.

```
bone$ cd 1
bone$ ls
a  b
bone$ cd a
bone$ ls
capture  duty_cycle  enable  period  polarity  power  uevent
```

Here is where you can set the period and duty_cycle (in ns) and enable the pwm. Attach in LED to P9_14 and if you set the period long enough you can see the LED flash.

```
bone$ echo 1000000000 > period
bone$ echo  500000000 > duty_cycle
bone$ echo 1 > enable
```

Your LED should now be flashing.

Headers to pwm channel mapping are the mapping I've figured out so far. I don't know how to get to the timers.

Table 15.3: Headers to pwm channel mapping

Pin	pwm	channel
P9_31	0	a
P9_29	0	b
P9_14	1	a
P9_16	1	b
P8_19	2	a
P8_13	2	b

Writing to an LED Matrix

Problem You have an I^{2}C-based LED matrix to interface.

Solution There are a number of nice LED matrices that allow you to control several LEDs via one interface. This solution uses an Adafruit Bicolor 8x8 LED Square Pixel Matrix w/|I2C| Backpack.

To make this recipe, you will need:

- Breadboard and jumper wires
- Two 4.7 kΩ resistors.
- I^{2}C LED matrix

The LED matrix is a 5 V device, but you can drive it from 3.3 V. Wire, as shown in *Wiring an I2C LED matrix*.

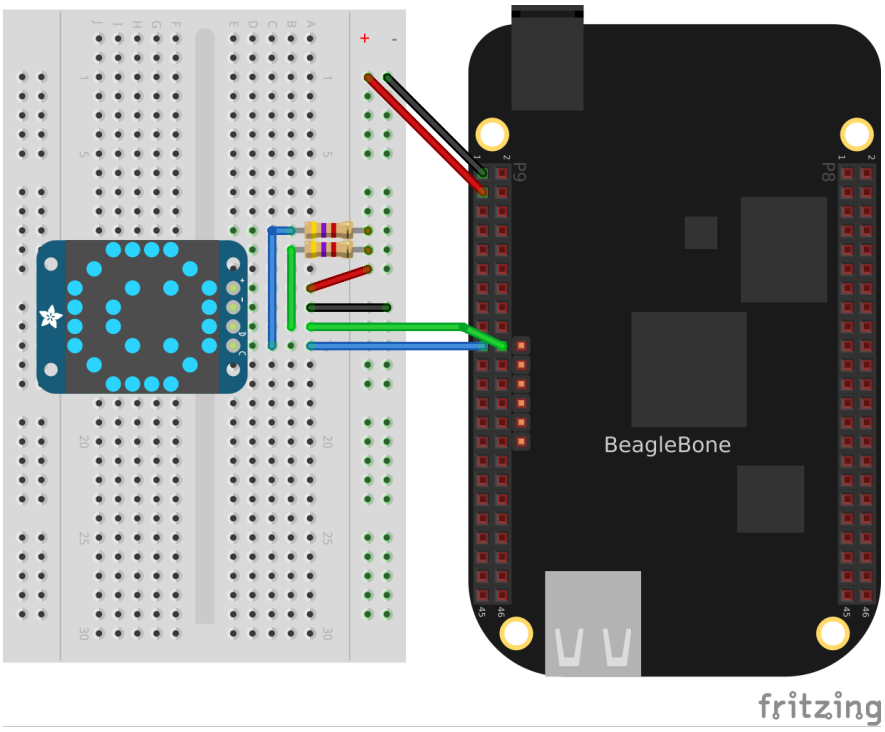


Fig. 15.31: Wiring an I^{2}C LED matrix

Measuring a Temperature shows how to use *i2cdetect* to discover the address of an I^{2}C device.

Run the *i2cdetect -y -r 2* command to discover the address of the display on I^{2}C bus 2, as shown in *Using I2C command-line tools to discover the address of the display*.

Using I²C command-line tools to discover the address of the display

```
bone$ i2cdetect -y -r 2
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- 49 -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

Here, you can see a device at *0x49* and *0x70*. I know I have a temperature sensor at *0x49*, so the LED matrix must be at *0x70*.

Find the code in *LED matrix display (matrixLEDi2c.py)* and run it by using the following command:

```
bone$ pip install smbus  # (Do this only once.)
bone$ ./matrixLEDi2c.py
```

LED matrix display (matrixLEDi2c.py)

Listing 15.23: LED matrix display (matrixLEDi2c.py)

```
#!/usr/bin/env python
# ////////////////////////////////////////
# //        i2cTemp.py
# //      Write an 8x8 Red/Green LED matrix.
# //        Wiring:        Attach to i2c as shown in text.
# //        Setup:        echo tmp101 0x49 > /sys/class/i2c-adapter/i2c-2/
 ↪new_device
# //        See:        https://www.adafruit.com/product/902
# ////////////////////////////////////////
import smbus
import time

bus = smbus.SMBus(2)  # Use i2c bus 2       ①
matrix = 0x70         # Use address 0x70    ②
ms = 1;               # Delay between images in ms

# The first byte is GREEN, the second is RED.   ③
smile = [0x00, 0x3c, 0x00, 0x42, 0x28, 0x89, 0x04, 0x85,
    0x04, 0x85, 0x28, 0x89, 0x00, 0x42, 0x00, 0x3c
]
frown = [0x3c, 0x00, 0x42, 0x00, 0x85, 0x20, 0x89, 0x00,
    0x89, 0x00, 0x85, 0x20, 0x42, 0x00, 0x3c, 0x00
]
neutral = [0x3c, 0x3c, 0x42, 0x42, 0xa9, 0xa9, 0x89, 0x89,
    0x89, 0x89, 0xa9, 0xa9, 0x42, 0x42, 0x3c, 0x3c
]

bus.write_byte_data(matrix, 0x21, 0)   # Start oscillator (p10) ④
bus.write_byte_data(matrix, 0x81, 0)   # Disp on, blink off (p11)
bus.write_byte_data(matrix, 0xe7, 0)   # Full brightness (page 15)

bus.write_i2c_block_data(matrix, 0, frown)      # ⑤
for fade in range(0xef, 0xe0, -1):              # ⑥
    bus.write_byte_data(matrix, fade, 0)
    time.sleep(ms/10)

```

(continues on next page)

(continued from previous page)

```
bus.write_i2c_block_data(matrix, 0, neutral)
for fade in range(0xe0, 0xef, 1):
    bus.write_byte_data(matrix, fade, 0)
    time.sleep(ms/10)

bus.write_i2c_block_data(matrix, 0, smile)
```

`matrixLEDi2c.py`

① This line states which bus to use. The last digit gives the I^{2}C bus number.

② This specifies the address of the LED matrix, *0x70* in our case.

③ This indicates which LEDs to turn on. The first byte is for the first column of `green` LEDs. In this case, all are turned off. The next byte is for the first column of `red` LEDs. The hex *0x3c* number is *0b00111100* in binary. This means the first two red LEDs are off, the next four are on, and the last two are off. The next byte (*0x00*) says the second column of *green* LEDs are all off, the fourth byte (*0x42 = 0b01000010*) says just two *red* LEDs are on, and so on. Declarations define four different patterns to display on the LED matrix, the last being all turned off.

④ Send three commands to the matrix to get it ready to display.

⑤ Now, we are ready to display the various patterns. After each pattern is displayed, we sleep a certain amount of time so that the pattern can be seen.

⑥ Finally, send commands to the LED matrix to set the brightness. This makes the display fade out and back in again.

Driving a 5 V Device

Problem You have a 5 V device to drive, and the Bone has 3.3 V outputs.

Solution If you are lucky, you might be able to drive a 5 V device from the Bone's 3.3 V output. Try it and see if it works. If not, you need a level translator.

What you will need for this recipe:

- A PCA9306 level translator
- A 5 V power supply (if the Bone's 5 V power supply isn't enough)

The PCA9306 translates signals at 3.3 V to 5 V in both directions. It's meant to work with I^{2}C devices that have a pull-up resistor, but it can work with anything needing translation.

Wiring a PCA9306 level translator to an LED matrix shows how to wire a PCA9306 to an LED matrix. The left is the 3.3 V side and the right is the 5 V side. Notice that we are using the Bone's built-in 5 V power supply.

Note: If your device needs more current than the Bone's 5 V power supply provides, you can wire in an external power supply.

Writing to a NeoPixel LED String Using the PRUs

Problem You have an Adafruit NeoPixel LED string or Adafruit NeoPixel LED matrix and want to light it up.

Solution The PRU Cookbook has a nice discussion (WS2812 (NeoPixel) driver) on driving NeoPixels.

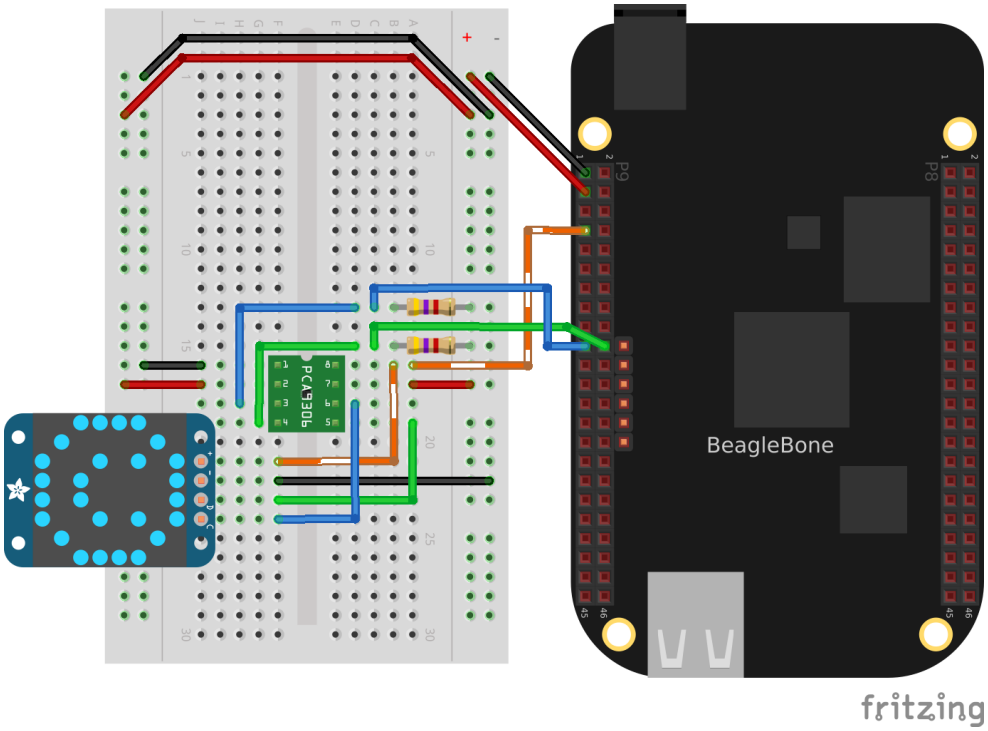


Fig. 15.32: Wiring a PCA9306 level translator to an LED matrix

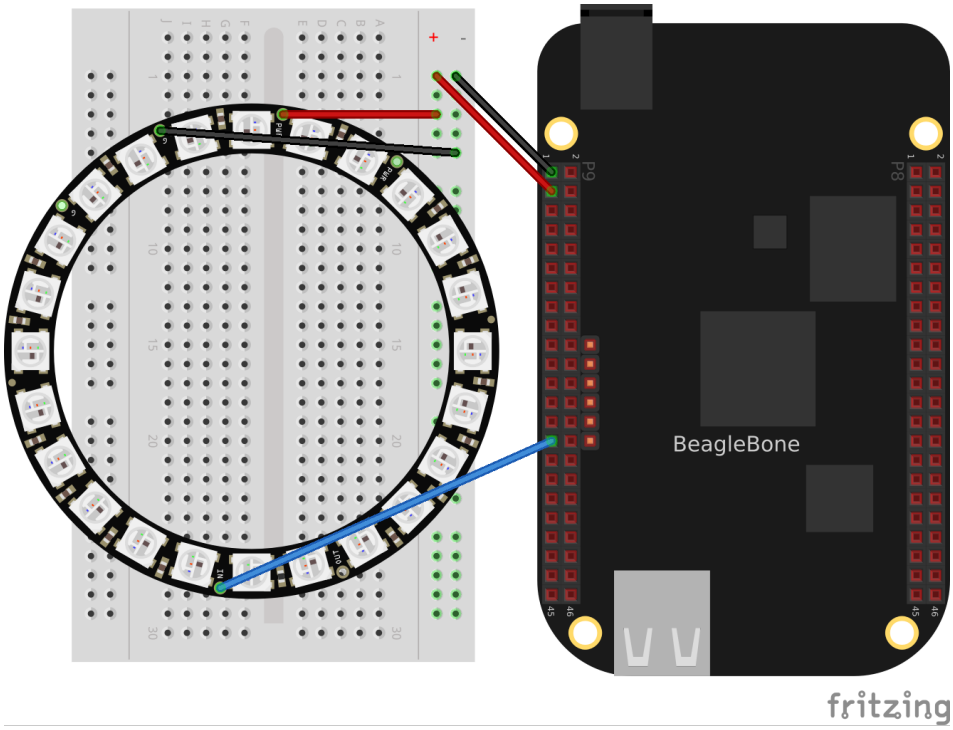


Fig. 15.33: Wiring an Adafruit NeoPixel LED matrix to *P9_29*

Writing to a NeoPixel LED String Using LEDscape

Making Your Bone Speak

Problem Your Bone wants to talk.

Solution Just install the `flite` text-to-speech program:

```
bone$ sudo apt install flite
```

Then add the code from *A program that talks (speak.js)* in a file called `speak.js` and run.

Listing 15.24: A program that talks (speak.js)

```
#!/usr/bin/env node

var exec = require('child_process').exec;

function speakForSelf(phrase) {
{
        exec('flite -t "' + phrase + '"', function (error, stdout, stderr) {
        console.log(stdout);
        if(error) {
            console.log('error: ' + error);
        }
        if(stderr) {
            console.log('stderr: ' + stderr);
        }
        });
}

speakForSelf("Hello, My name is Borris. " +
    "I am a BeagleBone Black, " +
    "a true open hardware, " +
    "community-supported embedded computer for developers and hobbyists. " +
    "I am powered by a 1 Giga Hertz Sitara™ ARM® Cortex-A8 processor. " +
    "I boot Linux in under 10 seconds. " +
    "You can get started on development in " +
    "less than 5 minutes with just a single USB cable." +
    "Bark, bark!"
    );
```

`speak.js`

See *Playing and Recording Audio* to see how to use a USB audio dongle and set your default audio out.

15.1.4 Motors

One of the many fun things about embedded computers is that you can move physical things with motors. But there are so many different kinds of motors (`servo`, `stepper`, `DC`), so how do you select the right one?

The type of motor you use depends on the type of motion you want:

- **R/C or hobby servo motor** Can be quickly positioned at various absolute angles, but some don't spin. In fact, many can turn only about 180{deg}.
- **Stepper motor** Spins and can also rotate in precise relative angles, such as turning 45°. Stepper motors come in two types: `bipolar` (which has four wires) and `unipolar` (which has five or six wires).
- **DC motor** Spins either clockwise or counter-clockwise and can have the greatest speed of the three. But a DC motor can't easily be made to turn to a given angle.

When you know which type of motor to use, interfacing is easy. This chapter shows how to interface with each of these motors.

Note: Motors come in many sizes and types. This chapter presents some of the more popular types and shows how they can interface easily to the Bone. If you need to turn on and off a 120 V motor, consider using something like the PowerSwitch presented in *Toggling a High-Voltage External Device*.

Note: The Bone has built-in 3.3 V and 5 V supplies, which can supply enough current to drive some small motors. Many motors, however, draw enough current that an external power supply is needed. Therefore, an external 5 V power supply is listed as optional in many of the recipes.

Note: All the examples in the book assume you have cloned the Cookbook repository on git.beagleboard.org. Go here *Cloning the Cookbook Repository* for instructions.

Controlling a Servo Motor

Problem You want to use BeagleBone to control the absolute position of a servo motor.

Solution We'll use the pulse width modulation (PWM) hardware of the Bone to control a servo motor.

To make the recipe, you will need:

- Servo motor.
- Breadboard and jumper wires.
- 1 kΩ resistor (optional)
- 5 V power supply (optional)

The 1 kΩ resistor isn't required, but it provides some protection to the general-purpose input/output (GPIO) pin in case the servo fails and draws a large current.

Wire up your servo, as shown in *Driving a servo motor with the 3.3 V power supply*.

Note: There is no standard for how servo motor wires are colored. One of my servos is wired like *Driving a servo motor with the 3.3 V power supply* red is 3.3 V, black is ground, and yellow is the control line. I have another servo that has red as 3.3 V and ground is brown, with the control line being orange. Generally, though, the 3.3 V is in the middle. Check the datasheet for your servo before wiring.

The code for controlling the servo motor is in `servoMotor.py`, shown in *Code for driving a servo motor (servoMotor.py)*. You need to configure the pin for PWM.

```
bone$ cd ~/beaglebone-cookbook-code/04motors
bone$ config-pin P9_16 pwm
bone$ ./servoMotor.py
```

Python

JavaScript

Listing 15.25: Code for driving a servo motor (servoMotor.py)

```
1 #!/usr/bin/env python
2 # ////////////////////////////////////
3 # //         servoMotor.py
```

(continues on next page)

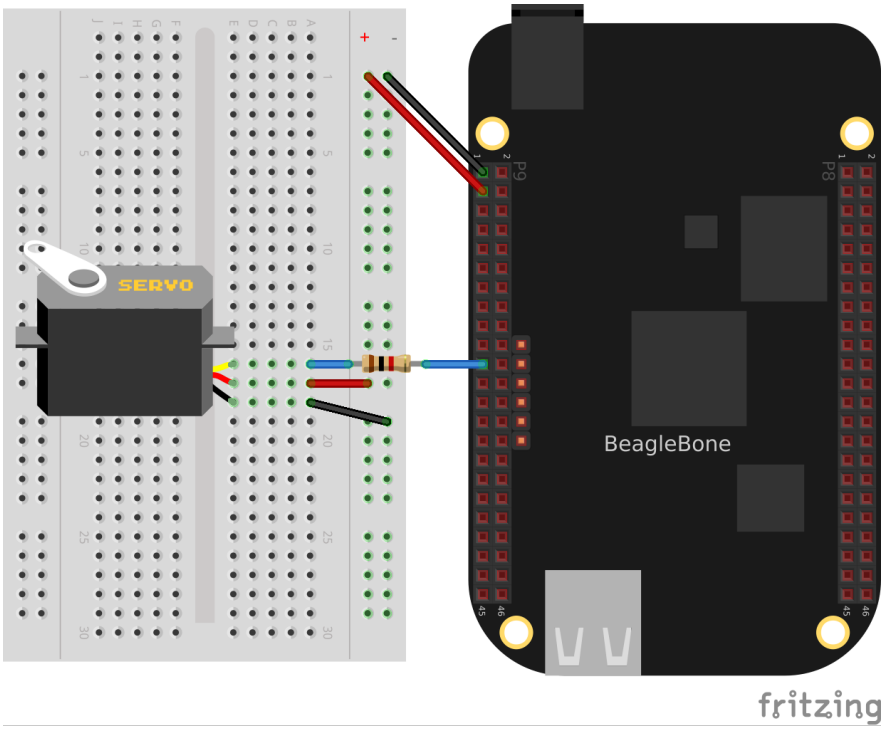


Fig. 15.34: Driving a servo motor with the 3.3 V power supply

(continued from previous page)

```
# //         Drive a simple servo motor back and forth on P9_16 pin
# //         Wiring:
# //         Setup:  config-pin P9_16 pwm
# //         See:
# ////////////////////////////////////////
import time
import signal
import sys

pwmPeriod = '20000000'     # Period in ns, (20 ms)
pwm =      '1' # pwm to use
channel = 'b' # channel to use
PWMPATH='/dev/bone/pwm/'+pwm+'/'+channel
low   = 0.8 # Smallest angle (in ms)
hi    = 2.4 # Largest angle (in ms)
ms    = 250 # How often to change position, in ms
pos   = 1.5 # Current position, about middle ms)
step  = 0.1 # Step size to next position

def signal_handler(sig, frame):
    print('Got SIGINT, turning motor off')
    f = open(PWMPATH+'/enable', 'w')
    f.write('0')
    f.close()
    sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)
print('Hit ^C to stop')

f = open(PWMPATH+'/period', 'w')
f.write(pwmPeriod)
f.close()
f = open(PWMPATH+'/enable', 'w')
f.write('1')
```

(continues on next page)

(continued from previous page)

```
f.close()

f = open(PWMPATH+'/duty_cycle', 'w')
while True:
    pos += step    # Take a step
    if(pos > hi or pos < low):
        step *= -1
    duty_cycle = str(round(pos*1000000))    # Convert ms to ns
    # print('pos = ' + str(pos) + ' duty_cycle = ' + duty_cycle)
    f.seek(0)
    f.write(duty_cycle)
    time.sleep(ms/1000)

# | Pin   | pwm | channel
# | P9_31 | 0   | a
# | P9_29 | 0   | b
# | P9_14 | 1   | a
# | P9_16 | 1   | b
# | P8_19 | 2   | a
# | P8_13 | 2   | b
```

servoMotor.py

Listing 15.26: Code for driving a servo motor (servoMotor.js)

```
#!/usr/bin/env node
////////////////////////////////////////
//        servoMotor.js
//        Drive a simple servo motor back and forth on P9_16 pin
//        Wiring:
//        Setup:  config-pin P9_16 pwm
//        See:
////////////////////////////////////////
const fs = require("fs");

const pwmPeriod = '20000000';    // Period in ns, (20 ms)
const pwm     = '1';  // pwm to use
const channel = 'b';  // channel to use
const PWMPATH='/dev/bone/pwm/'+pwm+'/'+channel;
const low  = 0.8, // Smallest angle (in ms)
      hi   = 2.4, // Largest angle (in ms)
      ms   = 250;  // How often to change position, in ms
var   pos = 1.5,  // Current position, about middle ms)
      step = 0.1; // Step size to next position

console.log('Hit ^C to stop');
fs.writeFileSync(PWMPATH+'/period', pwmPeriod);
fs.writeFileSync(PWMPATH+'/enable', '1');

var timer = setInterval(sweep, ms);

// Sweep from low to hi position and back again
function sweep() {
    pos += step;    // Take a step
    if(pos > hi || pos < low) {
        step *= -1;
    }
    var dutyCycle = parseInt(pos*1000000);    // Convert ms to ns
    // console.log('pos = ' + pos + ' duty cycle = ' + dutyCycle);
    fs.writeFileSync(PWMPATH+'/duty_cycle', dutyCycle);
}

```

(continues on next page)

(continued from previous page)

```
process.on('SIGINT', function() {
    console.log('Got SIGINT, turning motor off');
    clearInterval(timer);             // Stop the timer
    fs.writeFileSync(PWMPATH+'/enable', '0');
});

// | Pin    | pwm | channel
// | P9_31 | 0    | a
// | P9_29 | 0    | b
// | P9_14 | 1    | a
// | P9_16 | 1    | b
// | P8_19 | 2    | a
// | P8_13 | 2    | b
```

servoMotor.js

Running the code causes the motor to move back and forth, progressing to successive positions between the two extremes. You will need to press ^C (Ctrl-C) to stop the script.

Controlling a Servo with an Rotary Encoder

Problem You have a rotary encoder from *Reading a rotary encoder (rotaryEncoder.js)* that you want to control a servo motor.

Solution Combine the code from *Reading a rotary encoder (rotaryEncoder.js)* and *Controlling a Servo Motor*.

```
bone$ config-pin P9_16 pwm
bone$ config-pin P8_11 eqep
bone$ config-pin P8_12 eqep
bone$ ./servoEncoder.py
```

Listing 15.27: Code for driving a servo motor with a rotary encorder(servoEncoder.py)

```
#!/usr/bin/env python
# ////////////////////////////////////////
# //        servoEncoder.py
# //        Drive a simple servo motor using rotary encoder viq eQEP
# //        Wiring: Servo on P9_16, rotary encoder on P8_11 and P8_12
# //        Setup:  config-pin P9_16 pwm
# //                        config-pin P8_11 eqep
# //                        config-pin P8_12 eqep
# //        See:
# ////////////////////////////////////////
import time
import signal
import sys

# Set up encoder
eQEP = '2'
COUNTERPATH = '/dev/bone/counter/counter'+eQEP+'/count0'
maxCount = '180'

ms = 100         # Time between samples in ms

# Set the eEQP maximum count
fQEP = open(COUNTERPATH+'/ceiling', 'w')
fQEP.write(maxCount)
fQEP.close()
```

(continues on next page)

(continued from previous page)

```
26
27  # Enable
28  fQEP = open(COUNTERPATH+'/enable', 'w')
29  fQEP.write('1')
30  fQEP.close()
31
32  fQEP = open(COUNTERPATH+'/count', 'r')
33
34  # Set up servo
35  pwmPeriod = '20000000'    # Period in ns, (20 ms)
36  pwm     = '1'  # pwm to use
37  channel = 'b'  # channel to use
38  PWMPATH='/dev/bone/pwm/'+pwm+'/'+channel
39  low  = 0.6 # Smallest angle (in ms)
40  hi   = 2.5 # Largest angle (in ms)
41  ms   = 250 # How often to change position, in ms
42  pos  = 1.5 # Current position, about middle ms)
43  step = 0.1 # Step size to next position
44
45  def signal_handler(sig, frame):
46      print('Got SIGINT, turning motor off')
47      f = open(PWMPATH+'/enable', 'w')
48      f.write('0')
49      f.close()
50      sys.exit(0)
51  signal.signal(signal.SIGINT, signal_handler)
52
53  f = open(PWMPATH+'/period', 'w')
54  f.write(pwmPeriod)
55  f.close()
56  f = open(PWMPATH+'/duty_cycle', 'w')
57  f.write(str(round(int(pwmPeriod)/2)))
58  f.close()
59  f = open(PWMPATH+'/enable', 'w')
60  f.write('1')
61  f.close()
62
63  print('Hit ^C to stop')
64
65  olddata = -1
66  while True:
67          fQEP.seek(0)
68          data = fQEP.read()[:-1]
69          # Print only if data changes
70          if data != olddata:
71                  olddata = data
72                  # print(”data = ” + data)
73                  # # map 0-180  to low-hi
74                  duty_cycle = -1*int(data)*(hi-low)/180.0 + hi
75                  duty_cycle = str(int(duty_cycle*1000000))        # Convert␣
    ↪from ms to ns
76                  # print('duty_cycle = ' + duty_cycle)
77                  f = open(PWMPATH+'/duty_cycle', 'w')
78                  f.write(duty_cycle)
79                  f.close()
80          time.sleep(ms/1000)
81
82  # Black OR Pocket
83  # eQEP0:        P9.27 and P9.42 OR P1_33 and P2_34
84  # eQEP1:        P9.33 and P9.35
85  # eQEP2:        P8.11 and P8.12 OR P2_24 and P2_33
```

(continues on next page)

(continued from previous page)

```

# AI
# eQEP1:        P8.33 and P8.35
# eQEP2:        P8.11 and P8.12 or P9.19 and P9.41
# eQEP3:        P8.24 and P8.25 or P9.27 and P9.42

# | Pin   | pwm | channel
# | P9_31 | 0   | a
# | P9_29 | 0   | b
# | P9_14 | 1   | a
# | P9_16 | 1   | b
# | P8_19 | 2   | a
# | P8_13 | 2   | b
```

servoEncoder.py

Controlling the Speed of a DC Motor

Problem You have a DC motor (or a solenoid) and want a simple way to control its speed, but not the direction.

Solution It would be nice if you could just wire the DC motor to BeagleBone Black and have it work, but it won't. Most motors require more current than the GPIO ports on the Bone can supply. Our solution is to use a transistor to control the current to the bone.

Here we configure the encoder to returns value between 0 and 180 inclusive. This value is then mapped to a value between *min* (0.6 ms) and *max* (2.5 ms). This number is converted from milliseconds and nanoseconds (time 1000000) and sent to the servo motor via the pwm.

Here's what you will need:

- 3 V to 5 V DC motor
- Breadboard and jumper wires.
- 1 kΩ resistor.
- Transistor 2N3904.
- Diode 1N4001.
- Power supply for the motor (optional)

If you are using a larger motor (more current), you will need to use a larger transistor.

Wire your breadboard as shown in *Wiring a DC motor to spin one direction*.

Use the code in *Driving a DC motor in one direction (dcMotor.py)* to run the motor.

Python

JavaScript

Listing 15.28: Driving a DC motor in one direction (dcMotor.py)

```
#!/usr/bin/env python
# ////////////////////////////////////
# //        dcMotor.js
# //        This is an example of driving a DC motor
# //        Wiring:
# //        Setup:  config-pin P9_16 pwm
# //        See:
# ////////////////////////////////////
import time
import signal
```

(continues on next page)

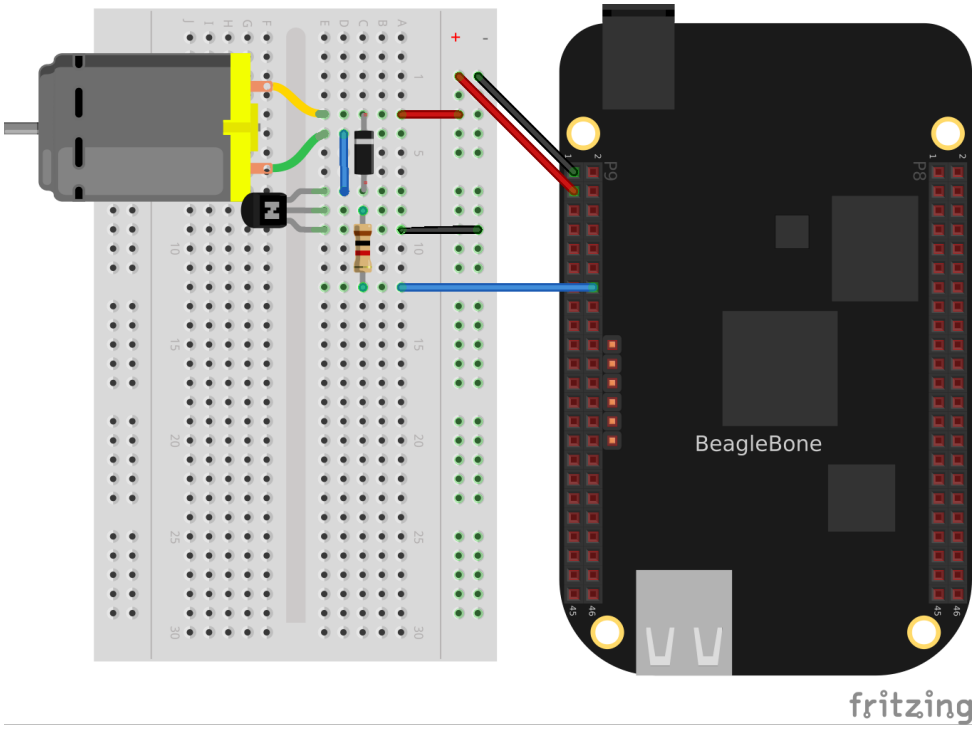


Fig. 15.35: Wiring a DC motor to spin one direction

(continued from previous page)

```
import sys

def signal_handler(sig, frame):
    print('Got SIGINT, turning motor off')
    f = open(PWMPATH+'/enable', 'w')
    f.write('0')
    f.close()
    sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)

pwmPeriod = '1000000'    # Period in ns
pwm     = '1'   # pwm to use
channel = 'b'   # channel to use
PWMPATH='/dev/bone/pwm/'+pwm+'/'+channel

low = 0.05      # Slowest speed (duty cycle)
hi  = 1         # Fastest (always on)
ms = 100        # How often to change speed, in ms
speed = 0.5     # Current speed
step = 0.05     # Change in speed

f = open(PWMPATH+'/duty_cycle', 'w')
f.write('0')
f.close()
f = open(PWMPATH+'/period', 'w')
f.write(pwmPeriod)
f.close()
f = open(PWMPATH+'/enable', 'w')
f.write('1')
f.close()

f = open(PWMPATH+'/duty_cycle', 'w')
while True:
```

(continues on next page)

(continued from previous page)

```
    speed += step
    if(speed > hi or speed < low):
        step *= -1
    duty_cycle = str(round(speed*1000000))     # Convert ms to ns
    f.seek(0)
    f.write(duty_cycle)
    time.sleep(ms/1000)
```

dcMotor.py

Listing 15.29: Driving a DC motor in one direction (dcMotor.js)

```
#!/usr/bin/env node
////////////////////////////////////////
//        dcMotor.js
//        This is an example of driving a DC motor
//        Wiring:
//        Setup:  config-pin P9_16 pwm
//        See:
////////////////////////////////////////
const fs = require("fs");

const pwmPeriod = '1000000';    // Period in ns
const pwm     = '1';  // pwm to use
const channel = 'b';  // channel to use
const PWMPATH='/dev/bone/pwm/'+pwm+'/'+channel;

const low = 0.05,     // Slowest speed (duty cycle)
      hi  = 1,        // Fastest (always on)
      ms = 100;       // How often to change speed, in ms
var   speed = 0.5,    // Current speed;
      step = 0.05;    // Change in speed

// fs.writeFileSync(PWMPATH+'/export', pwm);   // Export the pwm channel
// Set the period in ns, first 0 duty_cycle,
fs.writeFileSync(PWMPATH+'/duty_cycle', '0');
fs.writeFileSync(PWMPATH+'/period', pwmPeriod);
fs.writeFileSync(PWMPATH+'/duty_cycle', pwmPeriod/2);
fs.writeFileSync(PWMPATH+'/enable', '1');

timer = setInterval(sweep, ms);

function sweep() {
    speed += step;
    if(speed > hi || speed < low) {
        step *= -1;
    }
    fs.writeFileSync(PWMPATH+'/duty_cycle', parseInt(pwmPeriod*speed));
    // console.log('speed = ' + speed);
}

process.on('SIGINT', function() {
    console.log('Got SIGINT, turning motor off');
    clearInterval(timer);       // Stop the timer
    fs.writeFileSync(PWMPATH+'/enable', '0');
});
```

dcMotor.js

See Also

How do you change the direction of the motor? See *Controlling the Speed and Direction of a DC Motor*.

Controlling the Speed and Direction of a DC Motor

Problem You would like your DC motor to go forward and backward.

Solution Use an H-bridge to switch the terminals on the motor so that it will run both backward and forward. We'll use the `L293D` a common, single-chip H-bridge.

Here's what you will need:

- 3 V to 5 V motor.
- Breadboard and jumper wires.
- L293D H-Bridge IC.
- Power supply for the motor (optional)

Lay out your breadboard as shown in *Driving a DC motor with an H-bridge*. Ensure that the L293D is positioned correctly. There is a notch on one end that should be pointed up.

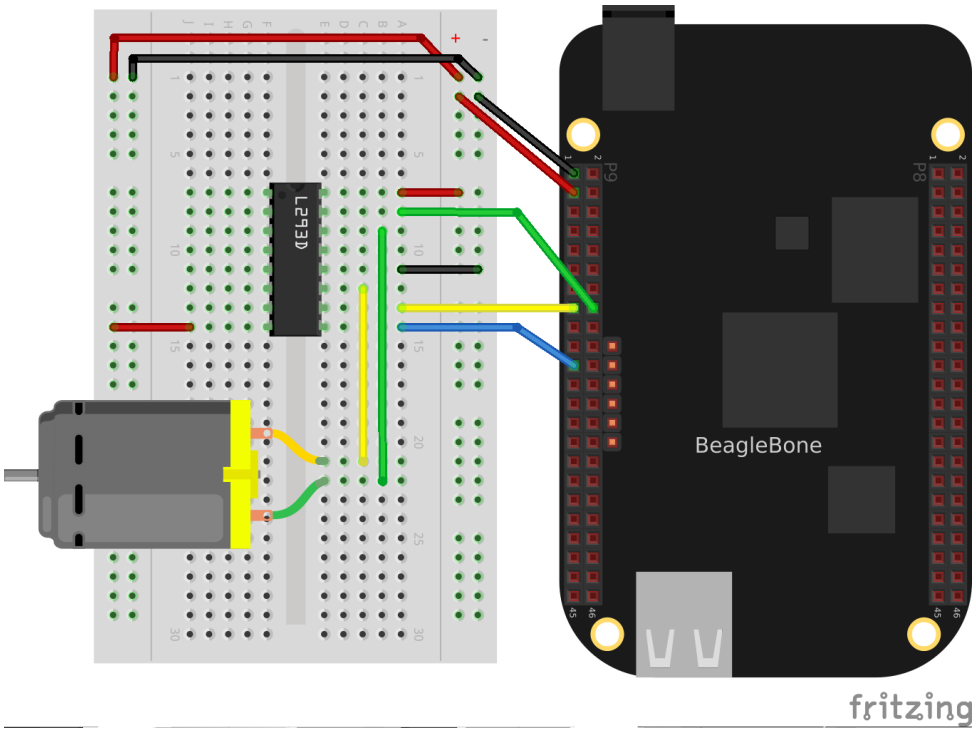


Fig. 15.36: Driving a DC motor with an H-bridge

The code in *Code for driving a DC motor with an H-bridge (h-bridgeMotor.js)* (`h-bridgeMotor.js`) looks much like the code for driving the DC motor with a transistor (*Driving a DC motor in one direction (dcMotor.js)*). The additional code specifies which direction to spin the motor.

Listing 15.30: Code for driving a DC motor with an H-bridge (h-bridgeMotor.js)

```
1 #!/usr/bin/env node
2
3 // This example uses an H-bridge to drive a DC motor in two directions
```

(continues on next page)

(continued from previous page)

```
4
5  var b = require('bonescript');
6
7  var enable = 'P9_21';    // Pin to use for PWM speed control
8      in1    = 'P9_15',
9      in2    = 'P9_16',
10     step = 0.05,    // Change in speed
11     min  = 0.05,    // Min duty cycle
12     max  = 1.0,     // Max duty cycle
13     ms   = 100,     // Update time, in ms
14     speed = min;    // Current speed;
15
16 b.pinMode(enable, b.ANALOG_OUTPUT, 6, 0, 0, doInterval);
17 b.pinMode(in1, b.OUTPUT);
18 b.pinMode(in2, b.OUTPUT);
19
20 function doInterval(x) {
21     if(x.err) {
22         console.log('x.err = ' + x.err);
23         return;
24     }
25     timer = setInterval(sweep, ms);
26 }
27
28 clockwise();        // Start by going clockwise
29
30 function sweep() {
31     speed += step;
32     if(speed > max || speed < min) {
33         step *= -1;
34         step>0 ? clockwise() : counterClockwise();
35     }
36     b.analogWrite(enable, speed);
37     console.log('speed = ' + speed);
38 }
39
40 function clockwise() {
41     b.digitalWrite(in1, b.HIGH);
42     b.digitalWrite(in2, b.LOW);
43 }
44
45 function counterClockwise() {
46     b.digitalWrite(in1, b.LOW);
47     b.digitalWrite(in2, b.HIGH);
48 }
49
50 process.on('SIGINT', function() {
51     console.log('Got SIGINT, turning motor off');
52     clearInterval(timer);         // Stop the timer
53     b.analogWrite(enable, 0);     // Turn motor off
54 });
```

h-bridgeMotor.js

Driving a Bipolar Stepper Motor

Problem You want to drive a stepper motor that has four wires.

Solution Use an L293D H-bridge. The bipolar stepper motor requires us to reverse the coils, so we need to use an H-bridge.

Here's what you will need:

- Breadboard and jumper wires.
- 3 V to 5 V bipolar stepper motor.
- L293D H-Bridge IC.

Wire as shown in *Bipolar stepper motor wiring*.

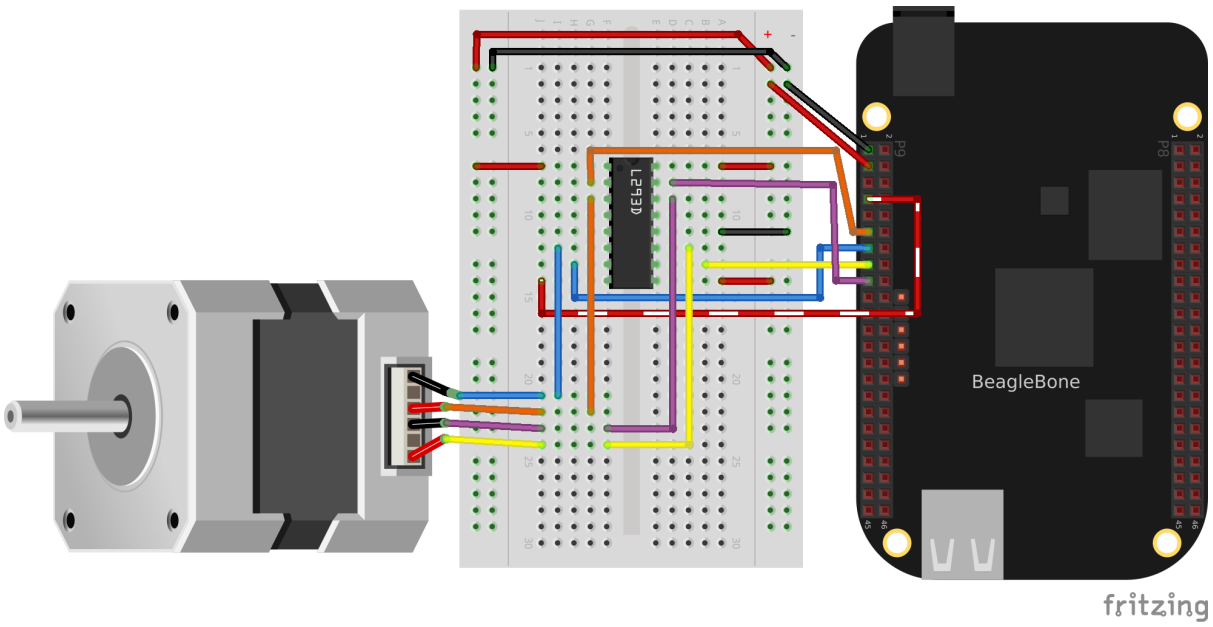


Fig. 15.37: Bipolar stepper motor wiring

Use the code in *Driving a bipolar stepper motor (bipolarStepperMotor.py)* to drive the motor.

Listing 15.31: Driving a bipolar stepper motor (bipolarStepperMotor.py)

```
#!/usr/bin/env python
import time
import os
import signal
import sys

# Motor is attached here
# controller = ["P9_11", "P9_13", "P9_15", "P9_17"];
# controller = ["30", "31", "48", "5"]
# controller = ["P9_14", "P9_16", "P9_18", "P9_22"];
controller = ["50", "51", "4", "2"]
states = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]
statesHiTorque = [[1,1,0,0], [0,1,1,0], [0,0,1,1], [1,0,0,1]]
statesHalfStep = [[1,0,0,0], [1,1,0,0], [0,1,0,0], [0,1,1,0],
                  [0,0,1,0], [0,0,1,1], [0,0,0,1], [1,0,0,1]]

curState = 0    # Current state
ms = 100        # Time between steps, in ms
maxStep = 22    # Number of steps to turn before turning around
minStep = 0     # minimum step to turn back around on

CW  =  1       # Clockwise
CCW = -1
pos =  0       # current position and direction
direction = CW
```

(continues on next page)

(continued from previous page)

```
GPIOPATH="/sys/class/gpio"

def signal_handler(sig, frame):
    print('Got SIGINT, turning motor off')
    for i in range(len(controller)) :
        f = open(GPIOPATH+"/gpio"+controller[i]+"/value", "w")
        f.write('0')
        f.close()
    sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)
print('Hit ^C to stop')

def move():
    global pos
    global direction
    global minStep
    global maxStep
    pos += direction
    print("pos: " + str(pos))
    # Switch directions if at end.
    if (pos >= maxStep or pos <= minStep) :
        direction *= -1
    rotate(direction)

# This is the general rotate
def rotate(direction) :
    global curState
    global states
        # print("rotate(%d)", direction);
    # Rotate the state according to the direction of rotation
    curState +=  direction
    if(curState >= len(states)) :
        curState = 0;
    elif(curState<0) :
        curState = len(states)-1
    updateState(states[curState])

# Write the current input state to the controller
def updateState(state) :
    global controller
    print(state)
    for i in range(len(controller)) :
        f = open(GPIOPATH+"/gpio"+controller[i]+"/value", "w")
        f.write(str(state[i]))
        f.close()

# Initialize motor control pins to be OUTPUTs
for i in range(len(controller)) :
    # Make sure pin is exported
    if (not os.path.exists(GPIOPATH+"/gpio"+controller[i])):
        f = open(GPIOPATH+"/export", "w")
        f.write(pin)
        f.close()
    # Make it an output pin
    f = open(GPIOPATH+"/gpio"+controller[i]+"/direction", "w")
    f.write("out")
    f.close()

# Put the motor into a known state
updateState(states[0])
rotate(direction)
```

(continues on next page)

(continued from previous page)

```
87
88  # Rotate
89  while True:
90      move()
91      time.sleep(ms/1000)
```

`bipolarStepperMotor.py`

When you run the code, the stepper motor will rotate back and forth.

Driving a Unipolar Stepper Motor

Problem You want to drive a stepper motor that has five or six wires.

Solution If your stepper motor has five or six wires, it's a `unipolar` stepper and is wired differently than the bipolar. Here, we'll use a `ULN2003 Darlington Transistor Array IC` to drive the motor.

Here's what you will need:

- Breadboard and jumper wires.
- 3 V to 5 V unipolar stepper motor.
- ULN2003 Darlington Transistor Array IC.

Wire, as shown in *Unipolar stepper motor wiring*.

Note: The IC in *Unipolar stepper motor wiring* is illustrated upside down from the way it is usually displayed.

That is, the notch for pin 1 is on the bottom. This made drawing the diagram much cleaner.

Also, notice the `banded` wire running the *P9_7* (5 V) to the UL2003A. The stepper motor I'm using runs better at 5 V, so I'm using the Bone's 5 V power supply. The signal coming from the GPIO pins is 3.3 V, but the U2003A will step them up to 5 V to drive the motor.

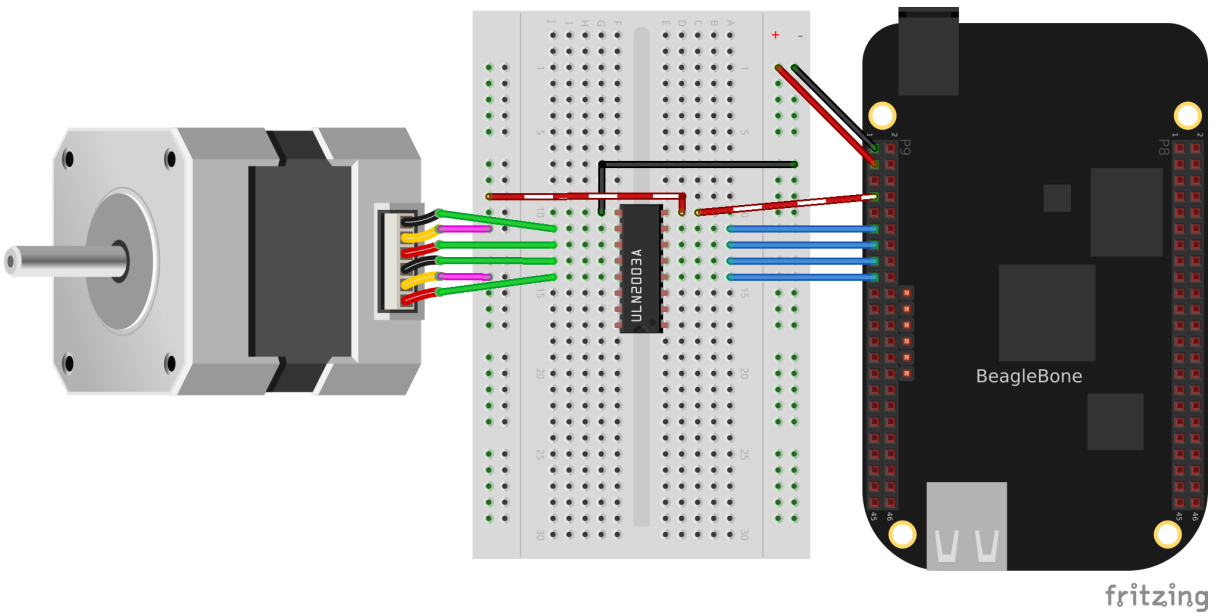


Fig. 15.38: Unipolar stepper motor wiring

The code for driving the motor is in `unipolarStepperMotor.js` however, it is almost identical to the bipolar stepper code (*Driving a bipolar stepper motor (bipolarStepperMotor.py)*), so *Changes to bipolar code to drive a unipolar stepper motor (unipolarStepperMotor.js.diff)* shows only the lines that you need to change.

Listing 15.32: Changes to bipolar code to drive a unipolar stepper motor (unipolarStepperMotor.py.diff)

```
# controller = ["P9_11", "P9_13", "P9_15", "P9_17"]
controller = ["30", "31", "48", "5"]
states = [[1,1,0,0], [0,1,1,0], [0,0,1,1], [1,0,0,1]]
curState = 0    // Current state
ms = 100        // Time between steps, in ms
max = 200       // Number of steps to turn before turning around
```

`unipolarStepperMotor.py.diff`

Listing 15.33: Changes to bipolar code to drive a unipolar stepper motor (unipolarStepperMotor.js.diff)

```
# var controller = ["P9_11", "P9_13", "P9_15", "P9_17"];
controller = ["30", "31", "48", "5"]
var states = [[1,1,0,0], [0,1,1,0], [0,0,1,1], [1,0,0,1]];
var curState = 0;    // Current state
var ms = 100,        // Time between steps, in ms
    max = 200,       // Number of steps to turn before turning around
```

`unipolarStepperMotor.js.diff`

The code in this example makes the following changes:

- The *states* are different. Here, we have two pins high at a time.
- The time between steps (*ms*) is shorter, and the number of steps per direction (*max*) is bigger. The unipolar stepper I'm using has many more steps per rotation, so I need more steps to make it go around.

15.1.5 Beyond the Basics

In *Basics*, you learned how to set up BeagleBone Black, and *Sensors*, *Displays and Other Outputs*, and *Motors* showed how to interface to the physical world. The remainder of the book moves into some more exciting advanced topics, and this chapter gets you ready for them.

The recipes in this chapter assume that you are running Linux on your host computer (*Selecting an OS for Your Development Host Computer*) and are comfortable with using Linux. We continue to assume that you are logged in as *debian* on your Bone.

Running Your Bone Standalone

Problem You want to use BeagleBone Black as a desktop computer with keyboard, mouse, and an HDMI display.

Solution The Bone comes with USB and a microHDMI output. All you need to do is connect your keyboard, mouse, and HDMI display to it.

To make this recipe, you will need:

- Standard HDMI cable and female HDMI-to-male microHDMI adapter, or
- MicroHDMI-to-HDMI adapter cable
- HDMI monitor
- USB keyboard and mouse

- Powered USB hub

Note: The microHDMI adapter is nice because it allows you to use a regular HDMI cable with the Bone. However, it will block other ports and can damage the Bone if you aren't careful. The microHDMI-to-HDMI cable won't have these problems.

Tip: You can also use an HDMI-to-DVI cable and use your Bone with a DVI-D display.

The adapter looks something like *Female HDMI-to-male microHDMI adapter*.



Fig. 15.39: Female HDMI-to-male microHDMI adapter

Plug the small end into the microHDMI input on the Bone and plug your HDMI cable into the other end of the adapter and your monitor. If nothing displays on your Bone, reboot.

If nothing appears after the reboot, edit the `/boot/uEnv.txt` file. Search for the line containing `disable_uboot_overlay_video=1` and make sure it's commented out:

```
###Disable auto loading of virtual capes (emmc/video/wireless/adc)
#disable_uboot_overlay_emmc=1
#disable_uboot_overlay_video=1
```

Then reboot.

The `/boot/uEnv.txt` file contains a number of configuration commands that are executed at boot time. The # character is used to add comments; that is, everything to the right of a +# is ignored by the Bone and is assumed to be for humans to read. In the previous example, *###Disable auto loading* is a comment that informs us the next line(s) are for disabling things. Two *disable_uboot_overlay* commands follow. Both should be commented-out and won't be executed by the Bone.

Why not just remove the line? Later, you might decide you need more general-purpose input/output (GPIO) pins and don't need the HDMI display. If so, just remove the # from the `disable_uboot_overlay_video=1` command. If you had completely removed the line earlier, you would have to look up the details somewhere to re-create it.

When in doubt, comment-out don't delete.

Note: If you want to re-enable the HDMI audio, just comment-out the line you added.

The Bone has only one USB port, so you will need to get either a keyboard with a USB hub or a USB hub. Plug the USB hub into the Bone and then plug your keyboard and mouse in to the hub. You now have a Beagle workstation no host computer is needed.

Tip: A powered hub is recommended because USB can supply only 500 mA, and you'll want to plug many things into the Bone.

This recipe disables the HDMI audio, which allows the Bone to try other resolutions. If this fails, see BeagleBoneBlack HDMI for how to force the Bone's resolution to match your monitor.

Selecting an OS for Your Development Host Computer

Problem Your project needs a host computer, and you need to select an operating system (OS) for it.

Solution For projects that require a host computer, we assume that you are running Linux Ubuntu 22.04 LTS. You can be running either a native installation, through Windows Subsystem for Linux, via a virtual machine such as VirtualBox, or in the cloud (Microsoft Azure or Amazon Elastic Compute Cloud, EC2, for example).

Recently I've been preferring Windows Subsystem for Linux.

Getting to the Command Shell via SSH

Problem You want to connect to the command shell of a remote Bone from your host computer.

Solution *Running Python and JavaScript Applications from Visual Studio Code* shows how to run shell commands in the Visual Studio Code *bash* tab. However, the Bone has Secure Shell (SSH) enabled right out of the box, so you can easily connect by using the following command to log in as user *debian*, (note the *$* at the end of the prompt):

```
host$ ssh debian@192.168.7.2
Warning: Permanently added '192.168.7.2' (ED25519) to the list of known␣
↪hosts.
Debian GNU/Linux 11

BeagleBoard.org Debian Bullseye IoT Image 2023-06-03
Support: https://bbb.io/debian
default username:password is [debian:temppwd]


The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jun  8 14:02:40 2023 from 192.168.7.1
bone$
```

Default password *debian* has the default password *temppwd*. It's best to change the password:

```
bone$ password
Changing password for debian.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
password: password updated successfully
```

Removing the *Message of the Day*

Problem Every time you login a long message is displayed that you don't need to see.

Solution The contents of the files */etc/motd*, */etc/issue* and */etc/issue.net* are displayed everytime you long it. You can prevent them from being displayed by moving them elsewhere.

```
bone$ sudo mv /etc/motd /etc/motd.orig
bone$ sudo mv /etc/issue /etc/issue.orig
bone$ sudo mv /etc/issue.net /etc/issue.net.orig
```

Now, the next time you *ssh* in they won't be displayed.

Getting to the Command Shell via the Virtual Serial Port

Problem You want to connect to the command shell of a remote Bone from your host computer without using SSH.

Solution Sometimes, you can't connect to the Bone via SSH, but you have a network working over USB to the Bone. There is a way to access the command line to fix things without requiring extra hardware. (*Viewing and Debugging the Kernel and u-boot Messages at Boot Time* shows a way that works even if you don't have a network working over USB, but it requires a special serial-to-USB cable.)

Note: This method doesn't work with WSL.

First, check to ensure that the serial port is there. On the host computer, run the following command:

```
host$ ls -ls /dev/ttyACM0
0 crw-rw---- 1 root dialout 166, 0 Jun 19 11:47 /dev/ttyACM0
```

/dev/ttyACM0 is a serial port on your host computer that the Bone creates when it boots up. The letters *crw-rw—-* show that you can't access it as a normal user. However, you can access it if you are part of *dialout* group. See if you are in the *dialout* group:

```
host$ groups
yoder adm tty uucp dialout cdrom sudo dip plugdev lpadmin sambashare
```

Looks like I'm already in the group, but if you aren't, just add yourself to the group:

```
host$ sudo adduser $USER dialout
```

You have to run *adduser* only once. Your host computer will remember the next time you boot up. Now, install and run the *screen* command:

```
host$ sudo apt install screen
host$ screen /dev/ttyACM0 115200
Debian GNU/Linux 7 beaglebone ttyGS0

default username:password is [debian:temppwd]

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

The IP Address for usb0 is: 192.168.7.2
beaglebone login:
```

The `/dev/ttyACM0` parameter specifies which serial port to connect to, and *115200* tells the speed of the connection. In this case, it's 115,200 bits per second.

Viewing and Debugging the Kernel and u-boot Messages at Boot Time

Problem You want to see the messages that are logged by BeagleBone Black as it comes to life.

Solution There is no network in place when the Bone first boots up, so *Getting to the Command Shell via SSH* and *Getting to the Command Shell via the Virtual Serial Port* won't work. This recipe uses some extra hardware (FTDI cable) to attach to the Bone's console serial port.

To make this recipe, you will need:

- 3.3 V FTDI cable

Warning: Be sure to get a 3.3 V FTDI cable (shown in *FTDI cable*), because the 5 V cables won't work.

Tip: The Bone's Serial Debug J1 connector has Pin 1 connected to ground, Pin 4 to receive, and Pin 5 to transmit. The other pins are not attached.

Look for a small triangle at the end of the FTDI cable (*FTDI connector*). It's often connected to the black wire.

Next, look for the FTDI pins of the Bone (labeled *J1* on the Bone), shown in *FTDI pins for the FTDI connector*. They are next to the P9 header and begin near pin 20. There is a white dot near P9_20.

Plug the FTDI connector into the FTDI pins, being sure to connect the `triangle` pin on the connector to the `white dot` pin of the *FTDI* connector.

Now, run the following commands on your host computer:

```
host$ ls -ls /dev/ttyUSB0
0 crw-rw---- 1 root dialout 188, 0 Jun 19 12:43 /dev/ttyUSB0
host$ sudo adduser $USER dialout
host$ screen /dev/ttyUSB0 115200
Debian GNU/Linux 7 beaglebone ttyO0

default username:password is [debian:temppwd]

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

The IP Address for usb0 is: 192.168.7.2
beaglebone login:
```

Note: Your screen might initially be blank. Press Enter a couple times to see the login prompt.



Fig. 15.40: FTDI cable

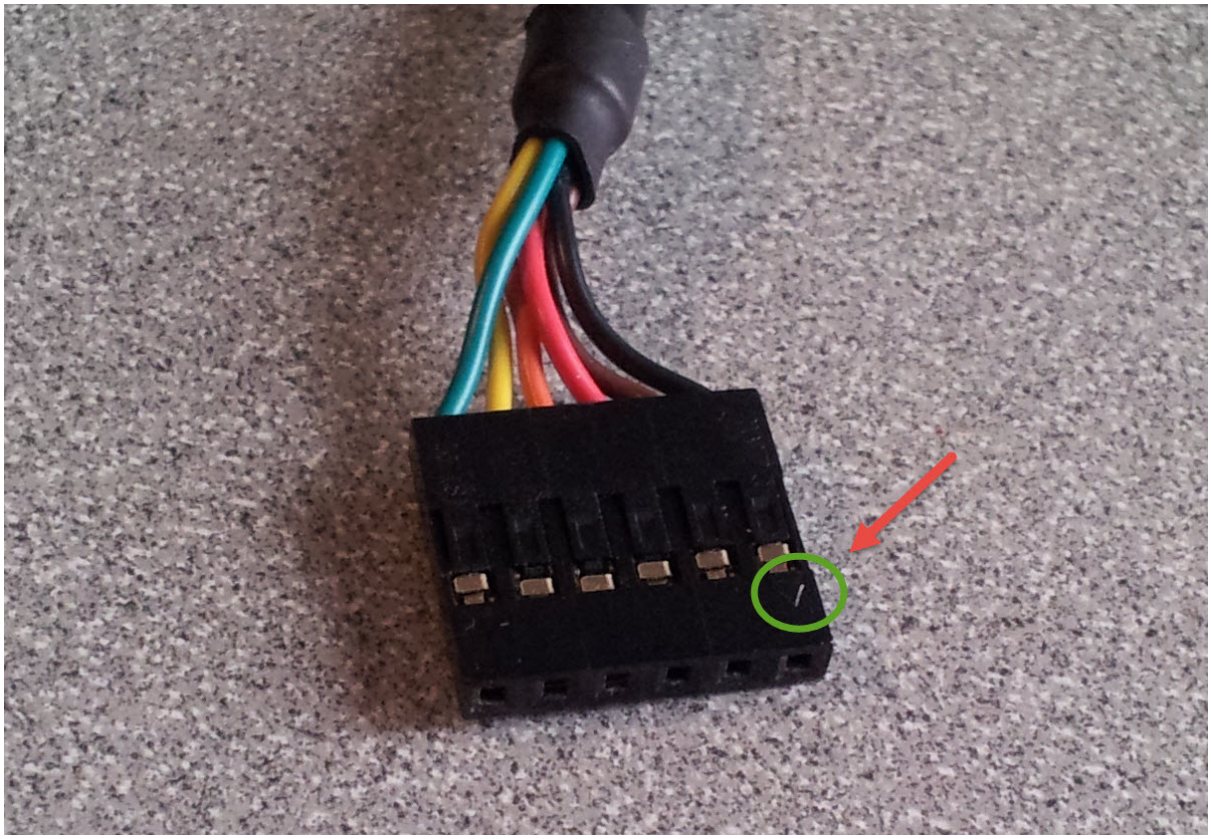


Fig. 15.41: FTDI connector

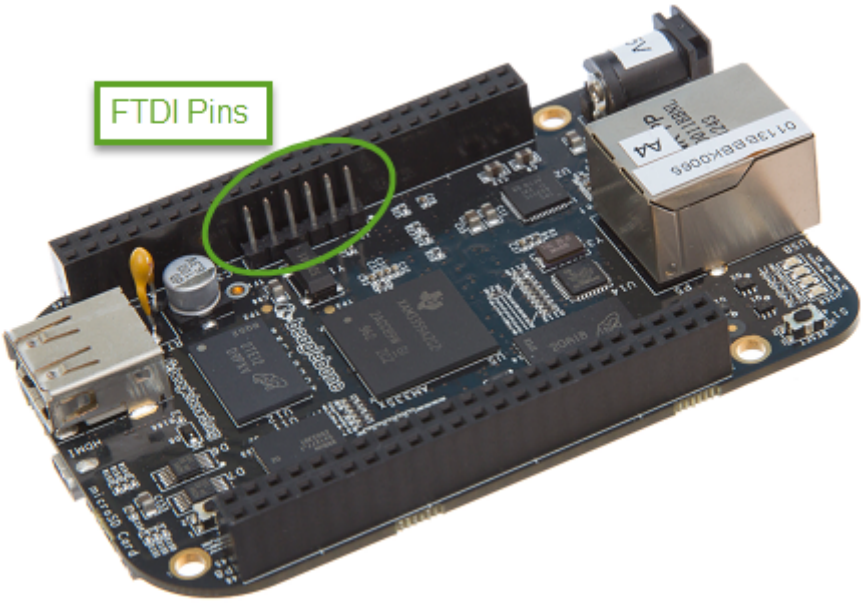


Fig. 15.42: FTDI pins for the FTDI connector

Verifying You Have the Latest Version of the OS on Your Bone from the Shell

Problem You are logged in to your Bone with a command prompt and want to know what version of the OS you are running.

Solution Log in to your Bone and enter the following command:

```
bone$ cat /etc/dogtag
BeagleBoard.org Debian Bullseye IoT Image 2023-06-03
```

Verifying You Have the Latest Version of the OS on Your Bone shows how to open the `/etc/dogtag` file to see the OS version. See *Running the Latest Version of the OS on Your Bone* if you need to update your OS.

Controlling the Bone Remotely with a VNC

Problem You want to access the BeagleBone's graphical desktop from your host computer.

Solution Install and run a Virtual Network Computing (VNC) server:

```
bone$ sudo apt update
bone$ sudo apt install tightvncserver
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
...
update-alternatives: using /usr/bin/Xtightvnc to provide /usr/bin/Xvnc␣
↪(Xvnc) in auto mode
update-alternatives: using /usr/bin/tightvncpasswd to provide /usr/bin/
↪vncpasswd (vncpasswd) in auto mode
Processing triggers for libc-bin (2.31-13+deb11u6) ...

bone$ tightvncserver

You will require a password to access your desktops.

Password:
Verify:
Would you like to enter a view-only password (y/n)? n
xauth: (argv):1:  bad display name "beaglebone:1" in "add" command

New 'X' desktop is beaglebone:1

Creating default startup script /home/debian/.vnc/xstartup
Starting applications specified in /home/debian/.vnc/xstartup
Log file is /home/debian/.vnc/beagleboard:1.log
```

To connect to the Bone, you will need to run a VNC client. There are many to choose from. Remmina Remote Desktop Client is already installed on Ubuntu. Start and select the new remote desktop file button (*Creating a new remote desktop file in Remmina Remote Desktop Client*).

Give your connection a name, being sure to select "Remmina VNC Plugin" Also, be sure to add *:1* after the server address, as shown in *Configuring the Remmina Remote Desktop Client*. This should match the *:1* that was displayed when you started *vncserver*.

Click Connect to start graphical access to your Bone, as shown in *The Remmina Remote Desktop Client showing the BeagleBone desktop*.

Tip: You might need to resize the VNC screen on your host to see the bottom menu bar on your Bone.

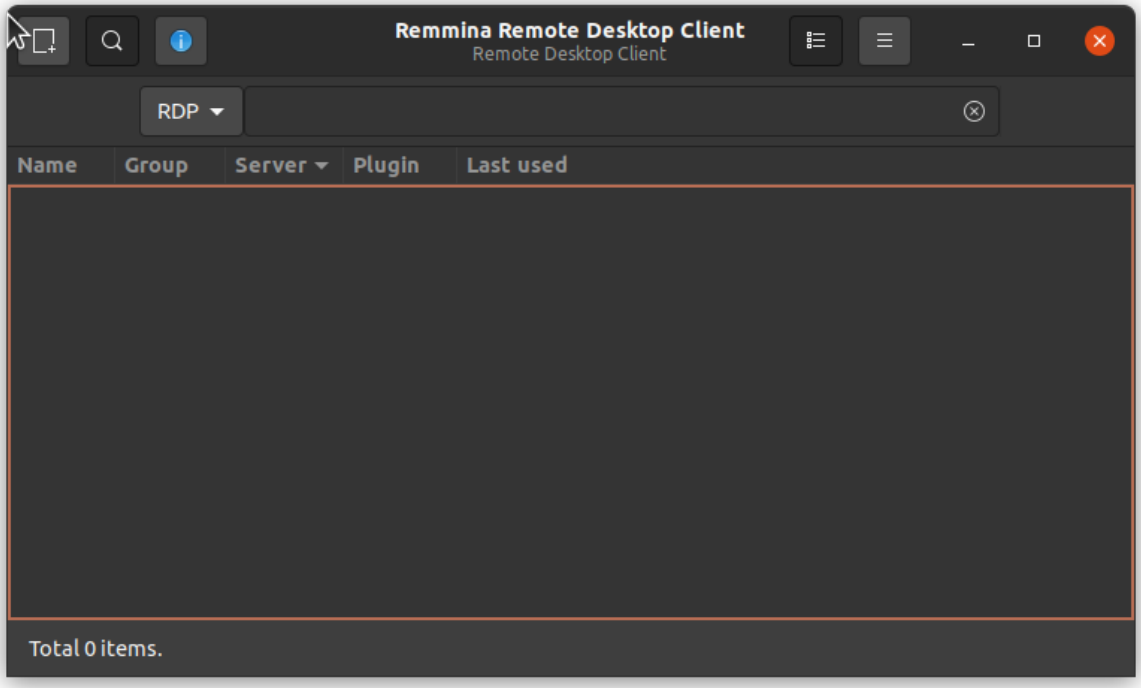


Fig. 15.43: Creating a new remote desktop file in Remmina Remote Desktop Client

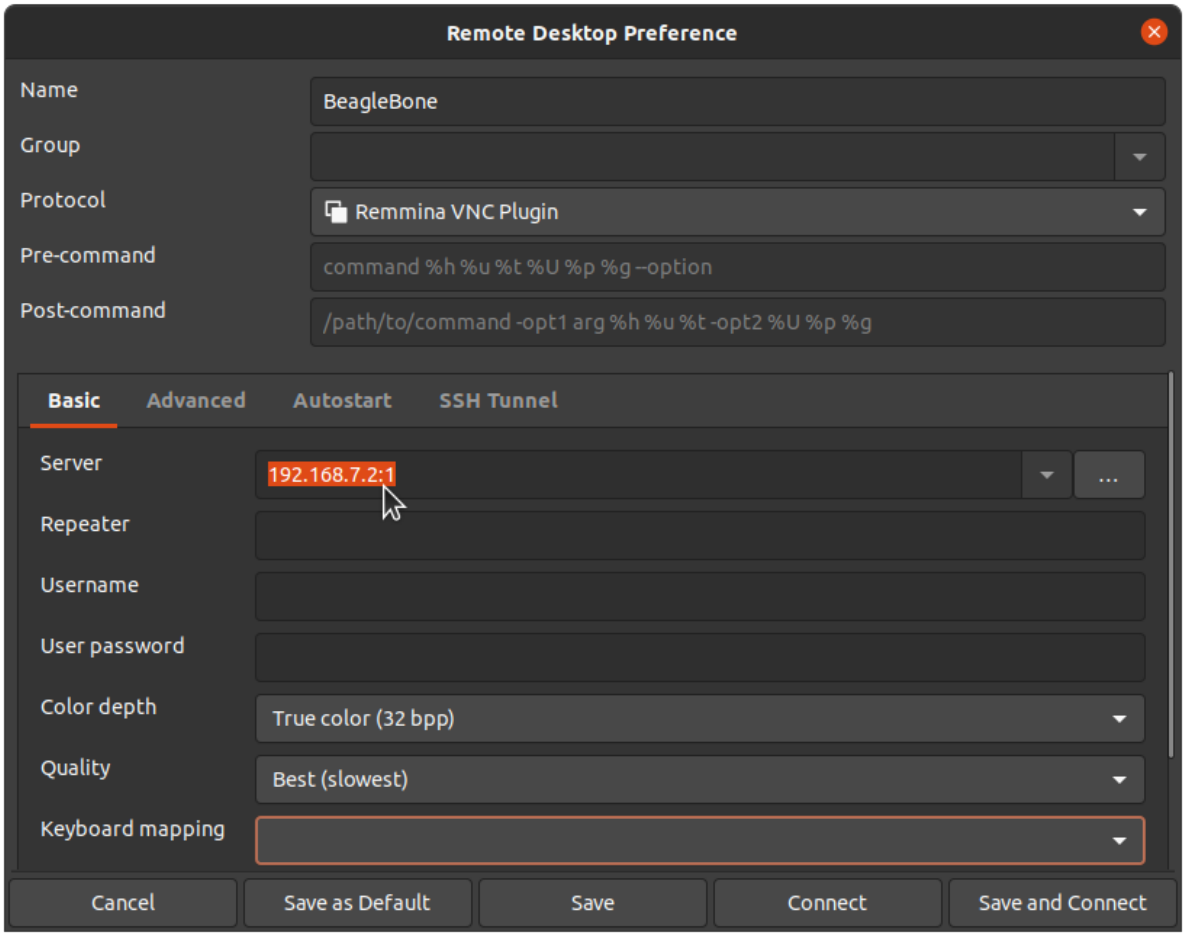


Fig. 15.44: Configuring the Remmina Remote Desktop Client



Fig. 15.45: The Remmina Remote Desktop Client showing the BeagleBone desktop

Note: You need to have X Windows installed and running for the VNC to work. Here's how to install it. This needs some 250M of disk space and 19 minutes to install.

```
bone$ bone$ sudo apt install bbb.io-xfce4-desktop
bone$ sdo cp /etc/bbb.io/templates/fbdev.xorg.conf /etc/X11/xorg.conf
bone$ startxfce4
/usr/bin/startxfce4: Starting X server
/usr/bin/startxfce4: 122: exec: xinit: not found
```

Learning Typical GNU/Linux Commands

Problem There are many powerful commands to use in Linux. How do you learn about them?

Solution *Common Linux commands* lists many common Linux commands.

Table 15.4: Common Linux commands

Command	Action
pwd	show current directory
cd	change current directory
ls	list directory contents
chmod	change file permissions
chown	change file ownership
cp	copy files
mv	move files
rm	remove files
mkdir	make directory
rmdir	remove directory
cat	dump file contents
less	progressively dump file
vi	edit file (complex)
nano	edit file (simple)
head	trim dump to top
tail	trim dump to bottom
echo	print/dump value
env	dump environment variables
export	set environment variable
history	dump command history
grep	search dump for strings
man	get help on command
apropos	show list of man pages
find	search for files
tar	create/extract file archives
gzip	compress a file
gunzip	decompress a file
du	show disk usage
df	show disk free space
mount	mount disks
tee	write dump to file in parallel
hexdump	readable binary dumps
whereis	locates binary and source files

Editing a Text File from the GNU/Linux Command Shell

Problem You want to run an editor to change a file.

Solution The Bone comes with a number of editors. The simplest to learn is *nano*. Just enter the following command:

```
bone$ nano file
```

You are now in nano (*Editing a file with nano*). You can't move around the screen using the mouse, so use the arrow keys. The bottom two lines of the screen list some useful commands. Pressing ^G (Ctrl-G) will display more useful commands. ^X (Ctrl-X) exits nano and gives you the option of saving the file.

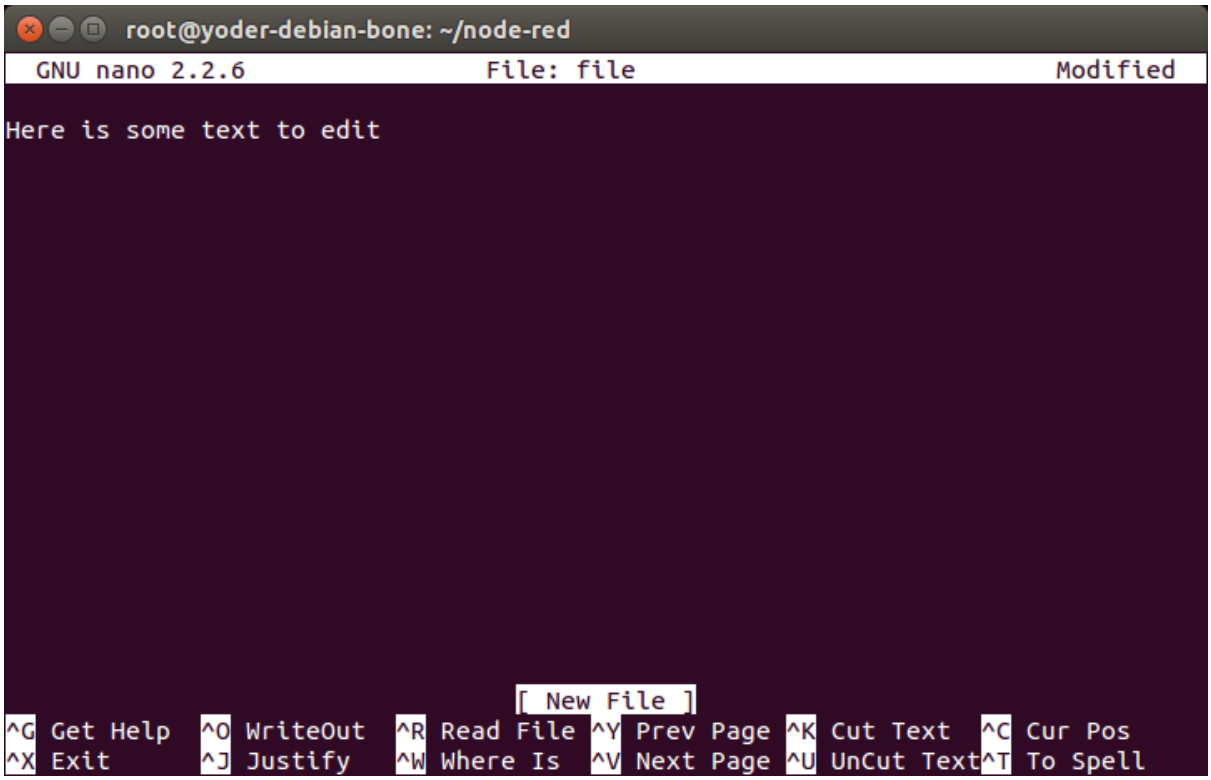


Fig. 15.46: Editing a file with nano

Tip: By default, the file you create will be saved in the directory from which you opened *nano*.

Many other text editors will run on the Bone. *vi*, *vim*, *emacs*, and even *eclipse* are all supported. See *Installing Additional Packages from the Debian Package Feed* to learn if your favorite is one of them.

Establishing an Ethernet-Based Internet Connection

Problem You want to connect your Bone to the Internet using the wired network connection.

Solution Plug one end of an Ethernet patch cable into the RJ45 connector on the Bone (see *The RJ45 port on the Bone*) and the other end into your home hub/router. The yellow and green link lights on both ends should begin to flash.

If your router is already configured to run DHCP (Dynamical Host Configuration Protocol), it will automatically assign an IP address to the Bone.

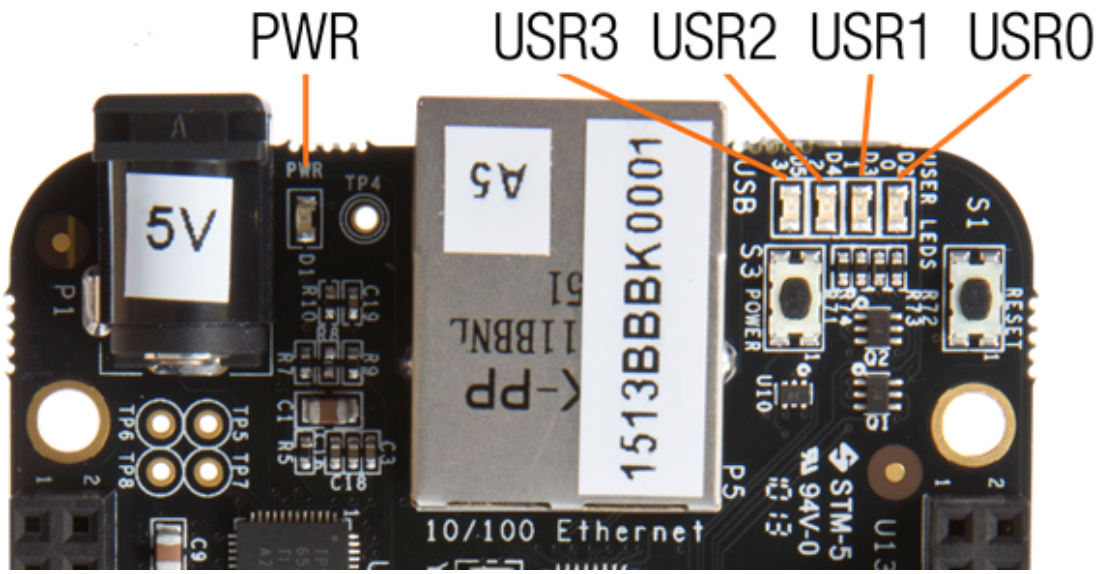


Fig. 15.47: The RJ45 port on the Bone

Warning: It might take a minute or two for your router to detect the Bone and assign the IP address.

To find the IP address, open a terminal window and run the *ip* command:

```
bone$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group␣
↪default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group␣
↪default qlen 1000
   link/ether c8:a0:30:a6:26:e8 brd ff:ff:ff:ff:ff:ff
   inet 10.0.5.144/24 brd 10.0.5.255 scope global dynamic eth0
      valid_lft 80818sec preferred_lft 80818sec
   inet6 fe80::caa0:30ff:fea6:26e8/64 scope link
      valid_lft forever preferred_lft forever
3: usb0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state␣
↪UP group default qlen 1000
   link/ether c2:3f:44:bb:41:0f brd ff:ff:ff:ff:ff:ff
   inet 192.168.7.2/24 brd 192.168.7.255 scope global usb0
      valid_lft forever preferred_lft forever
   inet6 fe80::c03f:44ff:febb:410f/64 scope link
      valid_lft forever preferred_lft forever
4: usb1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state␣
↪UP group default qlen 1000
   link/ether 76:7e:49:46:1b:78 brd ff:ff:ff:ff:ff:ff
   inet 192.168.6.2/24 brd 192.168.6.255 scope global usb1
      valid_lft forever preferred_lft forever
   inet6 fe80::747e:49ff:fe46:1b78/64 scope link
      valid_lft forever preferred_lft forever
5: can0: <NOARP,ECHO> mtu 16 qdisc no-op state DOWN group default qlen 10
   link/can
6: can1: <NOARP,ECHO> mtu 16 qdisc no-op state DOWN group default qlen 10
   link/can
```

My Bone is connected to the Internet in two ways: via the RJ45 connection (*eth0*) and via the USB cable (*usb0*). The *inet* field shows that my Internet address is *10.0.5.144* for the RJ45 connector.

On my university campus, you must register your MAC address before any device will work on the network. The *HWaddr* field gives the MAC address. For *eth0*, it's *c8:a0:30:a6:26:e8*.

The IP address of your Bone can change. If it's been assigned by DHCP, it can change at any time. The MAC address, however, never changes; it is assigned to your ethernet device when it's manufactured.

Warning: When a Bone is connected to some networks it becomes visible to the `world`. If you don't secure your Bone, the world will soon find it. See *Default password* and *Setting Up a Firewall*

On many home networks, you will be behind a firewall and won't be as visible.

Establishing a WiFi-Based Internet Connection

Problem You want BeagleBone Black to talk to the Internet using a USB wireless adapter.

Solution

Tip: For the correct instructions for the image you are using, go to latest-images and click on the image you are using.

I'm running Debian 11.x (Bullseye), the top one, on the BeagleBone Black.

Scroll to the top of the page and you'll see instructions on setting up Wifi. The instructions here are based on using **networkctl**.

Several WiFi adapters work with the Bone. Check WiFi Adapters for the latest list.

To make this recipe, you will need:

- USB Wifi adapter
- 5 V external power supply

Warning: Most adapters need at least 1 A of current to run, and USB supplies only 0.5 A, so be sure to use an external power supply. Otherwise, you will experience erratic behavior and random crashes.

First, plug in the WiFi adapter and the 5 V external power supply and reboot.

Then run *lsusb* to ensure that your Bone found the adapter:

```
bone$ lsusb
Bus 001 Device 002: ID 0bda:8176 Realtek Semiconductor Corp. RTL8188CUS 802.
↪11n
WLAN Adapter
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Note: There is a well-known bug in the Bone's 3.8 kernel series that prevents USB devices from being discovered when hot-plugged, which is why you should reboot. Newer kernels should address this issue.

Next, run *networkctl* to find your adapter's name. Mine is called *wlan0*, but you might see other names, such as *ra0*.

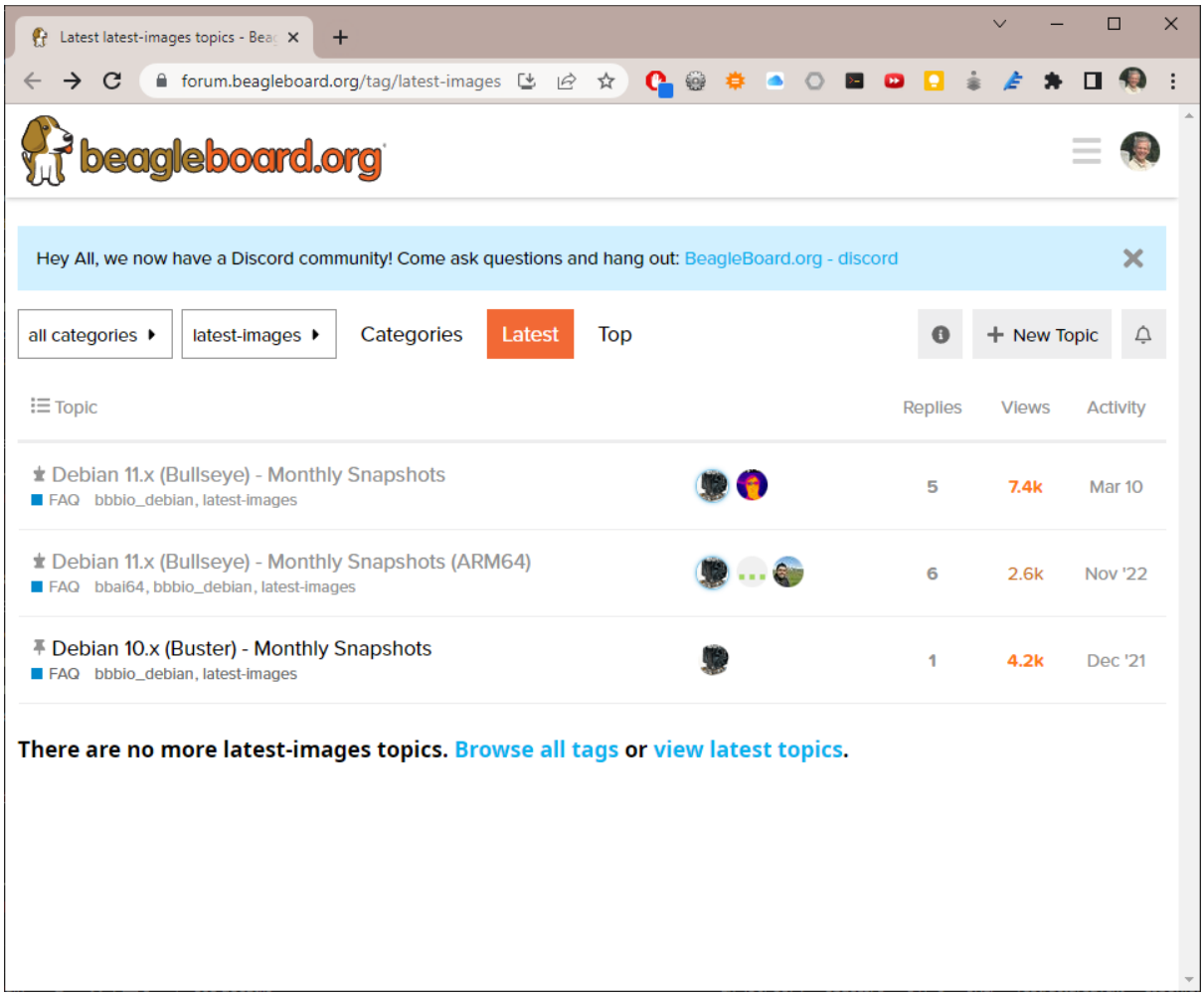


Fig. 15.48: Latest Beagle Images

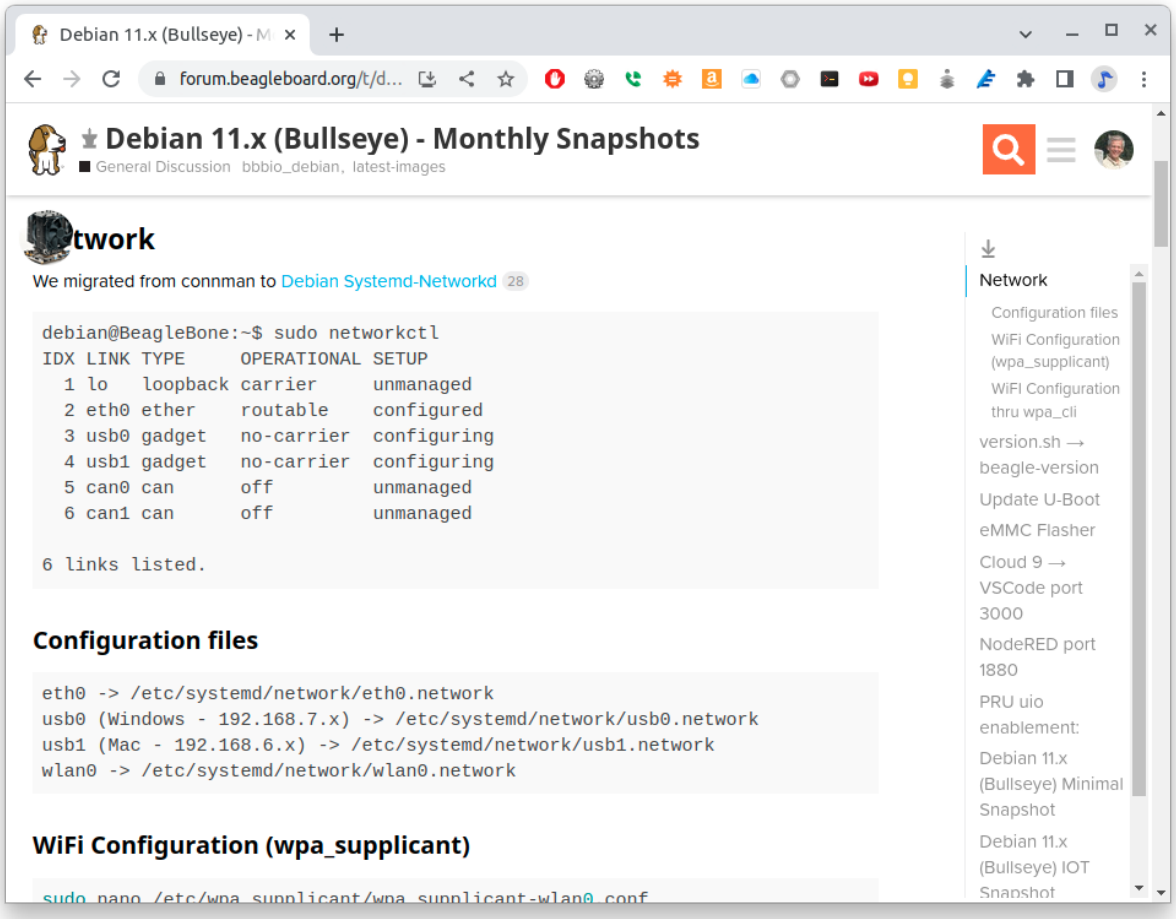


Fig. 15.49: Instructions for setting up your network.

```
bone$ networkctl
IDX LINK    TYPE     OPERATIONAL SETUP
1 lo      loopback carrier     unmanaged
2 eth0    ether    no-carrier  configuring
3 usb0    gadget   routable    configured
4 usb1    gadget   routable    configured
5 can0    can      off         unmanaged
6 can1    can      off         unmanaged
7 wlan0   wlan     routable    configured
8 SoftAp0 wlan     routable    configured

8 links listed.
```

If no name appears, try *ip a*:

```
bone$ ip a
...
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state␣
↪DOWN group default qlen 1000
   link/ether c8:a0:30:a6:26:e8 brd ff:ff:ff:ff:ff:ff
3: usb0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state␣
↪UP group default qlen 1000
   link/ether c2:3f:44:bb:41:0f brd ff:ff:ff:ff:ff:ff
   inet 192.168.7.2/24 brd 192.168.7.255 scope global usb0
      valid_lft forever preferred_lft forever
   inet6 fe80::c03f:44ff:febb:410f/64 scope link
      valid_lft forever preferred_lft forever
...
7: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group␣
↪default qlen 1000
   link/ether 64:69:4e:7e:5c:e4 brd ff:ff:ff:ff:ff:ff
   inet 10.0.7.21/24 brd 10.0.7.255 scope global dynamic wlan0
      valid_lft 85166sec preferred_lft 85166sec
   inet6 fe80::6669:4eff:fe7e:5ce4/64 scope link
      valid_lft forever preferred_lft forever


Next edit the configuration file */etc/wpa_supplicant/wpa_supplicant-wlan0.
↪conf*.
```

```
bone$ sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

In the file you'll see:

```
ctrl_interface=DIR=/run/wpa_supplicant GROUP=netdev
update_config=1
#country=US

network={
     ssid="Your SSID"
     psk="Your Password"
}
```

Change the *ssid* and *psk* entries for your network. Save your file, then run:

```
bone$ sudo systemctl restart systemd-networkd
bone$  ip a
bone$ ping -c2 google.com
PING google.com (142.250.191.206) 56(84) bytes of data.
64 bytes from ord38s31-in-f14.1e100.net (142.250.191.206): icmp_seq=1␣
↪ttl=115 time=19.5 ms
64 bytes from ord38s31-in-f14.1e100.net (142.250.191.206): icmp_seq=2␣
↪ttl=115 time=19.4 ms
```

(continues on next page)

(continued from previous page)

```
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 19.387/19.450/19.513/0.063 ms
```

wlan0 should now have an ip address and you should be on the network. If not, try rebooting.

Sharing the Host's Internet Connection over USB

Problem Your host computer is connected to the Bone via the USB cable, and you want to run the network between the two.

Solution *Establishing an Ethernet-Based Internet Connection* shows how to connect BeagleBone Black to the Internet via the RJ45 Ethernet connector. This recipe shows a way to connect without using the RJ45 connector.

A network is automatically running between the Bone and the host computer at boot time using the USB. The host's IP address is *192.168.7.1* and the Bone's is *192.168.7.2*. Although your Bone is talking to your host, it can't reach the Internet in general, nor can the Internet reach it. On one hand, this is good, because those who are up to no good can't access your Bone. On the other hand, your Bone can't reach the rest of the world.

Letting your bone see the world: setting up IP masquerading You need to set up IP masquerading on your host and configure your Bone to use it. Here is a solution that works with a host computer running Linux. Add the code in *Code for IP Masquerading (ipMasquerade.sh)* to a file called `ipMasquerade.sh` on your host computer.

Listing 15.34: Code for IP Masquerading (ipMasquerade.sh)

```
#!/bin/bash
# These are the commands to run on the host to set up IP
#  masquerading so the Bone can access the Internet through
#  the USB connection.
# This configures the host, run ./setDNS.sh to configure the Bone.
# Inspired by http://thoughtshubham.blogspot.com/2010/03/
#  internet-over-usb-otg-on-beagleboard.html

if [ $# -eq 0 ] ; then
echo "Usage: $0 interface (such as eth0 or wlan0)"
exit 1
fi

interface=$1
hostAddr=192.168.7.1
beagleAddr=192.168.7.2
ip_forward=/proc/sys/net/ipv4/ip_forward

if [ `cat $ip_forward` == 0 ]
  then
    echo "You need to set IP forwarding. Edit /etc/sysctl.conf using:"
    echo "$ sudo nano /etc/sysctl.conf"
    echo "and uncomment the line   \"net.ipv4.ip_forward=1\""
    echo "to enable forwarding of packets. Then run the following:"
    echo "$ sudo sysctl -p"
    exit 1
  else
    echo "IP forwarding is set on host."
fi
# Set up IP masquerading on the host so the bone can reach the outside world
sudo iptables -t nat -A POSTROUTING -s $beagleAddr -o $interface -j␣
↪MASQUERADE
```

ipMasquerade.sh

Then, on your host, run the following commands:

```
host$ chmod +x ipMasquerade.sh
host$ ./ipMasquerade.sh eth0
```

This will direct your host to take requests from the Bone and send them to *eth0*. If your host is using a wireless connection, change *eth0* to *wlan0*.

Now let's set up your host to instruct the Bone what to do. Add the code in *Code for setting the DNS on the Bone (setDNS.sh)* to setDNS.sh on your host computer.

Listing 15.35: Code for setting the DNS on the Bone (setDNS.sh)

```
#!/bin/bash
# These are the commands to run on the host so the Bone
#  can access the Internet through the USB connection.
# Run ./ipMasquerade.sh the first time. It will set up the host.
# Run this script if the host is already set up.
# Inspired by http://thoughtshubham.blogspot.com/2010/03/internet-over-usb-otg-on-beagleboard.html

hostAddr=192.168.7.1
beagleAddr=${1:-192.168.7.2}

# Save the /etc/resolv.conf on the Beagle in case we mess things up.
ssh root@$beagleAddr "mv -n /etc/resolv.conf /etc/resolv.conf.orig"
# Create our own resolv.conf
cat - << EOF > /tmp/resolv.conf
# This is installed by ./setDNS.sh on the host

EOF

TMP=/tmp/nmcli
# Look up the nameserver of the host and add it to our resolv.conf
# From: http://askubuntu.com/questions/197036/how-to-know-what-dns-am-i-using-in-ubuntu-12-04
# Use nmcli dev list for older version nmcli
# Use nmcli dev show for newer version nmcli
nmcli dev show > $TMP
if [ $? -ne 0 ]; then   # $? is the return code, if not 0 something bad happened.
    echo "nmcli failed, trying older 'list' instead of 'show'"
    nmcli dev list > $TMP
    if [ $? -ne 0 ]; then
        echo "nmcli failed again, giving up..."
        exit 1
    fi
fi

grep IP4.DNS $TMP | sed 's/IP4.DNS\[.\]:/nameserver/' >> /tmp/resolv.conf

scp /tmp/resolv.conf root@$beagleAddr:/etc

# Tell the beagle to use the host as the gateway.
ssh root@$beagleAddr "/sbin/route add default gw $hostAddr" || true
```

setDNS.sh

Then, on your host, run the following commands:

```
host$ chmod +x setDNS.sh
host$ ./setDNS.sh
```

(continues on next page)

(continued from previous page)

```
host$ ssh -X root@192.168.7.2
bone$ ping -c2 google.com
PING google.com (216.58.216.96) 56(84) bytes of data.
64 bytes from ord30s22....net (216.58.216.96): icmp_req=1 ttl=55 time=7.49 ms
64 bytes from ord30s22....net (216.58.216.96): icmp_req=2 ttl=55 time=7.62 ms

--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 7.496/7.559/7.623/0.107 ms
```

This will look up what Domain Name System (DNS) servers your host is using and copy them to the right place on the Bone. The *ping* command is a quick way to verify your connection.

Letting the world see your bone: setting up port forwarding Now your Bone can access the world via the USB port and your host computer, but what if you have a web server on your Bone that you want to access from the world? The solution is to use port forwarding from your host. Web servers typically listen to port *80*. First, look up the IP address of your host:

```
host$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group␣
↪default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1280 qdisc mq state UP group␣
↪default qlen 1000
   link/ether 00:15:5d:7c:e8:dc brd ff:ff:ff:ff:ff:ff
   inet 172.31.43.210/20 brd 172.31.47.255 scope global eth0
      valid_lft forever preferred_lft forever
   inet6 fe80::215:5dff:fe7c:e8dc/64 scope link
      valid_lft forever preferred_lft forever
```

It's the number following *inet*, which in my case is *172.31.43.210*.

Tip: If you are on a wireless network, find the IP address associated with *wlan0*.

Then run the following, using your host's IP address:

```
host$ sudo iptables -t nat -A PREROUTING -p tcp -s 0/0 \
     -d 172.31.43.210 --dport 1080 -j DNAT --to 192.168.7.2:80
```

Now browse to your host computer at port *1080*. That is, if your host's IP address is *123.456.789.0*, enter *123.456.789.0:1080*. The *:1080* specifies what port number to use. The request will be forwarded to the server on your Bone listening to port *80*. (I used *1080* here, in case your host is running a web server of its own on port *80*.)

Setting Up a Firewall

Problem You have put your Bone on the network and want to limit which IP addresses can access it.

Solution How-To Geek has a great posting on how do use *ufw*, the "uncomplicated firewall". Check out How to Secure Your Linux Server with a UFW Firewall. I'll summarize the initial setup here.

First install and check the status:

```
bone$ sudo apt update
bone$ sudo apt install ufw
bone$ sudo ufw status
Status: inactive
```

Now turn off everything coming in and leave on all outgoing. Note, this won't take effect until *ufw* is enabled.

```
bone$ sudo ufw default deny incoming
bone$ sudo ufw default allow outgoing
```

Don't enable yet, make sure *ssh* still has access

```
bone$ sudo ufw allow 22
```

Just to be sure, you can install *nmap* on your host computer to see what ports are currently open.

```
host$ sudo apt update
host$ sudo apt install nmap
host$ nmap 192.168.7.2
Starting Nmap 7.80 ( https://nmap.org ) at 2022-07-09 13:37 EDT
Nmap scan report for bone (192.168.7.2)
Host is up (0.014s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
3000/tcp open  ppp

Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
```

Currently there are three ports visible: 22, 80 and 3000 (visual studio code). Now turn on the firewall and see what happens.

```
bone$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup


host$ nmap 192.168.7.2
Starting Nmap 7.80 ( https://nmap.org ) at 2022-07-09 13:37 EDT
Nmap scan report for bone (192.168.7.2)
Host is up (0.014s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
```

Only port 22 (ssh) is accessible now.

The firewall will remain on, even after a reboot. Disable it now if you don't want it on.

```
bone$ sudo ufw disable
Firewall stopped and disabled on system startup
```

See the How-To Geek article for more examples.

Installing Additional Packages from the Debian Package Feed

Problem You want to do more cool things with your BeagleBone by installing more programs.

Warning: Your Bone needs to be on the network for this to work. See *Establishing an Ethernet-Based Internet Connection*, *Establishing a WiFi-Based Internet Connection*, or *Sharing the Host's Internet Connection over USB*.

Solution The easiest way to install more software is to use **apt**:

```
bone$ sudo apt update
bone$ sudo apt install "name of software"
```

A *sudo* is necessary since you aren't running as *root*. The first command downloads package lists from various repositories and updates them to get information on the newest versions of packages and their dependencies. (You need to run it only once a week or so.) The second command fetches the software and installs it and all packages it depends on.

How do you find out what software you can install? Try running this:

```
bone$ apt-cache pkgnames | sort > /tmp/list
bone$ wc /tmp/list
  67974   67974 1369852 /tmp/list
bone$ less /tmp/list
```

The first command lists all the packages that *apt* knows about and sorts them and stores them in `/tmp/list`. The second command shows why you want to put the list in a file. The *wc* command counts the number of lines, words, and characters in a file. In our case, there are over 67,000 packages from which we can choose! The *less* command displays the sorted list, one page at a time. Press the space bar to go to the next page. Press **q** to quit.

Suppose that you would like to install an online dictionary (*dict*). Just run the following command:

```
bone$ sudo apt install dict
```

Now you can run *dict*.

Removing Packages Installed with apt

Problem You've been playing around and installing all sorts of things with *apt* and now you want to clean things up a bit.

Solution *apt* has a *remove* option, so you can run the following command:

```
bone$ sudo apt remove dict
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer␣
↪required:
libmaa3 librecode0 recode
Use 'apt autoremove' to remove them.
The following packages will be REMOVED:
dict
0 upgraded, 0 newly installed, 1 to remove and 27 not upgraded.
After this operation, 164 kB disk space will be freed.
Do you want to continue [Y/n]? y
```

Copying Files Between the Onboard Flash and the MicroSD Card

Problem You want to move files between the onboard flash and the microSD card.

Solution If you booted from the microSD card, run the following command:

```
bone$ df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs          7.2G  2.0G  4.9G  29% /
udev             10M     0   10M   0% /dev
tmpfs           100M  1.9M   98M   2% /run
/dev/mmcblk0p2  7.2G  2.0G  4.9G  29% /
tmpfs           249M     0  249M   0% /dev/shm
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           100M     0  100M   0% /run/user
bone$ ls /dev/mmcblk*
/dev/mmcblk0    /dev/mmcblk0p2  /dev/mmcblk1boot0  /dev/mmcblk1p1
/dev/mmcblk0p1  /dev/mmcblk1    /dev/mmcblk1boot1
```

The *df* command shows what partitions are already mounted. The line `/dev/mmcblk0p2 7.2G 2.0G 4.9G 29% /` shows that *mmcblk0* partition *p2* is mounted as */*, the root file system. The general rule is that the media you're booted from (either the onboard flash or the microSD card) will appear as *mmcblk0*. The second partition (*p2*) is the root of the file system.

The *ls* command shows what devices are available to mount. Because *mmcblk0* is already mounted, */dev/mmcblk1p1* must be the other media that we need to mount. Run the following commands to mount it:

```
bone$ cd /mnt
bone$ sudo mkdir onboard
bone$ ls onboard
bone$ sudo mount /dev/mmcblk1p1 onboard/
bone$ ls onboard
bin   etc     lib         mnt           proc  sbin     sys  var
boot  home    lost+found  nfs-uEnv.txt  root  selinux  tmp
dev   ID.txt  media       opt           run   srv      usr
```

The *cd* command takes us to a place in the file system where files are commonly mounted. The *mkdir* command creates a new directory (`onboard`) to be a mount point. The *ls* command shows there is nothing in `onboard`. The *mount* command makes the contents of the onboard flash accessible. The next *ls* shows there now are files in `onboard`. These are the contents of the onboard flash, which can be copied to and from like any other file.

This same process should also work if you have booted from the onboard flash. When you are done with the onboard flash, you can unmount it by using this command:

```
bone$ sudo umount /mnt/onboard
```

Freeing Space on the Onboard Flash or MicroSD Card

Problem You are starting to run out of room on your microSD card (or onboard flash) and have removed several packages you had previously installed (*Removing Packages Installed with apt*), but you still need to free up more space.

Solution To free up space, you can remove preinstalled packages or discover big files to remove.

Removing preinstalled packages You might not need a few things that come preinstalled in the Debian image, including such things as OpenCV, the Chromium web browser, and some documentation.

Note: The Chromium web browser is the open source version of Google's Chrome web browser. Unless you are using the Bone as a desktop computer, you can probably remove it.

Here's how you can remove these:

```
bone$ sudo apt remove bb-node-red-installer (171M)
bone$ sudo apt autoremove
bone$ sudo -rf /usr/share/doc               (116M)
bone$ sudo -rf /usr/share/man               (19M)
```

Discovering big files The *du* (disk usage) command offers a quick way to discover big files:

```
bone$ sudo du -shx /*
12M  /bin
160M /boot
0    /dev
23M  /etc
835M /home
4.0K /ID.txt
591M /lib
16K  /lost+found
4.0K /media
8.0K /mnt
664M /opt
du: cannot access '/proc/1454/task/1454/fd/4': No such file or directory
du: cannot access '/proc/1454/task/1454/fdinfo/4': No such file or directory
du: cannot access '/proc/1454/fd/3': No such file or directory
du: cannot access '/proc/1454/fdinfo/3': No such file or directory
0    /proc
1.4M /root
1.4M /run
13M  /sbin
4.0K /srv
0    /sys
48K  /tmp
1.6G /usr
1.9G /var
```

If you booted from the microSD card, *du* lists the usage of the microSD. If you booted from the onboard flash, it lists the onboard flash usage.

The *-s* option summarizes the results rather than displaying every file. *-h* prints it in _human_ form–that is, using *M* and *K* postfixes rather than showing lots of digits. The */** specifies to run it on everything in the top-level directory. It looks like a couple of things disappeared while the command was running and thus produced some error messages.

Tip: For more help, try *du –help*.

The `/var` directory appears to be the biggest user of space at 1.9 GB. You can then run the following command to see what's taking up the space in `/var`:

```
bone$ sudo du -sh /var/*
4.0K /var/backups
76M  /var/cache
93M  /var/lib
4.0K /var/local
0    /var/lock
751M /var/log
4.0K /var/mail
4.0K /var/opt
0    /var/run
16K  /var/spool
987M /var/swap
```

(continues on next page)

(continued from previous page)

```
28K  /var/tmp
16K  /var/www
```

A more interactive way to explore your disk usage is by installing *ncdu* (ncurses disk usage):

```
bone$ sudo apt install ncdu
bone$ ncdu /
```

After a moment, you'll see the following:

```
ncdu 1.15.1 ~ Use the arrow keys to navigate, press ? for help
--- / ----------------------------------------------------------------
.   1.9 GiB [##########] /var
    1.5 GiB [########  ] /usr
  835.0 MiB [####      ] /home
  663.5 MiB [###       ] /opt
  590.9 MiB [###       ] /lib
  159.0 MiB [          ] /boot
.  22.8 MiB [          ] /etc
   12.5 MiB [          ] /sbin
   11.1 MiB [          ] /bin
.   1.4 MiB [          ] /run
.  40.0 KiB [          ] /tmp
!  16.0 KiB [          ] /lost+found
    8.0 KiB [          ] /mnt
e   4.0 KiB [          ] /srv
!   4.0 KiB [          ] /root
e   4.0 KiB [          ] /media
    4.0 KiB [          ]  ID.txt
.   0.0   B [          ] /sys
.   0.0   B [          ] /proc
    0.0   B [          ] /dev

Total disk usage:   5.6 GiB  Apparent size:   5.5 GiB  Items: 206148
```

ncdu is a character-based graphics interface to *du*. You can now use your arrow keys to navigate the file structure to discover where the big unused files are. Press ? for help.

Warning: Be careful not to press the **d** key, because it's used to delete a file or directory.

Using C to Interact with the Physical World

Problem You want to use C on the Bone to talk to the world.

Solution The C solution isn't as simple as the JavaScript or Python solution, but it does work and is much faster. The approach is the same, write to the */sys/class/gpio* files.

Listing 15.36: Use C to blink an LED (blinkLED.c)

```
1 ////////////////////////////////////////
2 //        blinkLED.c
3 //        Blinks the P9_14 pin
4 //        Wiring:
5 //        Setup:
6 //        See:
7 ////////////////////////////////////////
8 #include <stdio.h>
9 #include <string.h>
```

(continues on next page)

(continued from previous page)

```
#include <unistd.h>
#define MAXSTR 100
// Look up P9.14 using gpioinfo | grep -e chip -e P9.14.  chip 1, line 18␣
↪maps to 50
int main() {
  FILE *fp;
  char pin[] = "50";
  char GPIOPATH[] = "/sys/class/gpio";
  char path[MAXSTR] = "";

  // Make sure pin is exported
  snprintf(path, MAXSTR, "%s%s%s", GPIOPATH, "/gpio", pin);
  if (!access(path, F_OK) == 0) {
    snprintf(path, MAXSTR, "%s%s", GPIOPATH, "/export");
    fp = fopen(path, "w");
    fprintf(fp, "%s", pin);
    fclose(fp);
  }

  // Make it an output pin
  snprintf(path, MAXSTR, "%s%s%s%s", GPIOPATH, "/gpio", pin, "/direction");
  fp = fopen(path, "w");
  fprintf(fp, "out");
  fclose(fp);

  // Blink every .25 sec
  int state = 0;
  snprintf(path, MAXSTR, "%s%s%s%s", GPIOPATH, "/gpio", pin, "/value");
  fp = fopen(path, "w");
  while (1) {
    fseek(fp, 0, SEEK_SET);
    if (state) {
      fprintf(fp, "1");
    } else {
      fprintf(fp, "0");
    }
    state = ~state;
    usleep(250000);   // sleep time in microseconds
  }
}
```

blinkLED.c

Here, as with JavaScript and Python, the gpio pins are referred to by the Linux gpio number. *Mapping from header pin to internal GPIO number* shows how the P8 and P9 Headers numbers map to the gpio number. For this example P9_14 is used, which the table shows in gpio 50.

Compile and run the code:

```
bone$ gcc -o blinkLED blinkLED.c
bone$ ./blinkLED
^C
```

Hit ^C to stop the blinking.

15.1.6 Internet of Things

You can easily connect BeagleBone Black to the Internet via a wire (*Establishing an Ethernet-Based Internet Connection*), wirelessly (*Establishing a WiFi-Based Internet Connection*), or through the USB to a host and then to the Internet (*Sharing the Host's Internet Connection over USB*). Either way, it opens up a world of possibilities for the "Internet of Things" (IoT).

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	GPIO_113
GPIO_111	29	30	GPIO_112
GPIO_110	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

Fig. 15.50: Mapping from header pin to internal GPIO number

Now that you're online, this chapter offers various things to do with your connection.

Accessing Your Host Computer's Files on the Bone

Problem You want to access a file on a Linux host computer that's attached to the Bone.

Solution If you are running Linux on a host computer attached to BeagleBone Black, it's not hard to mount the Bone's files on the host or the host's files on the Bone by using *sshfs*. Suppose that you want to access files on the host from the Bone. First, install *sshfs*:

```
bone$ sudo apt install sshfs
```

Now, mount the files to an empty directory (substitute your username on the host computer for *username* and the IP address of the host for *192.168.7.1*):

```
bone$ mkdir host
bone$ sshfs username@$192.168.7.1:. host
bone$ cd host
bone$ ls
```

The *ls* command will now list the files in your home directory on your host computer. You can edit them as if they were local to the Bone. You can access all the files by substituting *:/* for the *:.* following the IP address.

You can go the other way, too. Suppose that you are on your Linux host computer and want to access files on your Bone. Install *sshfs*:

```
host$ sudo apt install sshfs
```

and then access:

```
host$ mkdir /mnt/bone
host$ sshfs debian@$192.168.7.2:/ /mnt/bone
host$ cd /mnt/bone
host$ ls
```

Here, we are accessing the files on the Bone as *debian*. We've mounted the entire file system, starting with */*, so you can access any file. Of course, with great power comes great responsibility, so be careful.

The *sshfs* command gives you easy access from one computer to another. When you are done, you can unmount the files by using the following commands:

```
host$ umount /mnt/bone
bone$ umount home
```

Serving Web Pages from the Bone

Problem You want to use BeagleBone Black as a web server.

Solution BeagleBone Black already has the *nginx* web server running.

When you point your browser to *192.168.7.2*, you are using the *nginx* web server. The web pages are served from */var/www/html/*. Add the HTML in *A sample web page (test.html)* to a file called */var/www/html/test.html*, and then point your browser to *192.168.7.2:/test.html*.

Listing 15.37: A sample web page (test.html)

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

`test.html`

You will see the web page shown in *test.html as served by nginx*.

Interacting with the Bone via a Web Browser

Problem BeagleBone Black is interacting with the physical world nicely and you want to display that information on a web browser.

Solution Flask is a Python web framework built with a small core and easy-to-extend philosophy. *Serving Web Pages from the Bone* shows how to use nginx, the web server that's already running. This recipe shows how easy it is to build your own server. This is an adaptation of Python WebServer With Flask and Raspberry Pi.

First, install flask:

```
bone$ sudo apt update
bone$ sudo apt install python3-flask
```

All the code in is the Cookbook repo:

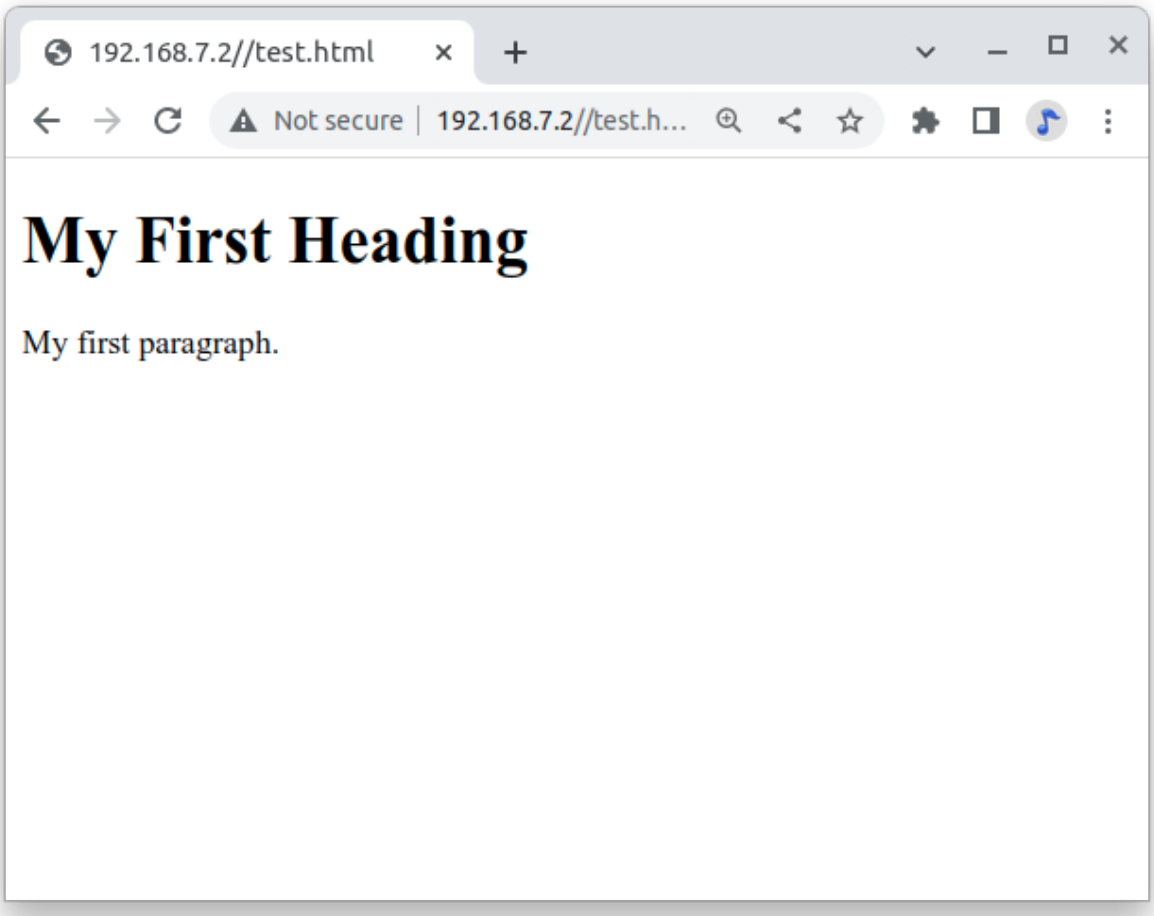


Fig. 15.51: test.html as served by nginx

```
bone$ git clone https://git.beagleboard.org/beagleboard/beaglebone-cookbook-
↪code
bone$ cd beaglebone-cookbook-code/06iot/flask
```

First Flask - hello, world

Our first example is *helloWorld.py*

Listing 15.38: Python code for flask hello world (helloWorld.py)

```
#!/usr/bin/env python
# From: https://towardsdatascience.com/python-webserver-with-flask-and-
↪raspberry-pi-398423cc6f5d

from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
    return 'hello, world'
if __name__ == '__main__':
    app.run(debug=True, port=8080, host='0.0.0.0')
```

`helloWorld.py`

1. The first line loads the Flask module into your Python script.
2. The second line creates a Flask object called `app`.
3. The third line is where the action is, it says to run the index() function when someone accesses the root URL ('/') of the server. In this case, send the text "hello, world" to the client's web browser via return.
4. The last line says to "listen" on port 8080, reporting any errors.

Now on your host computer, browse to 192.168.7.2:8080 flask an you should see.

Adding a template

Let's improve our "hello, world" application, by using an HTML template and a CSS file for styling our page. Note: these have been created for you in the "templates" sub-folder. So, we will create a file named *index1.html*, that has been saved in */templates*.

Here's what's in *templates/index1.html*:

Listing 15.39: index1.html

```
<!DOCTYPE html>
   <head>
      <title>{{ title }}</title>
   </head>
   <body>
      <h1>Hello, World!</h1>
      <h2>The date and time on the server is: {{ time }}</h2>
   </body>
</html>
```

`index1.html`

Note: a style sheet (style.css) is also included. This will be populated later.

Observe that anything in double curly braces within the HTML template is interpreted as a variable that would be passed to it from the Python script via the render_template function. Now, let's create a new Python script. We will name it app1.py:

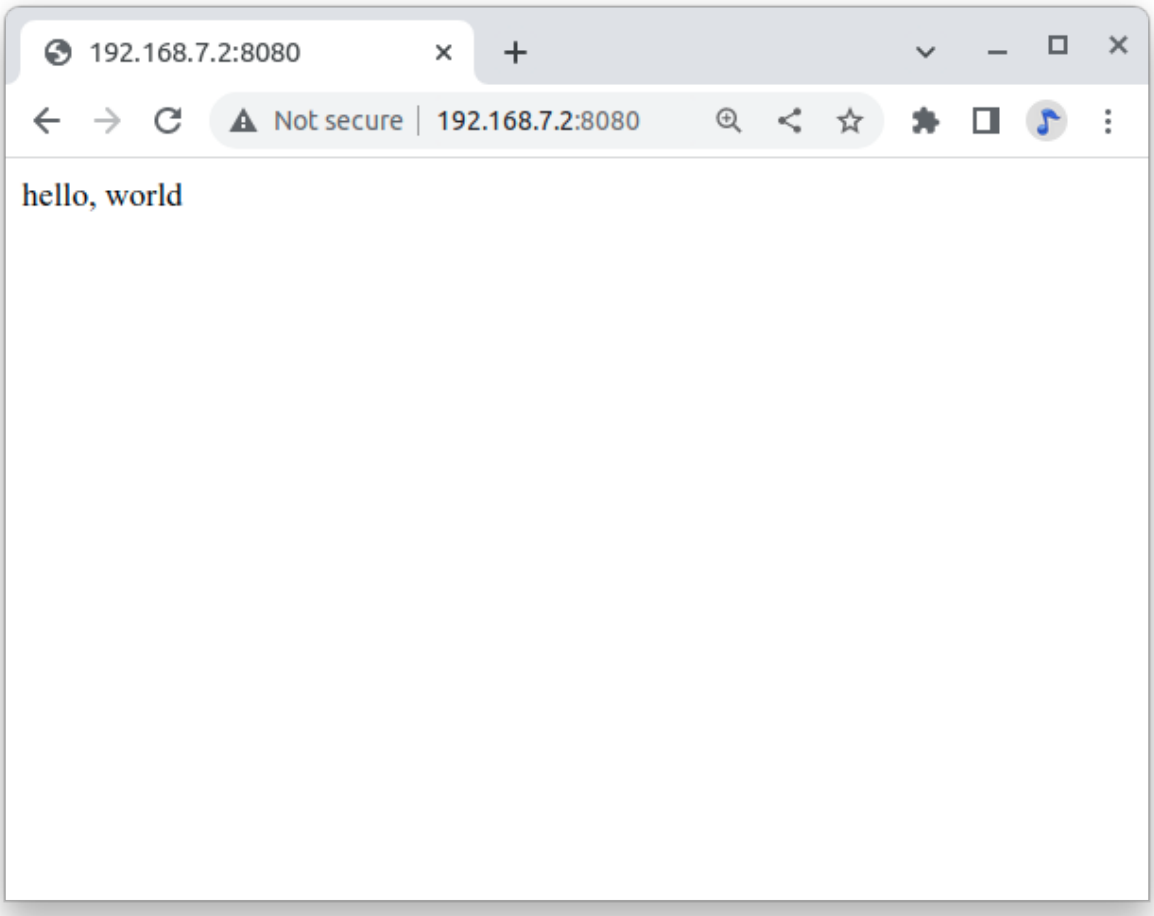


Fig. 15.52: Test page served by our custom flask server

Listing 15.40: app1.py

```
#!/usr/bin/env python
# From: https://towardsdatascience.com/python-webserver-with-flask-and-
↪raspberry-pi-398423cc6f5d

'''
Code created by Matt Richardson
for details, visit:  http://mattrichardson.com/Raspberry-Pi-Flask/inde...
'''
from flask import Flask, render_template
import datetime
app = Flask(__name__)
@app.route("/")
def hello():
   now = datetime.datetime.now()
   timeString = now.strftime("%Y-%m-%d %H:%M")
   templateData = {
      'title' : 'HELLO!',
      'time': timeString
      }
   return render_template('index1.html', **templateData)
if __name__ == "__main__":
   app.run(host='0.0.0.0', port=8080, debug=True)

```

app1.py

Note that we create a formatted string ("timeString") using the date and time from the "now" object, that has the current time stored on it.

Next important thing on the above code, is that we created a dictionary of variables (a set of keys, such as the title that is associated with values, such as HELLO!) to pass into the template. On "return", we will return the *index1.html* template to the web browser using the variables in the *templateData* dictionary.

Execute the Python script:

```
bone$ .\app.py
```

Open any web browser and browse to 192.168.7.2:8080. You should see:

Note that the page's content changes dynamically any time that you refresh it with the actual variable data passed by Python script. In our case, "title" is a fixed value, but "time" changes every minute.

Displaying GPIO Status in a Web Browser - reading a button

Problem You want a web page to display the status of a GPIO pin.

Solution This solution builds on the Flask-based web server solution in *Interacting with the Bone via a Web Browser*.

To make this recipe, you will need:

- Breadboard and jumper wires.
- Pushbutton switch.

Wire your pushbutton as shown in *Diagram for wiring a pushbutton and magnetic reed switch input*. Wire a button to *P9_11* and have the web page display the value of the button.

Let's use a new Python script named *app2.py*.

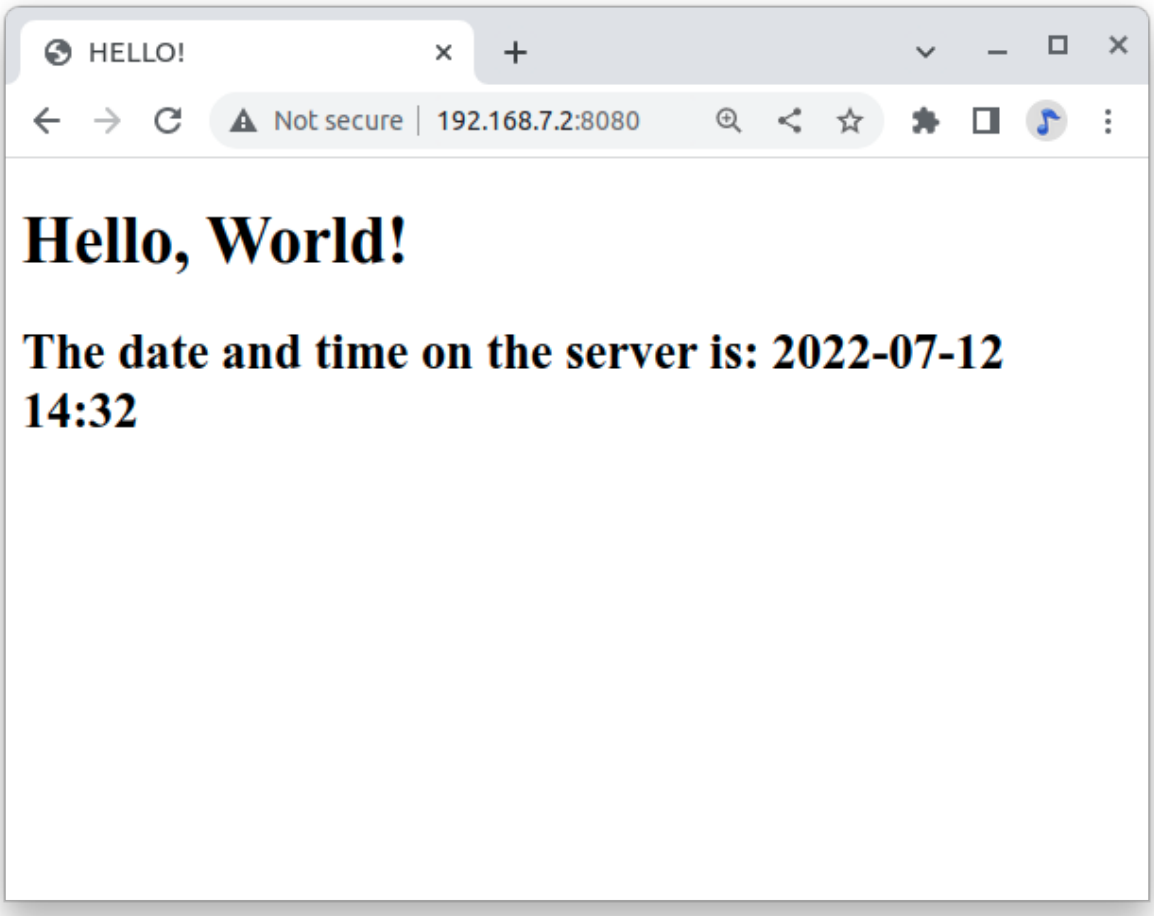


Fig. 15.53: Test page served by app1.py

Listing 15.41: A simple Flask-based web server to read a GPIO (app2.py)

```
#!/usr/bin/env python
# From: https://towardsdatascience.com/python-webserver-with-flask-and-raspberry-pi-398423cc6f5d
import os
from flask import Flask, render_template
app = Flask(__name__)

pin = '30' #  P9_11 is gpio 30
GPIOPATH="/sys/class/gpio"
buttonSts = 0

# Make sure pin is exported
if (not os.path.exists(GPIOPATH+"/gpio"+pin)):
    f = open(GPIOPATH+"/export", "w")
    f.write(pin)
    f.close()

# Make it an input pin
f = open(GPIOPATH+"/gpio"+pin+"/direction", "w")
f.write("in")
f.close()

@app.route("/")
def index():
        # Read Button Status
        f = open(GPIOPATH+"/gpio"+pin+"/value", "r")
        buttonSts = f.read()[:-1]
        f.close()

        # buttonSts = GPIO.input(button)
        templateData = {
      'title' : 'GPIO input Status!',
      'button'  : buttonSts,
      }
        return render_template('index2.html', **templateData)
if __name__ == "__main__":
   app.run(host='0.0.0.0', port=8080, debug=True)
```

app2.py

What we are doing is defining the button on *P9_11* as input, reading its value and storing it in *buttonSts*. Inside the function *index()*, we will pass that value to our web page through "button" that is part of our variable dictionary: *templateData*.

Let's also see the new *index2.html* to show the GPIO status:

Listing 15.42: A simple Flask-based web server to read a GPIO (index2.html)

```
<!DOCTYPE html>
   <head>
      <title>{{ title }}</title>
      <link rel="stylesheet" href='../static/style.css'/>
   </head>
   <body>
         <h1>{{ title }}</h1>
      <h2>Button pressed:  {{ button }}</h1>
   </body>
</html>
```

index2.html

Now, run the following command:

```
bone$ ./app2.py
```

Point your browser to *http://192.168.7.2:8080*, and the page will look like *Status of a GPIO pin on a web page*.

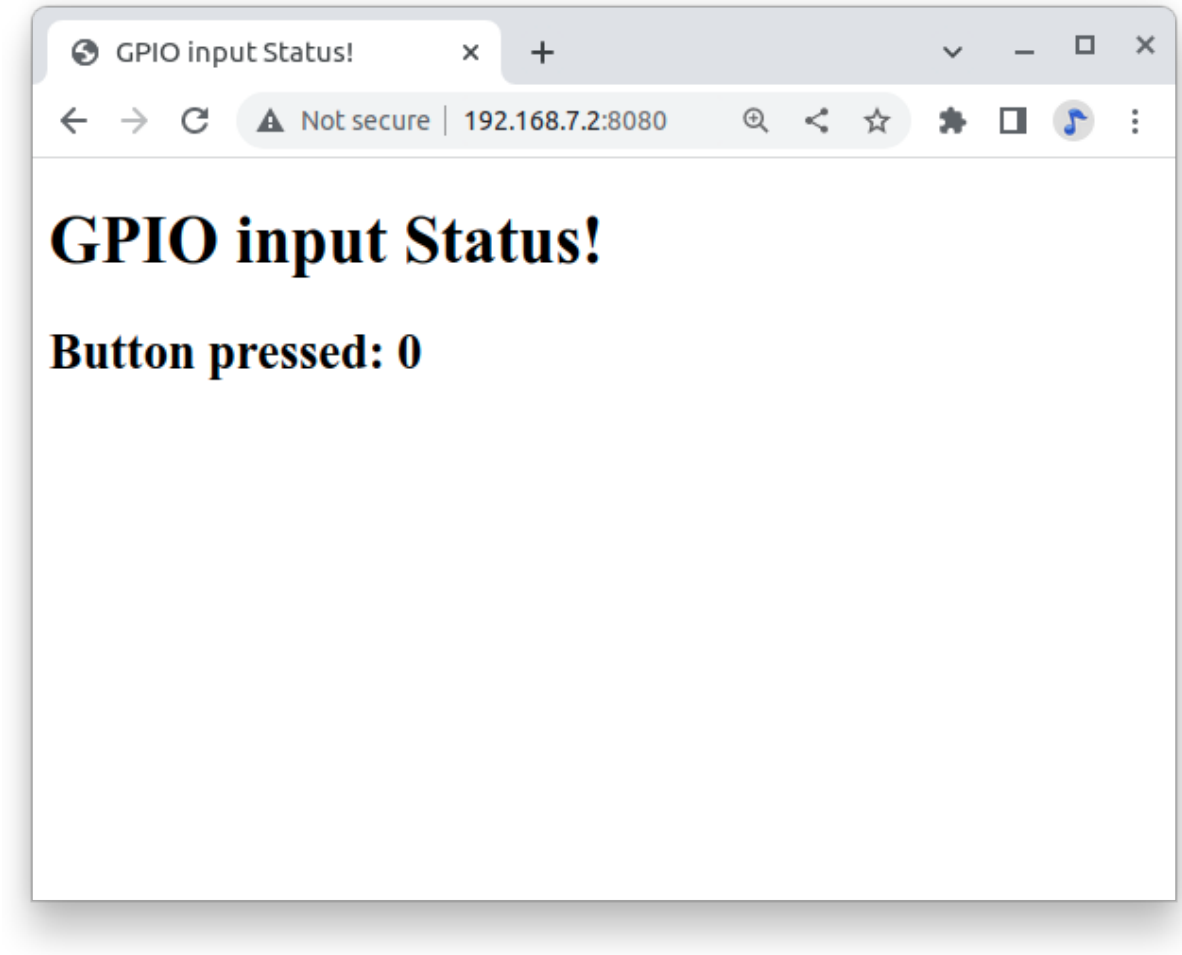


Fig. 15.54: Status of a GPIO pin on a web page

Currently, the *0* shows that the button isn't pressed. Try refreshing the page while pushing the button, and you will see *1* displayed.

It's not hard to assemble your own HTML with the GPIO data. It's an easy extension to write a program to display the status of all the GPIO pins.

Controlling GPIOs

Problem You want to control an LED attached to a GPIO pin.

Solution Now that we know how to "read" GPIO Status, let's change them. What we will do will control the LED via the web page. We have an LED connected to *P9_14*. Controlling remotely we will change its status from LOW to HIGH and vice-versa.

Create a new Python script and name it *app3.py*.

Listing 15.43: A simple Flask-based web server to read a GPIO (app3.py)

```
#!/usr/bin/env python
# From: https://towardsdatascience.com/python-webserver-with-flask-and-raspberry-pi-398423cc6f5d
# import Adafruit_BBIO.GPIO as GPIO
import os
from flask import Flask, render_template, request
app = Flask(__name__)
#define LED GPIO
ledRed = "P9_14"
pin = '50' #  P9_14 is gpio 50
GPIOPATH="/sys/class/gpio"

#initialize GPIO status variable
ledRedSts = 0
# Make sure pin is exported
if (not os.path.exists(GPIOPATH+"/gpio"+pin)):
    f = open(GPIOPATH+"/export", "w")
    f.write(pin)
    f.close()
# Define led pin as output
f = open(GPIOPATH+"/gpio"+pin+"/direction", "w")
f.write("out")
f.close()
# turn led OFF
f = open(GPIOPATH+"/gpio"+pin+"/value", "w")
f.write("0")
f.close()

@app.route("/")
def index():
        # Read Sensors Status
        f = open(GPIOPATH+"/gpio"+pin+"/value", "r")
        ledRedSts = f.read()
        f.close()
        templateData = {
              'title' : 'GPIO output Status!',
              'ledRed'  : ledRedSts,
        }
        return render_template('index3.html', **templateData)

@app.route("/<deviceName>/<action>")
def action(deviceName, action):
        if deviceName == 'ledRed':
                actuator = ledRed
        f = open(GPIOPATH+"/gpio"+pin+"/value", "w")
        if action == "on":
                f.write("1")
        if action == "off":
                f.write("0")
```

(continues on next page)

(continued from previous page)

```
        f.close()

        f = open(GPIOPATH+"/gpio"+pin+"/value", "r")
        ledRedSts = f.read()
        f.close()

        templateData = {
              'ledRed'   : ledRedSts,
        }
        return render_template('index3.html', **templateData)
if __name__ == "__main__":
   app.run(host='0.0.0.0', port=8080, debug=True)
```

app3.py

What we have new on above code is the new "route":

@app.route("/<deviceName>/<action>")

From the webpage, calls will be generated with the format:

http://192.168.7.2:8081/ledRed/on

or

http://192.168.7.2:8081/ledRed/off

For the above example, *ledRed* is the "deviceName" and *on* or *off* are examples of possible "action". Those routes will be identified and properly "worked". The main steps are:

- Convert the string "ledRED", for example, on its equivalent GPIO pin. The integer variable ledRed is equivalent to P9_14. We store this value on variable "actuator"
- For each actuator, we will analyze the "action", or "command" and act properly. If "action = on" for example, we must use the command: `f.write("1")`
- Update the status of each actuator
- Return the data to index.html

Let's now create an *index.html* to show the GPIO status of each actuator and more importantly, create "buttons" to send the commands:

Listing 15.44: A simple Flask-based web server to write a GPIO (index3.html)

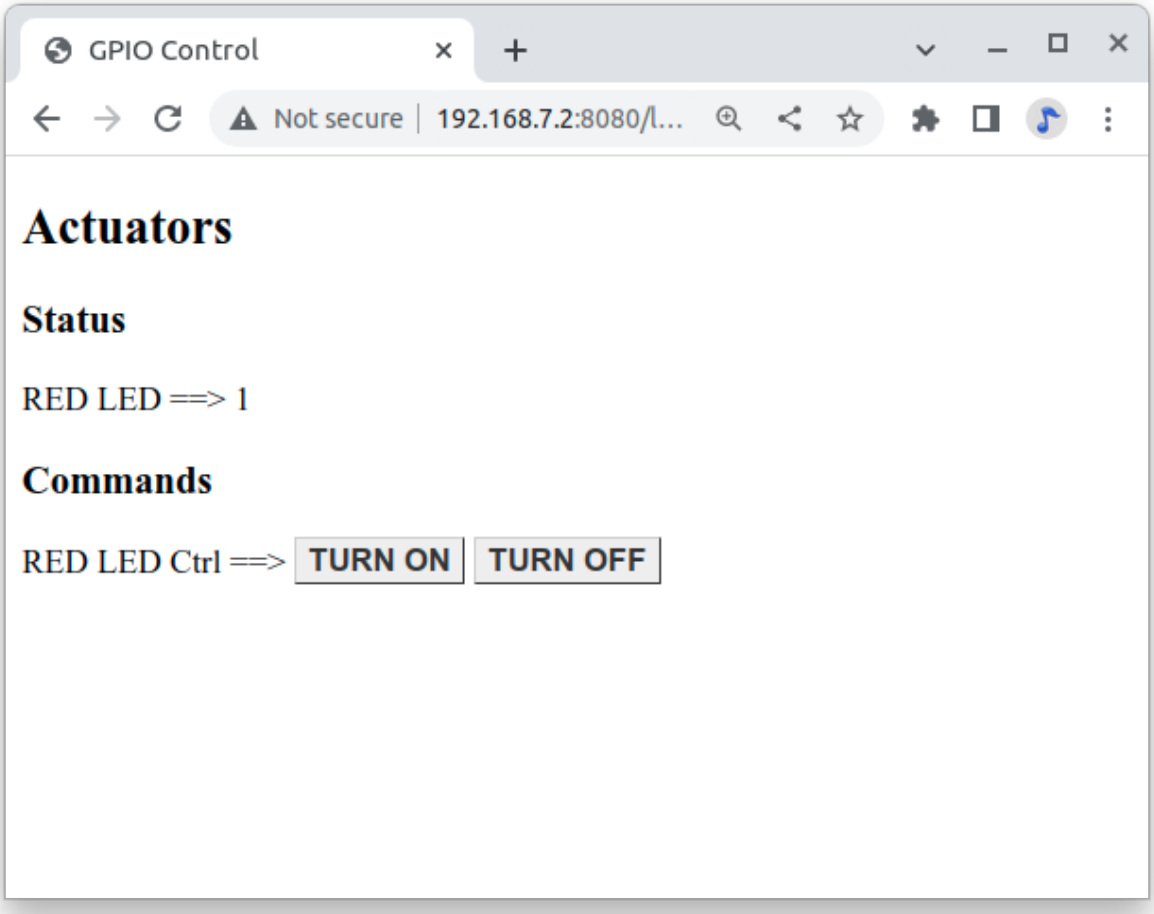
```
<!DOCTYPE html>
   <head>
      <title>GPIO Control</title>
      <link rel="stylesheet" href='../static/style.css'/>
   </head>
   <body>
                  <h2>Actuators</h2>
                  <h3> Status </h3>
                       RED LED ==>  {{ ledRed  }}
                  <br>
                  <h3> Commands </h3>
                          RED LED Ctrl ==>
                          <a href="/ledRed/on" class="button">TURN ON</a>
                          <a href="/ledRed/off"class="button">TURN OFF</a>
   </body>
</html>
```

index3.html

```
bone$ ./app3.py
```

Point your browser as before and you will see:

Status of a GPIO pin on a web page



Try clicking the "TURN ON" and "TURN OFF" buttons and your LED will respond.

app4.py and *app5.py* combine the previous apps. Try them out.

```
app4.py app5.py
```

Plotting Data

Problem You have live, continuous, data coming into your Bone via one of the Analog Ins, and you want to plot it.

Solution

Analog in - Continuous (This is based on information at: http://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/Foundational_Components/Kernel/Kernel_Drivers/ADC.html#Continuous%20Mode)

Reading a continuous analog signal requires some set up. First go to the iio devices directory.

```
bone$ cd /sys/bus/iio/devices/iio:device0
bone$ ls -F
buffer/  in_voltage0_raw  in_voltage2_raw  in_voltage4_raw  in_voltage6_raw
↪name      power/          subsystem@
dev      in_voltage1_raw  in_voltage3_raw  in_voltage5_raw  in_voltage7_raw
↪of_node@  scan_elements/  uevent
```

Here you see the files used to read the one shot values. Look in *scan_elements* to see how to enable continuous input.

```
bone$ ls scan_elements
in_voltage0_en     in_voltage1_index  in_voltage2_type   in_voltage4_en
↪in_voltage5_index  in_voltage6_type
in_voltage0_index  in_voltage1_type   in_voltage3_en     in_voltage4_index
↪in_voltage5_type   in_voltage7_en
in_voltage0_type   in_voltage2_en     in_voltage3_index  in_voltage4_type
↪in_voltage6_en     in_voltage7_index
in_voltage1_en     in_voltage2_index  in_voltage3_type   in_voltage5_en
↪in_voltage6_index  in_voltage7_type
```

Here you see three values for each analog input, _en (enable), _index (index of this channel in the buffer's chunks) and _type (how the ADC stores its data). (See the link above for details.) Let's use the input at *P9.40* which is *AIN1*. To enable this input:

```
bone$ echo 1 > scan_elements/in_voltage1_en
```

Next set the buffer size.

```
bone$ ls buffer
data_available  enable  length  watermark
```

Let's use a 512 sample buffer. You might need to experiment with this.

```
bone$ echo 512 > buffer/length

Then start it running.
```

```
bone$ echo 1 > buffer/enable

Now, just read from */dev/iio:device0*.
```

An example Python program that does the above and reads and plots the buffer is **analogInContinuous.py**.

Listing 15.45: Code to read and plot a continuous analog input(analogInContinuous.py)

```
1  #!/usr/bin/python
2  ##########################################
3  #        analogInContinuous.py
4  #         Read analog data via IIO continuous mode and plots it.
5  ##########################################
6  # From: https://stackoverflow.com/questions/20295646/python-ascii-plots-in-
   ↪terminal
7  # https://github.com/dkogan/gnuplotlib
8  # https://github.com/dkogan/gnuplotlib/blob/master/guide/guide.org
9  # sudo apt install gnuplot  (10 minute to install)
10 # sudo apt install libatlas-base-dev
11 # pip3 install gnuplotlib
12 # This uses X11, so when connecting to the bone from the host use:  ssh -X
   ↪bone
13
14 # See https://elinux.org/index.php?title=EBC_Exercise_10a_Analog_In#Analog_
   ↪in_-_Continuous.2C_Change_the_sample_rate
```

(continues on next page)

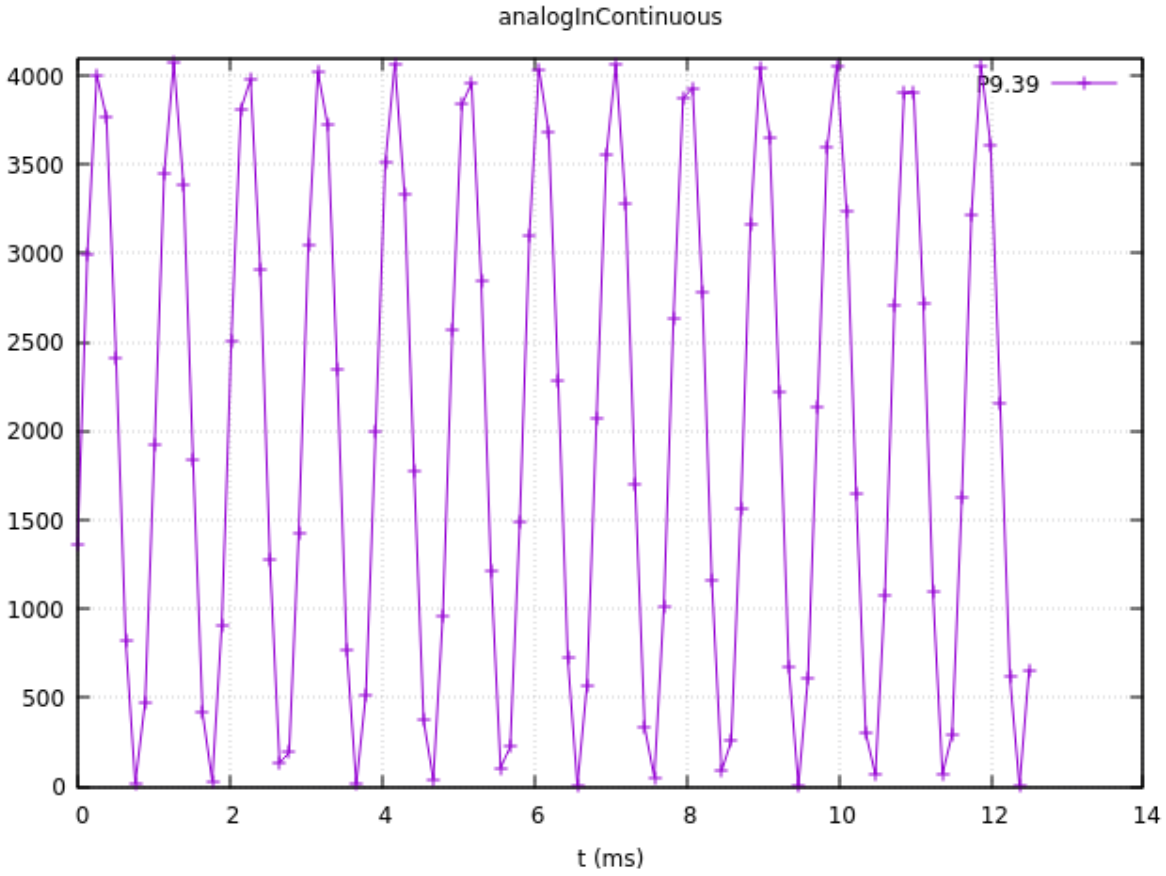


Fig. 15.55: 1KHz sine wave sampled at 8KHz

(continued from previous page)

```
# for instructions on changing the sampling rate.  Can go up to 200KHz.

fd = open(IIODEV, "r")
import numpy      as np
import gnuplotlib as gp
import time
# import struct

IIOPATH='/sys/bus/iio/devices/iio:device0'
IIODEV='/dev/iio:device0'
LEN = 100
SAMPLERATE=8000
AIN='2'

# Setup IIO for Continous reading
# Enable AIN
try:
    file1 = open(IIOPATH+'/scan_elements/in_voltage'+AIN+'_en', 'w')
    file1.write('1')
    file1.close()
except:     # carry on if it's already enabled
    pass
# Set buffer length
file1 = open(IIOPATH+'/buffer/length', 'w')
file1.write(str(2*LEN))     # I think LEN is in 16-bit values, but here we pass bytes
file1.close()
# Enable continuous
file1 = open(IIOPATH+'/buffer/enable', 'w')
file1.write('1')
file1.close()

x = np.linspace(0, 1000*LEN/SAMPLERATE, LEN)
# Do a dummy plot to give time of the fonts to load.
gp.plot(x, x)
print("Waiting for fonts to load")
time.sleep(10)

print('Hit ^C to stop')

fd = open(IIODEV, "r")

try:
    while True:
        y = np.fromfile(fd, dtype='uint16', count=LEN)*1.8/4096
        # print(y)
        gp.plot(x, y,
            xlabel = 't (ms)',
            ylabel = 'volts',
            _yrange = [0, 2],
            title  = 'analogInContinuous',
            legend = np.array( ("P9.39", ), ),
            # ascii=1,
            # terminal="xterm",
            # legend = np.array( ("P9.40", "P9.38"), ),
            # _with  = 'lines'
            )

except KeyboardInterrupt:
    print("Turning off input.")
    # Disable continuous
```

(continues on next page)

(continued from previous page)

```
    file1 = open(IIOPATH+'/buffer/enable', 'w')
    file1.write('0')
    file1.close()

    file1 = open(IIOPATH+'/scan_elements/in_voltage'+AIN+'_en', 'w')
    file1.write('0')
    file1.close()

# // Bone  | Pocket | AIN
# // ----- | ------ | ---
# // P9_39 | P1_19  | 0
# // P9_40 | P1_21  | 1
# // P9_37 | P1_23  | 2
# // P9_38 | P1_25  | 3
# // P9_33 | P1_27  | 4
# // P9_36 | P2_35  | 5
# // P9_35 | P1_02  | 6
```

`analogInContinuous.py`

Be sure to read the instillation instructions in the comments. Also note this uses X windows and you need to *ssh -X 192.168.7.2* for X to know where the display is.

Run it:

```
host$ ssh -X bone

bone$ cd beaglebone-cookbook-code/06iot
bone$ ./analogInContinuous.py
Hit ^C to stop
```

1KHz sine wave sampled at 8KHz is the output of a 1KHz sine wave.

It's a good idea to disable the buffer when done.

```
bone$ echo 0 > /sys/bus/iio/devices/iio:device0/buffer/enable
```

Analog in - Continuous, Change the sample rate The built in ADCs sample at 8k samples/second by default. They can run as fast as 200k samples/second by editing a device tree.

```
bone$ cd /opt/source/bb.org-overlays
bone$ make
```

This will take a while the first time as it compiles all the device trees.

```
bone$ vi src/arm/src/arm/BB-ADC-00A0.dts
```

Around line 57 you'll see

```
Line    Code
57      // For each step, number of adc clock cycles to wait between setting␣
 ↪up muxes and sampling.
58      //  range: 0 .. 262143
59      //  optional, default is 152 (XXX but why?!)
60      ti,chan-step-opendelay = <152 152 152 152 152 152 152 152>;
61      //`
62      // XXX is there any purpose to set this nonzero other than to fine-
 ↪tune the sample rate?
63
64
65      // For each step, how many times it should sample to average.
```

(continues on next page)

(continued from previous page)

```
66      //  range: 1 .. 16, must be power of two (i.e. 1, 2, 4, 8, or 16)
67      //  optional, default is 16
68      ti,chan-step-avg = <16 16 16 16 16 16 16 16>;
```

The comments give lots of details on how to adjust the device tree to change the sample rate. Line 68 says for every sample returned, average 16 values. This will give you a cleaner signal, but if you want to go fast, change the 16's to 1's. Line 60 says to delay 152 cycles between each sample. Set this to 0 to got as fast a possible.

```
ti,chan-step-avg = <1 1 1 1 1 1 1 1>;
ti,chan-step-opendelay = <0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00>;
```

Now compile it.

```
bone$ make
  DTC     src/arm/BB-ADC-00A0.dtbo
gcc -o config-pin ./tools/pmunts_muntsos/config-pin.c
```

It knows to only recompile the file you just edited. Now install and reboot.

```
bone$ sudo make install
...
'src/arm/AM335X-PRU-UIO-00A0.dtbo' -> '/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
↪'
'src/arm/BB-ADC-00A0.dtbo' -> '/lib/firmware/BB-ADC-00A0.dtbo'
'src/arm/BB-BBBMINI-00A0.dtbo' -> '/lib/firmware/BB-BBBMINI-00A0.dtbo'
...
bone$ reboot
```

A number of files get installed, including the ADC file. Now try rerunning.

```
bone$ cd beaglebone-cookbook-code/06iot
bone$ ./analogInContinuous.py
Hit ^C to stop
```

Here's the output of a 10KHz triangle wave.

It's still a good idea to disable the buffer when done.

```
bone$ echo 0 > /sys/bus/iio/devices/iio:device0/buffer/enable
```

Sending an Email

Problem You want to send an email via Gmail from the Bone.

Solution This example came from https://realpython.com/python-send-email/. First, you need to set up a Gmail account, if you don't already have one. Then add the code in *Sending email using nodemailer (emailtTest.py)* to a file named `emailTest.py`. Substitute your own Gmail username. For the password:

- Go to: https://myaccount.google.com/security
- Go to *2-Step Verification* and at the bottom, select App password.
- Generate your own 16 char password and copy it into `emailTest.py`.
- Be sure to delete password when done https://myaccount.google.com/apppasswords .

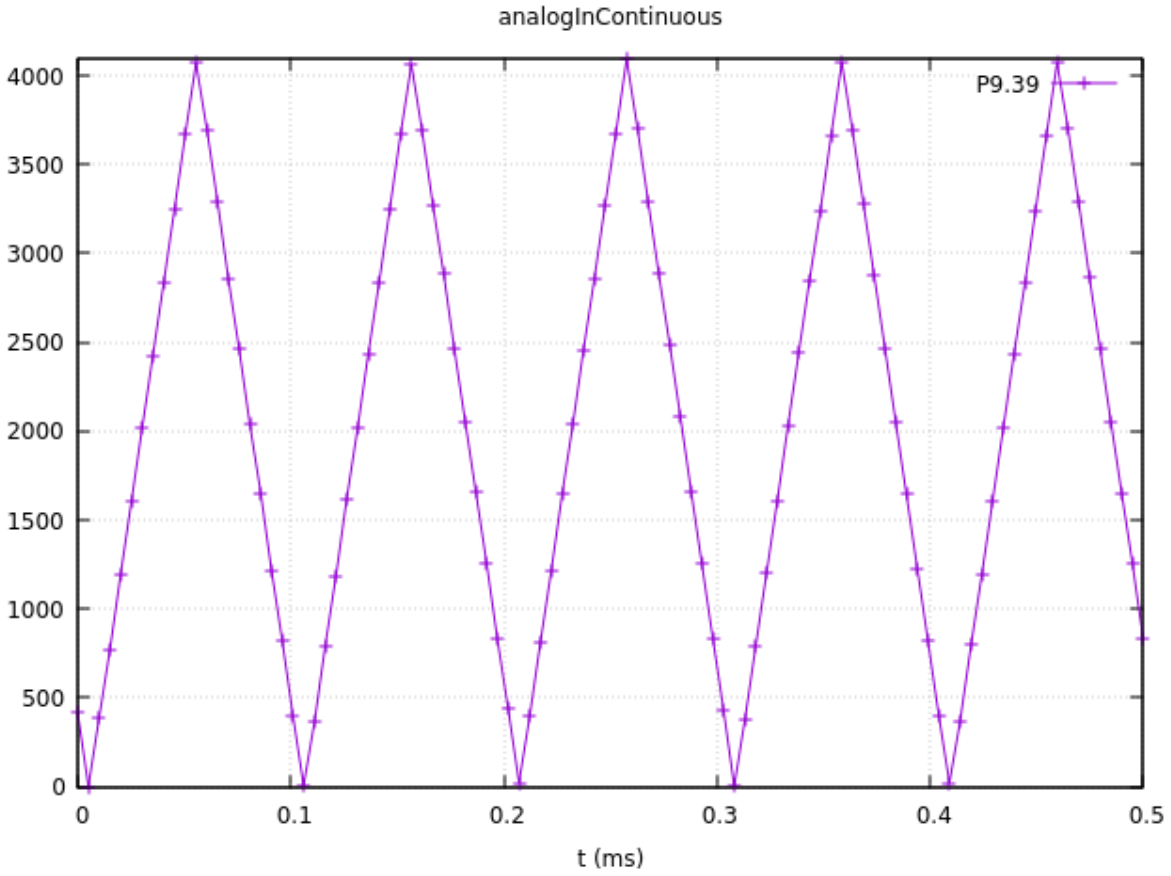


Fig. 15.56: 10KHz triangle wave sampled at 200KHz

Listing 15.46: Sending email using nodemailer (emailtTest.py)

```
#!/usr/bin/env python
# From: https://realpython.com/python-send-email/
import smtplib, ssl

port = 587  # For starttls
smtp_server  = "smtp.gmail.com"
sender_email = "from_account@gmail.com"
receiver_email = "to_account@gmail.com"
# Go to: https://myaccount.google.com/security
# Select App password
# Generate your own 16 char password, copy here
# Delete password when done
password = "cftqhcejjdjfdwjh"
message = """\
Subject: Testing email

This message is sent from Python.

"""
context = ssl.create_default_context()
with smtplib.SMTP(smtp_server, port) as server:
    server.starttls(context=context)
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, message)
```

emailTest.py

Then run the script to send the email:

```
bone$ chmod *x emailTest.py
bone$ .\emailTest.py
```

Warning: This solution requires your Gmail password to be in plain text in a file, which is a security problem. Make sure you know who has access to your Bone. Also, if you remove the microSD card, make sure you know who has access to it. Anyone with your microSD card can read your Gmail password.

Be careful about putting this into a loop. Gmail presently limits you to 500 emails per day and 10 MB per message.

See https://realpython.com/python-send-email/ for an example that sends an attached file.

Sending an SMS Message

Problem You want to send a text message from BeagleBone Black.

Solution There are a number of SMS services out there. This recipe uses Twilio because you can use it for free, but you will need to verify the number to which you are texting. First, go to Twilio's home page and set up an account. Note your account SID and authorization token. If you are using the free version, be sure to verify your numbers.

Next, install Trilio by using the following command for python:

```
bone$ sudo apt install python-pip
bone$ sudo pip install twilio
```

or for Javascript:

```
bone$ npm install -g twilio
```

Finally, add the code in *Sending SMS messages using Twilio (twilioTest.py)* to a file named `twilioTest.py` and run it. Your text will be sent.

Python

JavaScript

Listing 15.47: Sending SMS messages using Twilio (`twilioTest.py`)

```
#!/usr/bin/env python
# Download the helper library from https://www.twilio.com/docs/python/install
import os
from twilio.rest import Client


# Find your Account SID and Auth Token at twilio.com/console
# and set the environment variables. See http://twil.io/secure
account_sid = os.environ['TWILIO_ACCOUNT_SID']
auth_token = os.environ['TWILIO_AUTH_TOKEN']
client = Client(account_sid, auth_token)

message = client.messages \
                .create(
                     body="Join Earth's mightiest heroes. Like Kevin Bacon.",
                     from_='+18122333219',
                     to='+18122333219'
                 )

print(message.sid)
```

`twilioTest.py`

Listing 15.48: Sending SMS messages using Twilio (`twilio-test.js`)

```
#!/usr/bin/env node
// From: http://twilio.github.io/twilio-node/
// Twilio Credentials
var accountSid = '';
var authToken = '';

//require the Twilio module and create a REST client
var client = require('twilio')(accountSid, authToken);

client.messages.create({
        to: "812555121",
        from: "+2605551212",
        body: "This is a test",
}, function(err, message) {
        console.log(message.sid);
});

// https://github.com/twilio/twilio-node/blob/master/LICENSE
```

`twilio-test.js`

Twilio allows a small number of free text messages, enough to test your code and to play around some.

Displaying the Current Weather Conditions

Problem You want to display the current weather conditions.

Solution Because your Bone is on the network, it's not hard to access the current weather conditions from a weather API.

- Go to https://openweathermap.org/ and create an account.
- Go to https://home.openweathermap.org/api_keys and get your API key.
- Store your key in the *bash* variable *APPID*.

```
bash$ export APPID="Your key"
```

- Then add the code in *Code for getting current weather conditions (weather.py)* to a file named `weather.py`.
- Run the python script.

Listing 15.49: Code for getting current weather conditions (`weather.py`)

```
#!/usr/bin/env python3
# Displays current weather and forecast
import os
import sys
from datetime import datetime
import requests     # For getting weather

# http://api.openweathermap.org/data/2.5/onecall
params = {
    'appid': os.environ['APPID'],
    # 'city': 'brazil,indiana',
    'exclude': "minutely,hourly",
    'lat':  '39.52',
    'lon': '-87.12',
    'units': 'imperial'
    }
urlWeather = "http://api.openweathermap.org/data/2.5/onecall"

print("Getting weather")

try:
    r = requests.get(urlWeather, params=params)
    if(r.status_code==200):
        # print("headers: ", r.headers)
        # print("text: ", r.text)
        # print("json: ", r.json())
        weather = r.json()
        print("Temp: ", weather['current']['temp'])          # ①
        print("Humid:", weather['current']['humidity'])
        print("Low:  ", weather['daily'][1]['temp']['min'])
        print("High: ", weather['daily'][0]['temp']['max'])
        day = weather['daily'][0]['sunrise']-weather['timezone_offset']
        print("sunrise: " + datetime.utcfromtimestamp(day).strftime('%Y-%m-
↪%d %H:%M:%S'))
        # print("Day: " + datetime.utcfromtimestamp(day).strftime('%a'))
        # print("weather: ", weather['daily'][1])            # ②
        # print("weather: ", weather)                        # ③
        # print("icon: ", weather['current']['weather'][0]['icon'])
        # print()

```

(continues on next page)

(continued from previous page)

```
    else:
        print("status_code: ", r.status_code)
except IOError:
    print("File not found: " + tmp101)
    print("Have you run setup.sh?")
except:
    print("Unexpected error:", sys.exc_info())
```

weather.py

1. Prints current conditions.
2. Prints the forecast for the next day.
3. Prints everything returned by the weather site.

Uncomment what you want to be displayed.

Run this by using the following commands:

```
bone$ ./weather.js
Getting weather
Temp:  73.72
Humid: 31
Low:   54.21
High:  75.47
sunrise: 2023-06-09 14:21:07
```

The weather API returns lots of information. Use Python to extract the information you want.

Sending and Receiving Tweets

Problem You want to send and receive tweets (Twitter posts) with your Bone.

Solution Twitter has a whole git repo of sample code for interacting with Twitter. Here I'll show how to create a tweet and then how to delete it.

Creating a Project and App

- Follow the directions here to create a project and app.
- Be sure to give your app Read and Write permission.
- Then go to the developer portal and select you app by clicking on the gear icon to the right of the app name.
- Click on the *Keys and tokens* tab. Here you can get to all your keys and tokens.

Tip: Be sure to record them, you can't get them later.

- Open the file *twitterKeys.sh* and record your keys in it.

```
export API_KEY='XXX'
export API_SECRET_KEY='XXX'
export BEARER_TOKEN='XXX'
export TOKEN='XXXX'
export TOKEN_SECRET='XXX'
```

- Next, source the file so the values will appear in your bash session.

```
bash$ source twitterKeys.sh
```

You'll need to do this every time you open a new *bash* window.

Creating a tweet

Add the code in *Create a Tweet (twitter_create_tweet.py)* to a file called `twitter_create_tweet_.py` and run it to see your timeline.

Listing 15.50: Create a Tweet (`twitter_create_tweet.py`)

```
#!/usr/bin/env python
# From: https://github.com/twitterdev/Twitter-API-v2-sample-code/blob/main/Manage-Tweets/create_tweet.py
from requests_oauthlib import OAuth1Session
import os
import json

# In your terminal please set your environment variables by running the following lines of code.
# export 'API_KEY'='<your_consumer_key>'
# export 'API_SECRET_KEY'='<your_consumer_secret>'

consumer_key = os.environ.get("API_KEY")
consumer_secret = os.environ.get("API_SECRET_KEY")

# Be sure to add replace the text of the with the text you wish to Tweet. You can also add parameters to post polls, quote Tweets, Tweet with reply settings, and Tweet to Super Followers in addition to other features.
payload = {"text": "Hello world!"}

# Get request token
request_token_url = "https://api.twitter.com/oauth/request_token?oauth_callback=oob&x_auth_access_type=write"
oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

try:
    fetch_response = oauth.fetch_request_token(request_token_url)
except ValueError:
    print(
        "There may have been an issue with the consumer_key or consumer_secret you entered."
    )

resource_owner_key = fetch_response.get("oauth_token")
resource_owner_secret = fetch_response.get("oauth_token_secret")
print("Got OAuth token: %s" % resource_owner_key)

# Get authorization
base_authorization_url = "https://api.twitter.com/oauth/authorize"
authorization_url = oauth.authorization_url(base_authorization_url)
print("Please go here and authorize: %s" % authorization_url)
verifier = input("Paste the PIN here: ")

# Get the access token
access_token_url = "https://api.twitter.com/oauth/access_token"
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=resource_owner_key,
    resource_owner_secret=resource_owner_secret,
```

(continues on next page)

(continued from previous page)

```
45      verifier=verifier,
46  )
47  oauth_tokens = oauth.fetch_access_token(access_token_url)
48
49  access_token = oauth_tokens["oauth_token"]
50  access_token_secret = oauth_tokens["oauth_token_secret"]
51
52  # Make the request
53  oauth = OAuth1Session(
54      consumer_key,
55      client_secret=consumer_secret,
56      resource_owner_key=access_token,
57      resource_owner_secret=access_token_secret,
58  )
59
60  # Making the request
61  response = oauth.post(
62      "https://api.twitter.com/2/tweets",
63      json=payload,
64  )
65
66  if response.status_code != 201:
67      raise Exception(
68          "Request returned an error: {} {}".format(response.status_code,␣
    ↪response.text)
69      )
70
71  print("Response code: {}".format(response.status_code))
72
73  # Saving the response as JSON
74  json_response = response.json()
75  print(json.dumps(json_response, indent=4, sort_keys=True))
```

twitter_create_tweet.py

Run the code and you'll have to authorize.

```
bash$ ./twitter_create_tweet.py
Got OAuth token: tWBldQAAAAAAWBJgAAABggJt7qg
Please go here and authorize: https://api.twitter.com/oauth/authorize?oauth_
↪token=tWBldQAAAAAAWBJgAAABggJt7qg
Paste the PIN here: 4859044
Response code: 201
{
    "data": {
        "id": "1547963178700533760",
        "text": "Hello world!"
    }
}
```

Check your twitter account and you'll see the new tweet. Record the *id* number and we'll use it next to delete the tweet.

Deleting a tweet

Use the code in *Code to delete a tweet (twitter_delete_tweet.py)* to delete a tweet. Around line 15 is the *id* number. Paste in the value returned above.

Listing 15.51: Code to delete a tweet (twitter_delete_tweet.py)

```
#!/usr/bin/env python
# From: https://github.com/twitterdev/Twitter-API-v2-sample-code/blob/main/Manage-Tweets/delete_tweet.py
from requests_oauthlib import OAuth1Session
import os
import json

# In your terminal please set your environment variables by running the following lines of code.
# export 'API_KEY'='<your_consumer_key>'
# export 'API_SECRET_KEY'='<your_consumer_secret>'

consumer_key = os.environ.get("API_KEY")
consumer_secret = os.environ.get("API_SECRET_KEY")

# Be sure to replace tweet-id-to-delete with the id of the Tweet you wish to delete. The authenticated user must own the list in order to delete
id = "1547963178700533760"

# Get request token
request_token_url = "https://api.twitter.com/oauth/request_token?oauth_callback=oob&x_auth_access_type=write"
oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)

try:
    fetch_response = oauth.fetch_request_token(request_token_url)
except ValueError:
    print(
        "There may have been an issue with the consumer_key or consumer_secret you entered."
    )

resource_owner_key = fetch_response.get("oauth_token")
resource_owner_secret = fetch_response.get("oauth_token_secret")
print("Got OAuth token: %s" % resource_owner_key)

# Get authorization
base_authorization_url = "https://api.twitter.com/oauth/authorize"
authorization_url = oauth.authorization_url(base_authorization_url)
print("Please go here and authorize: %s" % authorization_url)
verifier = input("Paste the PIN here: ")

# Get the access token
access_token_url = "https://api.twitter.com/oauth/access_token"
oauth = OAuth1Session(
    consumer_key,
    client_secret=consumer_secret,
    resource_owner_key=resource_owner_key,
    resource_owner_secret=resource_owner_secret,
    verifier=verifier,
)
oauth_tokens = oauth.fetch_access_token(access_token_url)

access_token = oauth_tokens["oauth_token"]
access_token_secret = oauth_tokens["oauth_token_secret"]

# Make the request
oauth = OAuth1Session(
    consumer_key,
```

(continues on next page)

(continued from previous page)

```
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

# Making the request
response = oauth.delete("https://api.twitter.com/2/tweets/{}".format(id))

if response.status_code != 200:
    raise Exception(
        "Request returned an error: {} {}".format(response.status_code,
↪response.text)
    )

print("Response code: {}".format(response.status_code))

# Saving the response as JSON
json_response = response.json()
print(json_response)
```

twitter_delete_tweet.py

The code in *Tweet when a button is pushed (twitterPushbutton.js)* sends a tweet whenever a button is pushed.

Listing 15.52: Tweet when a button is pushed (twitterPushbutton.js)

```
#!/usr/bin/env node
// From: https://www.npmjs.org/package/node-twitter
// Tweets with attached image media (JPG, PNG or GIF) can be posted
// using the upload API endpoint.
var Twitter = require('node-twitter');
var b = require('bonescript');
var key = require('./twitterKeys');
var gpio = "P9_42";
var count = 0;

b.pinMode(gpio, b.INPUT);
b.attachInterrupt(gpio, sendTweet, b.FALLING);

var twitterRestClient = new Twitter.RestClient(
    key.API_KEY, key.API_SECRET,
    key.TOKEN,   key.TOKEN_SECRET
);

function sendTweet() {
    console.log("Sending...");
    count++;

    twitterRestClient.statusesUpdate(
        {'status': 'Posting tweet ' + count + ' via my BeagleBone Black', },
        function(error, result) {
            if (error) {
                console.log('Error: ' +
                    (error.code ? error.code + ' ' + error.message : error.
↪message));
            }

            if (result) {
                console.log(result);
            }
        }
    );
```

(continues on next page)

(continued from previous page)

```
}

// node-twitter is made available under terms of the BSD 3-Clause License.
// http://www.opensource.org/licenses/BSD-3-Clause
```

`twitterPushbutton.js`

To see many other examples, go to Twitter for Node.js on NPMJS.com.

This opens up many new possibilities. You can read a temperature sensor and tweet its value whenever it changes, or you can turn on an LED whenever a certain hashtag is used. What are you going to tweet?

Wiring the IoT with Node-RED

Problem You want BeagleBone to interact with the Internet, but you want to program it graphically.

Solution Node-RED is a visual tool for wiring the IoT. It makes it easy to turn on a light when a certain hashtag is tweeted, or spin a motor if the forecast is for hot weather.

Starting Node-RED

Node-RED is already installed, to run Node-RED, use the following command to start.

```
bone$ sudo systemctl start nodered
```

Or run the following to have Node-RED start everytime you reboot.

```
bone$ sudo systemctl enable --now nodered
```

Node-RED is listening on part *1880*. Point your browser to http://192.168.7.2:1880, and you will see the screen shown in *The Node-RED web page*.

Building a Node-RED Flow

The example in this recipe builds a Node-RED flow that will toggle an LED whenever a certain hashtag is tweeted. But first, you need to set up the Node-RED flow with the *twitter* node:

- On the Node-RED web page, scroll down until you see the *social* nodes on the left side of the page.
- Drag the *twitter* node to the canvas, as shown in *Node-RED twitter node*.

Authorize Twitter by double-clicking the *twitter* node. You'll see the screen shown in *Node-RED Twitter authorization, step 1*.

Click the pencil button to bring up the dialog box shown in *Node-RED twitter authorization, step 2*.

- Click the "here" link, as shown in *Node-RED twitter authorization, step 2*, and you'll be taken to Twitter to authorize Node-RED.
- Log in to Twitter and click the "Authorize app" button (*Node-RED Twitter site authorization*).
- When you're back to Node-RED, click the Add button, add your Twitter credentials, enter the hashtags to respond to (*Node-RED adding the #BeagleBone hashtag*), and then click the Ok button.
- Go back to the left panel, scroll up to the top, and then drag the *debug* node to the canva- (*debug* is in the *output* section.)
- Connect the two nodes by clicking and dragging (*Node-RED Twitter adding debug node and connecting*).
- In the right panel, in the upper-right corner, click the "debug" tab.

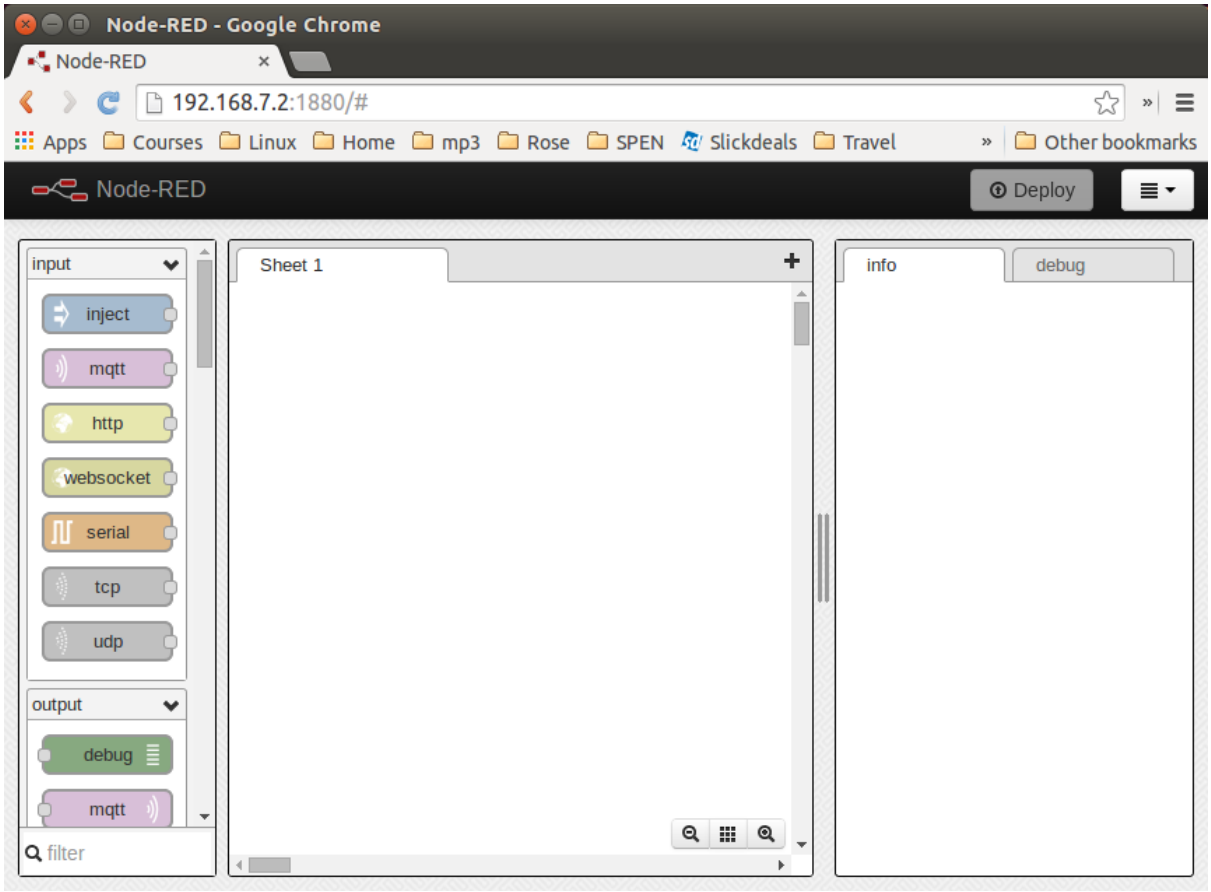


Fig. 15.57: The Node-RED web page

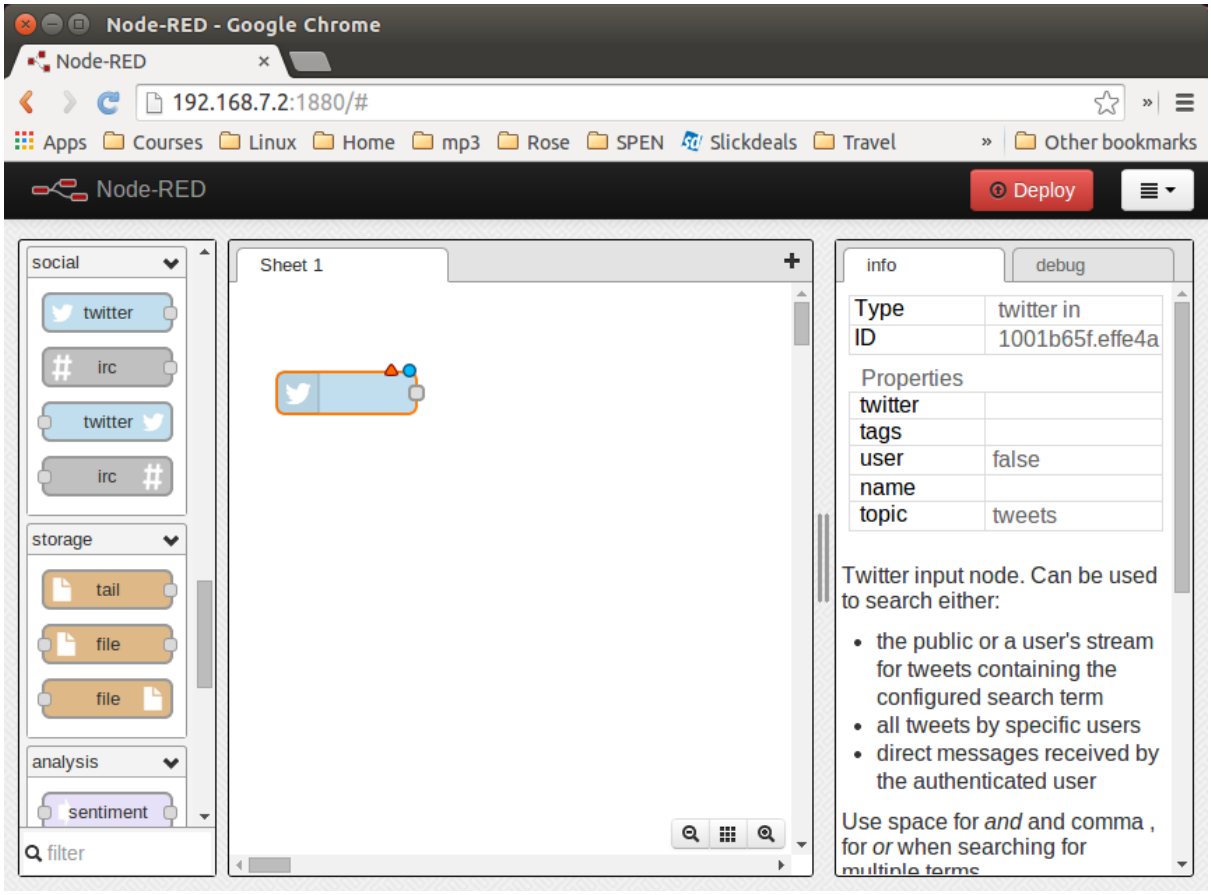


Fig. 15.58: Node-RED twitter node

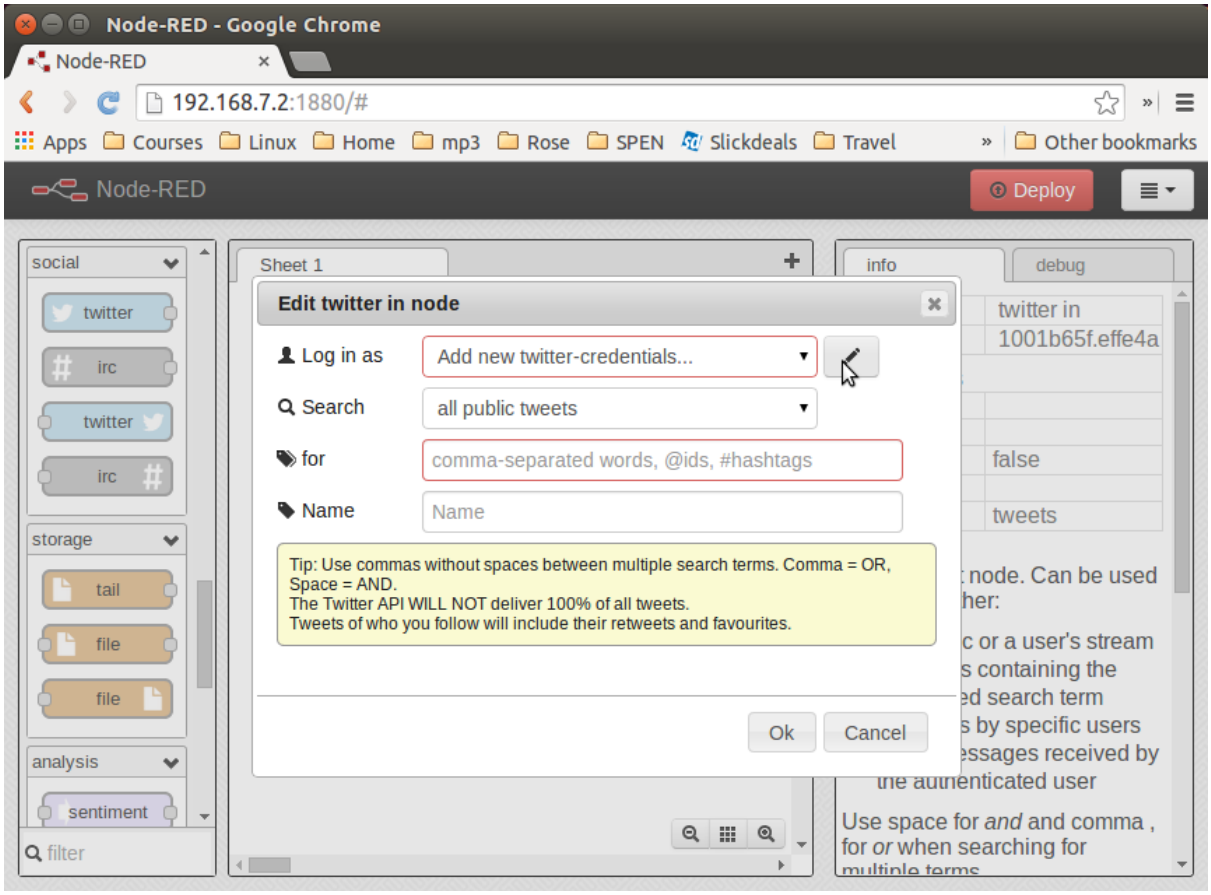


Fig. 15.59: Node-RED Twitter authorization, step 1

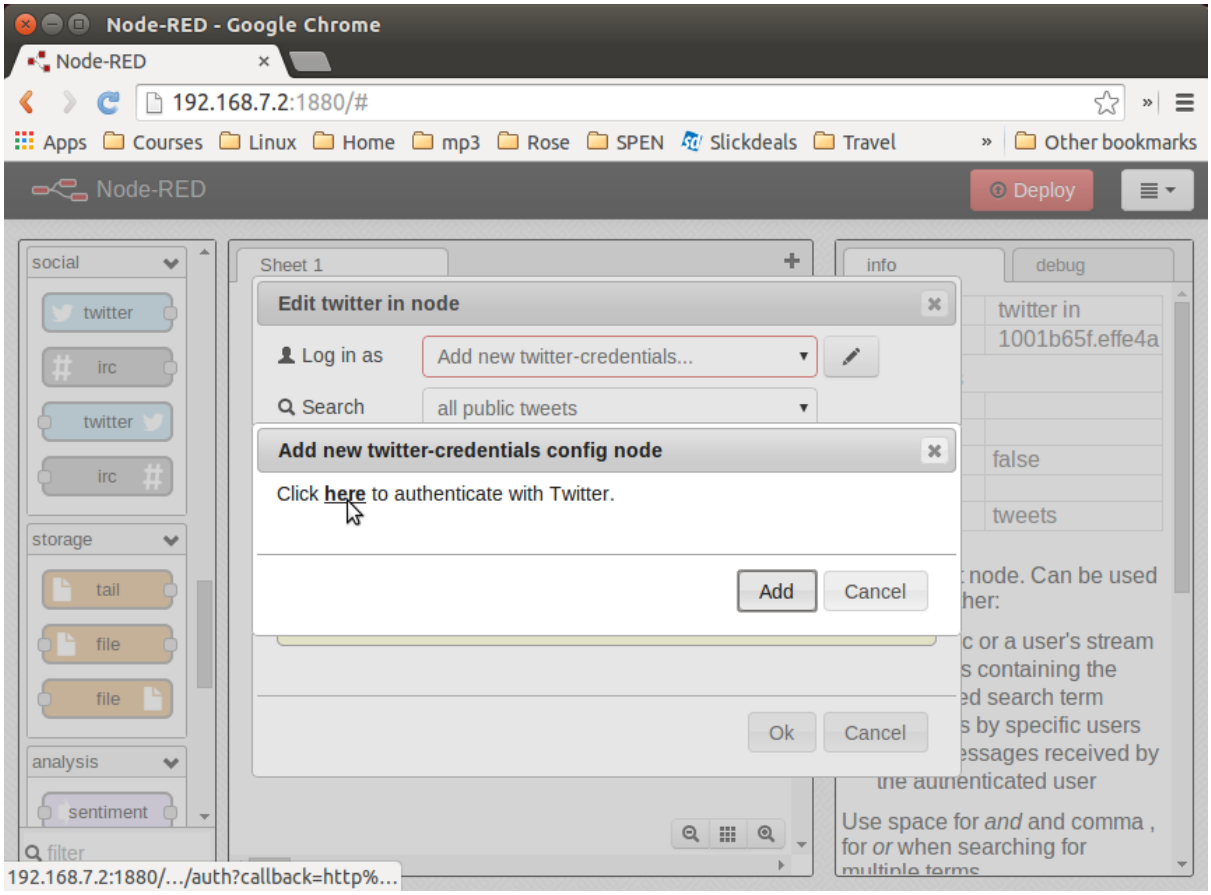


Fig. 15.60: Node-RED twitter authorization, step 2

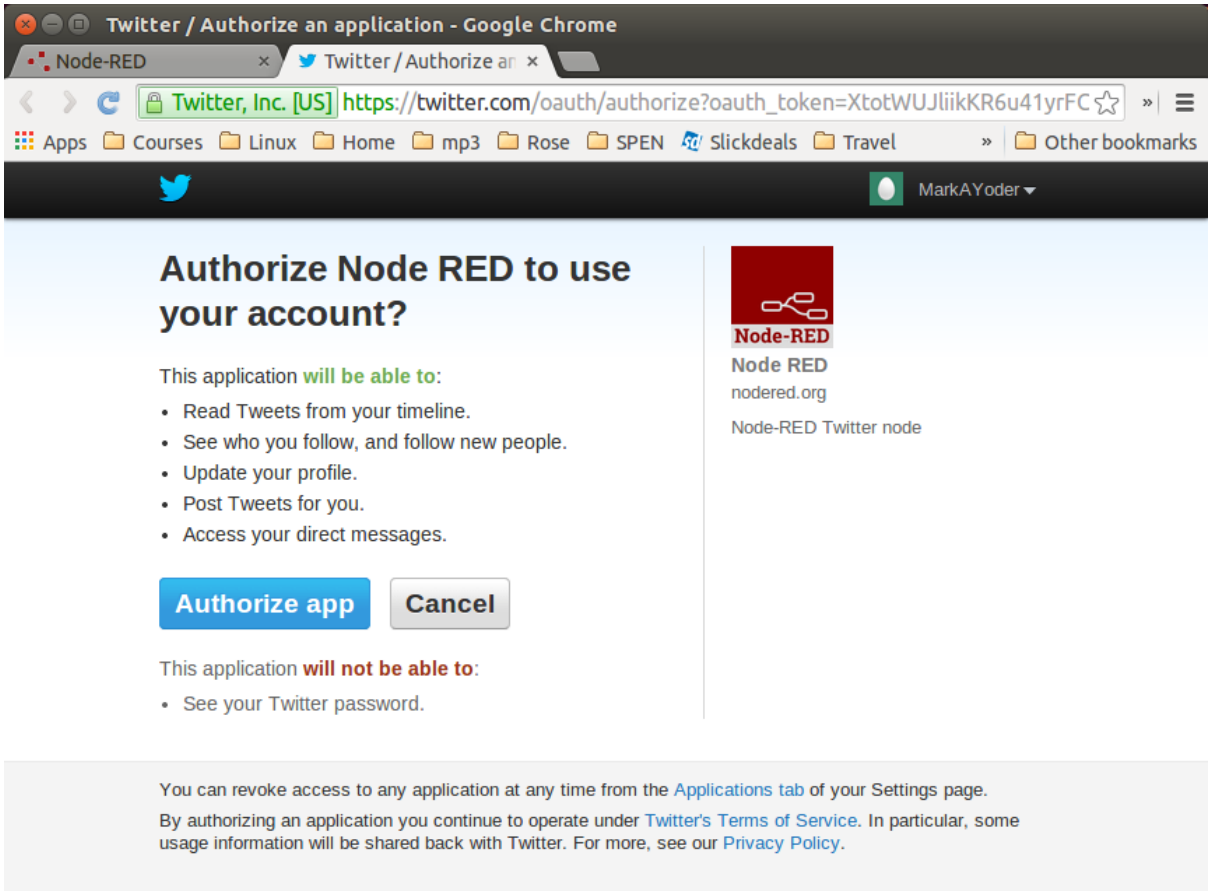


Fig. 15.61: Node-RED Twitter site authorization

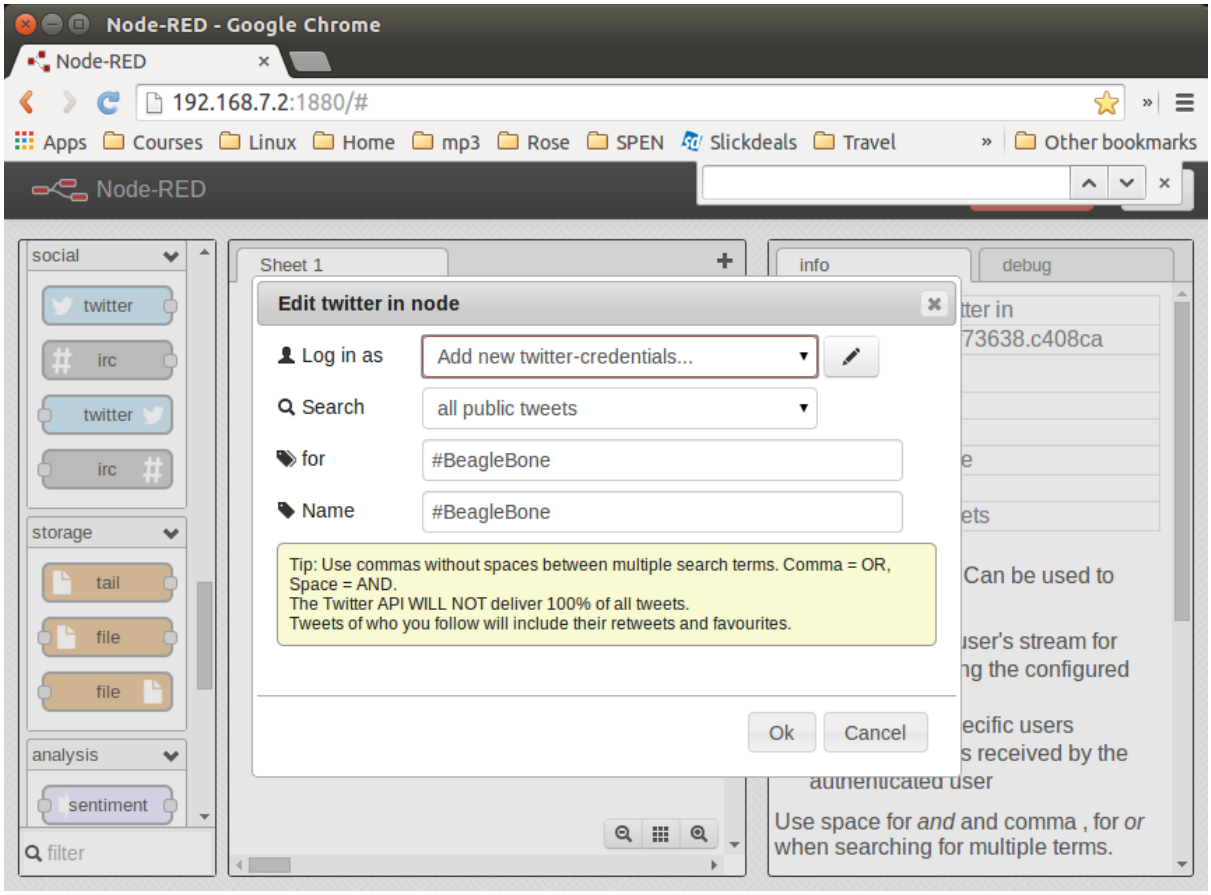


Fig. 15.62: Node-RED adding the #BeagleBone hashtag

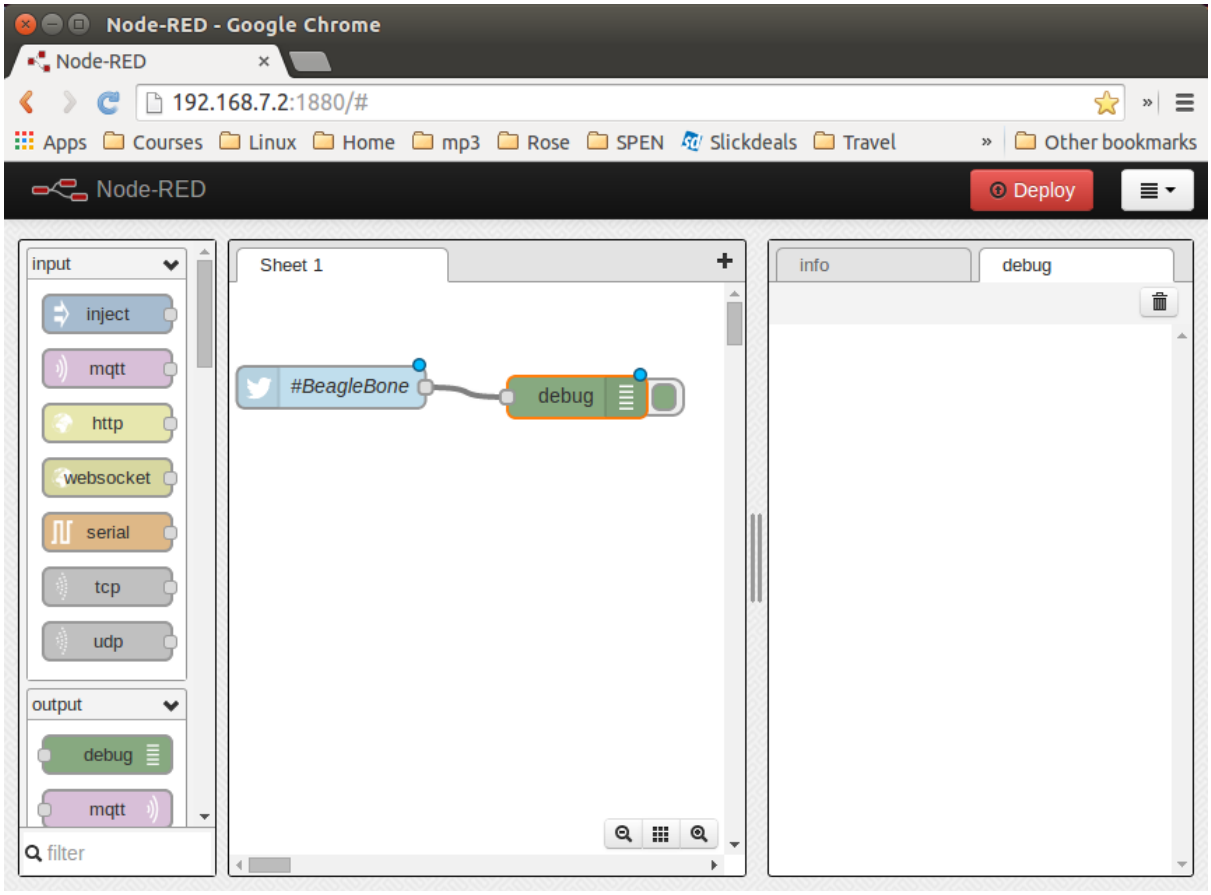


Fig. 15.63: Node-RED Twitter adding *debug* node and connecting

- Finally, click the Deploy button above the “debug” tab.

Your Node-RED flow is now running on the Bone. Test it by going to Twitter and tweeting something with the hashtag *#BeagleBone*. Your Bone is now responding to events happening out in the world.

Adding an LED Toggle

Now, we’re ready to add the LED toggle:

- Wire up an LED as shown in *Toggling an External LED*. Mine is wired to *P9_14*.
- Scroll to the bottom of the left panel and drag the *bbb-discrete-out* node (second from the bottom of the *bbb* nodes) to the canvas and wire it (*Node-RED adding bbb-discrete-out node*).

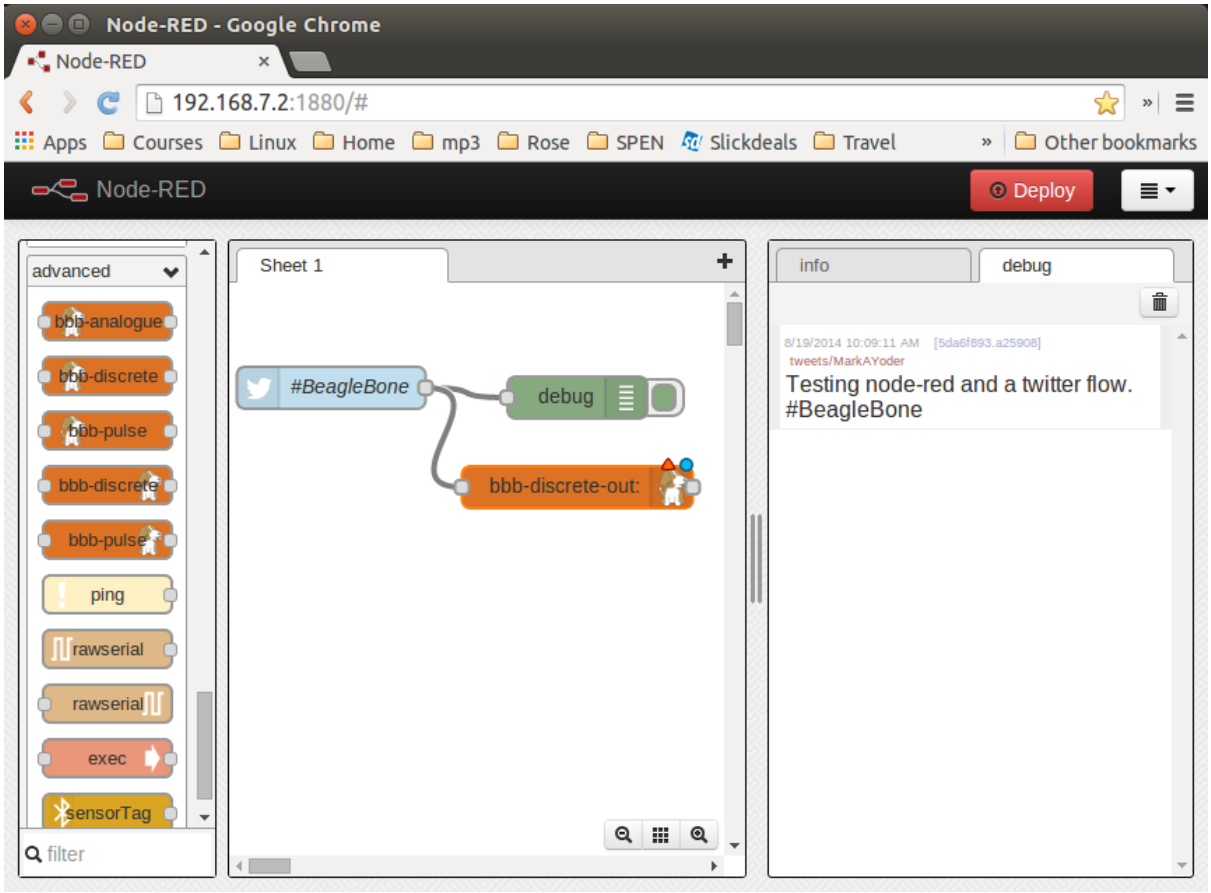


Fig. 15.64: Node-RED adding bbb-discrete-out node

Double-click the node, select your GPIO pin and “Toggle state,” and then set “Startup as” to *1* (*Node-RED adding bbb-discrete-out configuration*).

Click Ok and then Deploy.

Test again. The LED will toggle every time the hashtag *#BeagleBone* is tweeted. With a little more exploring, you should be able to have your Bone ringing a bell or spinning a motor in response to tweets.

Communicating over a Serial Connection to an Arduino or LaunchPad

Problem You would like your Bone to talk to an Arduino or LaunchPad.

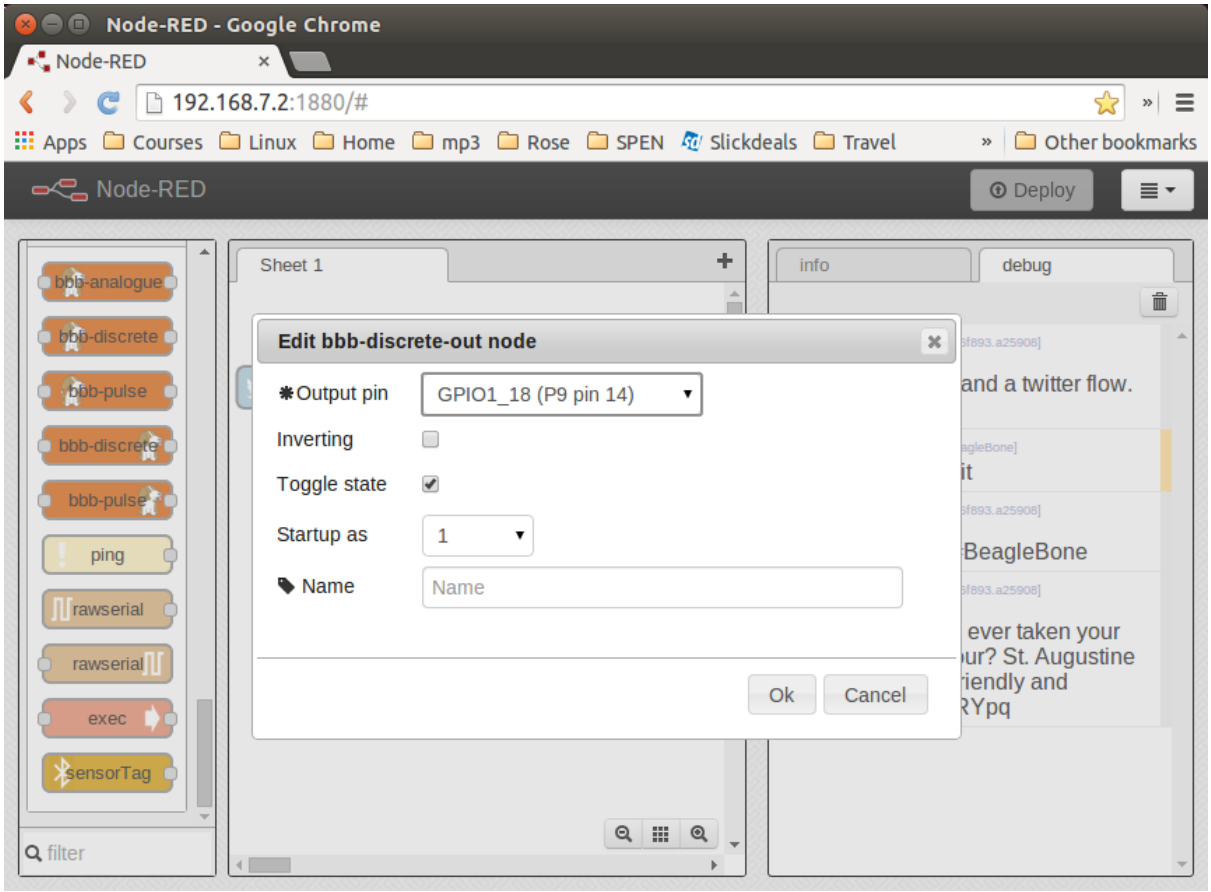


Fig. 15.65: Node-RED adding bbb-discrete-out configuration

Solution The common serial port (also known as a UART) is the simplest way to talk between the two. Wire it up as shown in *Wiring a LaunchPad to a Bone via the common serial port*.

Warning: BeagleBone Black runs at 3.3 V. When wiring other devices to it, ensure that they are also 3.3 V. The LaunchPad I'm using is 3.3 V, but many Arduinos are 5.0 V and thus won't work. Or worse, they might damage your Bone.

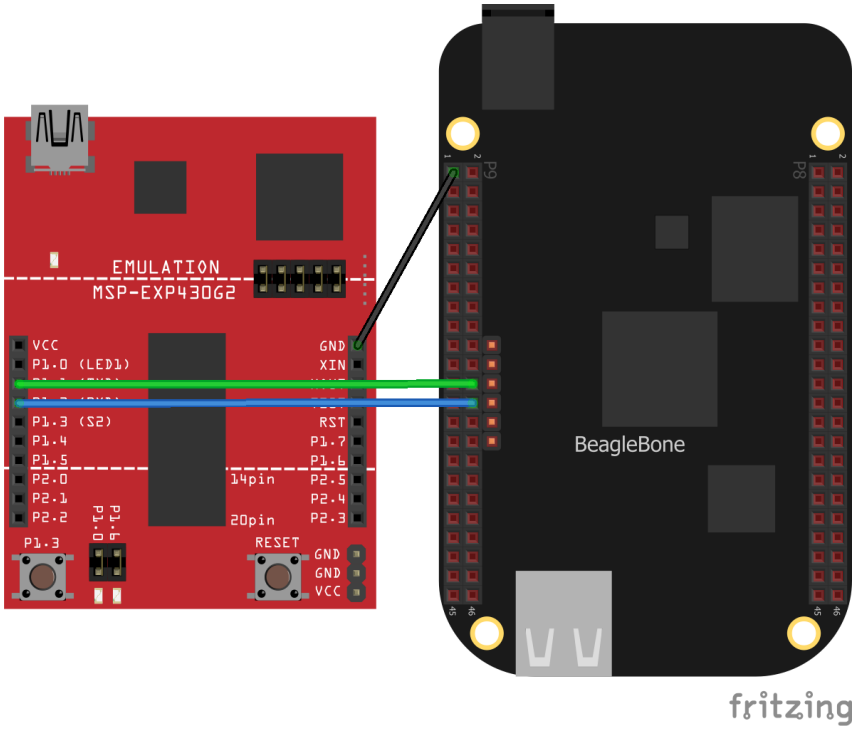


Fig. 15.66: Wiring a LaunchPad to a Bone via the common serial port

Add the code (or `sketch`, as it's called in Arduino-speak) in *LaunchPad code for communicating via the UART (launchPad.ino)* to a file called `launchPad.ino` and run it on your LaunchPad.

Listing 15.53: LaunchPad code for communicating via the UART (launchPad.ino)

```
/*
  Tests connection to a BeagleBone
  Mark A. Yoder
  Waits for input on Serial Port
  g - Green toggle
  r - Red toggle
*/
char inChar = 0; // incoming serial byte
int red = 0;
int green = 0;

void setup()
{
  // initialize the digital pin as an output.
  pinMode(RED_LED, OUTPUT);            // 
  pinMode(GREEN_LED, OUTPUT);
  // start serial port at 9600 bps:
  Serial.begin(9600);                  // 
  Serial.print("Command (r, g): ");    // 

```

(continues on next page)

(continued from previous page)

```
  digitalWrite(GREEN_LED, green);      // ②
  digitalWrite(  RED_LED, red);
}

void loop()
{
  if(Serial.available() > 0 ) {        // ③
    inChar = Serial.read();
    switch(inChar) {                   // ④
      case 'g':
        green = ~green;
        digitalWrite(GREEN_LED, green);
        Serial.println("Green");
        break;
      case 'r':
        red = ~red;
        digitalWrite(RED_LED, red);
        Serial.println("Red");
        break;
    }
    Serial.print("Command (r, g): ");
  }
}

```

launchPad.ino

① Set the mode for the built-in red and green LEDs.

② Start the serial port at 9600 baud.

③ Prompt the user, which in this case is the Bone.

④ Set the LEDs to the current values of the *red* and *green* variables.

⑤ Wait for characters to arrive on the serial port.

⑥ After the characters are received, read it and respond to it.

On the Bone, add the script in *Code for communicating via the UART (launchPad.js)* to a file called *launchPad.js* and run it.

Listing 15.54: Code for communicating via the UART (launchPad.js)

```
#!/usr/bin/env node
// Need to add exports.serialParsers = m.module.parsers;
// to /usr/local/lib/node_modules/bonescript/serial.js
var b = require('bonescript');

var port = '/dev/ttyO1';                    // ①
var options = {
    baudrate: 9600,                         // ②
    parser: b.serialParsers.readline("\n")  // ③
};

b.serialOpen(port, options, onSerial);      // ④

function onSerial(x) {                      // ⑤
    console.log(x.event);
    if (x.err) {
        console.log('***ERROR*** ' + JSON.stringify(x));
    }
    if (x.event == 'open') {
        console.log('***OPENED***');
```

(continues on next page)

(continued from previous page)

```
        setInterval(sendCommand, 1000);      // ⑥
    }
    if (x.event == 'data') {
        console.log(String(x.data));
    }
}

var command = ['r', 'g'];                    // ⑦
var commIdx = 1;

function sendCommand() {
    // console.log('Command: ' + command[commIdx]);
    b.serialWrite(port, command[commIdx++]); // ⑧
    if(commIdx >= command.length) {          // ⑨
        commIdx = 0;
    }
}
```

launchPad.js

① Select which serial port to use. *Table of UART outputs* sows what's available. We've wired *P9_24* and *P9_26*, so we are using serial port */dev/ttyO1*. (Note that's the letter O and not the number zero.)

② Set the baudrate to 9600, which matches the setting on the LaunchPad.

③ Read one line at a time up to the newline character (*n*).

④ Open the serial port and call *onSerial()* whenever there is data available.

⑤ Determine what event has happened on the serial port and respond to it.

⑥ If the serial port has been *opened*, start calling *sendCommand()* every 1000 ms.

⑦ These are the two commands to send.

⑧ Write the character out to the serial port and to the LaunchPad.

⑨ Move to the next command.

Discussion When you run the script in *Code for communicating via the UART (launchPad.js)*, the Bone opens up the serial port and every second sends a new command, either *r* or *g*. The LaunchPad waits for the command, when it arrives, responds by toggling the corresponding LED.

15.1.7 The Kernel

The kernel is the heart of the Linux operating system. It's the software that takes the low-level requests, such as reading or writing files, or reading and writing general-purpose input/output (GPIO) pins, and maps them to the hardware. When you install a new version of the OS (*Verifying You Have the Latest Version of the OS on Your Bone*), you get a certain version of the kernel.

You usually won't need to mess with the kernel, but sometimes you might want to try something new that requires a different kernel. This chapter shows how to switch kernels. The nice thing is you can have multiple kernels on your system at the same time and select from among them which to boot up.

Updating the Kernel

Problem You have an out-of-date kernel and want to make it current.

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
UART1_RTSn	19	20	UART1_CTSn
UART2_TXD	21	22	UART2_RXD
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	GPIO_113
GPIO_111	29	30	GPIO_112
GPIO_110	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	UART3_TXD
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
UART5_CTSn+	31	32	UART5_RTSn
UART4_RTSn	33	34	UART3_RTSn
UART4_CTSn	35	36	UART3_CTSn
UARR5_TXD+	37	38	UART5_RXD+
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

Fig. 15.67: Table of UART outputs

Solution Use the following command to determine which kernel you are running:

```
bone$ uname -a
Linux beaglebone 5.10.168-ti-r62 #1bullseye SMP PREEMPT Tue May 23 20:15:00␣
↪UTC 2023 armv7l GNU/Linux
GNU/Linux
```

The *5.10.168-ti-r62* string is the kernel version.

To update to the current kernel, ensure that your Bone is on the Internet (*Sharing the Host's Internet Connection over USB* or *Establishing an Ethernet-Based Internet Connection*) and then run the following commands:

```
bone$ apt-cache pkgnames | grep linux-image | sort | less
...
linux-image-5.10.162-ti-r59
linux-image-5.10.162-ti-rt-r56
linux-image-5.10.162-ti-rt-r57
linux-image-5.10.162-ti-rt-r58
linux-image-5.10.162-ti-rt-r59
linux-image-5.10.168-armv7-lpae-x71
linux-image-5.10.168-armv7-rt-x71
linux-image-5.10.168-armv7-x71
linux-image-5.10.168-bone71
linux-image-5.10.168-bone-rt-r71
linux-image-5.10.168-ti-r60
linux-image-5.10.168-ti-r61
linux-image-5.10.168-ti-r62
linux-image-5.10.168-ti-rt-r60
linux-image-5.10.168-ti-rt-r61
linux-image-5.10.168-ti-rt-r62
...

bone$ sudo apt install linux-image-5.10.162-ti-rt-r59
```

(continues on next page)

(continued from previous page)

```
bone$ sudo reboot

bone$ uname -a
Linux beaglebone 5.10.162-ti-rt-r59 #1 SMP PREEMPT Wed Nov 19 21:11:08 UTC␣
↪2014 armv7l
GNU/Linux
```

The first command lists the versions of the kernel that are available. The second command installs one. After you have rebooted, the new kernel will be running.

If the current kernel is doing its job adequately, you probably don't need to update, but sometimes a new software package requires a more up-to-date kernel. Fortunately, precompiled kernels are available and ready to download.

Seeing which kernels are installed You can have multiple kernels install at the same time. T hey are saved in **/boot**

```
bone$ cd /boot
bone$ ls
config-5.10.168-ti-r62      initrd.img-5.10.168-ti-r63  uboot              ␣
↪    vmlinuz-5.10.168-ti-r63
config-5.10.168-ti-r63      SOC.sh                      uEnv.txt
dtbs                        System.map-5.10.168-ti-r62  uEnv.txt.orig
initrd.img-5.10.168-ti-r62  System.map-5.10.168-ti-r63  vmlinuz-5.10.168-ti-
↪r62
```

Here I have two kernel versions installed.

Bone

Play

On the Bone (Not the Play) the file **uEnv.txt** tells which kernel to use on the next reboot. Here are the first few lines:

```
Line
1 #Docs: http://elinux.org/Beagleboard:U-boot_partitioning_layout_2.0
2
3 # uname_r=4.14.108-ti-r137
4 uname_r=4.19.94-ti-r50
5 # uname_r=5.4.52-ti-r17
6 #uuid=
```

Lines 3-5 list the various kernels, and the uncommented one on line 4 is the one that will be used next time. You will have to add your own uname's. Get the names from the files in /boot. Be careful, if you mistype the name your Bone won't boot.

On the Play you can see which version of the kernel will boot next by:

```
play$ cat /boot/firmware/kversion
5.10.168-ti-arm64-r106
```

If you want to change the version run:

```
bone$ sudo apt install linux-image-5.10.168-ti-arm64-r105 --reinstall
```

Building and Installing Kernel Modules

Problem You need to use a peripheral for which there currently is no driver, or you need to improve the performance of an interface previously handled in user space.

Solution The solution is to run in kernel space by building a kernel module. There are entire books on writing Linux Device Drivers. This recipe assumes that the driver has already been written and shows how to compile and install it. After you've followed the steps for this simple module, you will be able to apply them to any other module.

For our example module, add the code in *Simple Kernel Module (hello.c)* to a file called `hello.c`.

Listing 15.55: Simple Kernel Module (hello.c)

```
#include <linux/module.h>       /* Needed by all modules */
#include <linux/kernel.h>       /* Needed for KERN_INFO */
#include <linux/init.h>         /* Needed for the macros */

static int __init hello_start(void)
{
    printk(KERN_INFO "Loading hello module...\n");
    printk(KERN_INFO "Hello, World!\n");
    return 0;
}

static void __exit hello_end(void)
{
    printk(KERN_INFO "Goodbye Boris\n");
}

module_init(hello_start);
module_exit(hello_end);

MODULE_AUTHOR("Boris Houndleroy");
MODULE_DESCRIPTION("Hello World Example");
MODULE_LICENSE("GPL");
```

`hello.c`

When compiling on the Bone, all you need to do is load the Kernel Headers for the version of the kernel you're running:

```
bone$ sudo apt install linux-headers-`uname -r`
```

Note: The quotes around `uname -r` are backtick characters. On a United States keyboard, the backtick key is to the left of the 1 key.

This took a little more than three minutes on my Bone. The `uname -r` part of the command looks up what version of the kernel you are running and loads the headers for it.

Note: If you don't have a network connection you can get the headers from the running kernel with the following.

```
sudo modprobe kheaders
rm -rf $HOME/headers
mkdir -p $HOME/headers
tar -xvf /sys/kernel/kheaders.tar.xz -C $HOME/headers > /dev/null
cd my-kernel-module
make -C $HOME/headers M=$(pwd) modules
sudo rmmod kheaders
```

The `modprobe kheaders` makes the `/sys/kernel/kheaders.tar.xz` appear.

Next, add the code in *Simple Kernel Module (Makefile)* to a file called `Makefile`.

Listing 15.56: Simple Kernel Module (`Makefile`)

```
obj-m := hello.o
KDIR  := /lib/modules/$(shell uname -r)/build

all:
<TAB>make -C $(KDIR) M=$$PWD

clean:
<TAB>rm hello.mod.c hello.o modules.order hello.mod.o Module.symvers
```

`Makefile.display`

Note: Replace the two instances of *<TAB>* with a tab character (the key left of the Q key on a United States keyboard). The tab characters are very important to makefiles and must appear as shown.

Now, compile the kernel module by using the *make* command:

```
bone$ make
make -C /lib/modules/5.10.168-ti-r62/build M=$PWD
make[1]: Entering directory '/usr/src/linux-headers-5.10.168-ti-r62'
CC [M]  /home/debian/docs.beagleboard.io/books/beaglebone-cookbook/code/
 ↪07kernel/hello.o
MODPOST /home/debian/docs.beagleboard.io/books/beaglebone-cookbook/code/
 ↪07kernel/Module.symvers
CC [M]  /home/debian/docs.beagleboard.io/books/beaglebone-cookbook/code/
 ↪07kernel/hello.mod.o
LD [M]  /home/debian/host/BeagleBoard/docs.beagleboard.io/books/beaglebone-
 ↪cookbook/code/07kernel/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.10.168-ti-r62'

bone$ ls
Makefile        hello.c   hello.mod.c  hello.o
Module.symvers  hello.ko  hello.mod.o  modules.order
```

Notice that several files have been created. `hello.ko` is the one you want. Try a couple of commands with it:

```
bone$ modinfo hello.ko
filename:       /home/debian/host/BeagleBoard/docs.beagleboard.io/books/
 ↪beaglebone-cookbook/code/07kernel/hello.ko
license:        GPL
description:    Hello World Example
author:         Boris Houndleroy
depends:
name:           hello
vermagic:       5.10.168-ti-r62 SMP preempt mod_unload modversions ARMv7 p2v8

bone$ sudo insmod hello.ko
bone$ dmesg | tail -4
[  377.944777] lm75 1-004a: hwmon1: sensor 'tmp101'
[  377.944976] i2c i2c-1: new_device: Instantiated device tmp101 at 0x4a
[85819.772666] Loading hello module...
[85819.772687] Hello, World!
```

The first command displays information about the module. The *insmod* command inserts the module into the running kernel. If all goes well, nothing is displayed, but the module does print something in the kernel log. The *dmesg* command displays the messages in the log, and the *tail -4* command shows the last four messages. The last two messages are from the module. It worked!

Compiling the Kernel

Problem You need to download, patch, and compile the kernel from its source code.

Solution This is easier than it sounds, thanks to some very powerful scripts.

Warning: Be sure to run this recipe on your host computer. The Bone has enough computational power to compile a module or two, but compiling the entire kernel takes lots of time and resources.

Downloading and Compiling the Kernel

To download and compile the kernel, follow these steps:

```
host$ git clone https://git.beagleboard.org/RobertCNelson/ti-linux-kernel-
↪dev # ①
host$ cd ti-linux-kernel-dev
host$ git checkout ti-linux-5.10.y # ②
host$ ./build_deb.sh # ③
```

Note: If you are using a 64 bit Bone, **git checkout ti-linux-arm64-5.10.y**

① The first command clones a repository with the tools to build the kernel for the Bone.

② When you know which kernel to try, use *git checkout* to check it out. This command checks out branch *ti-linux-5.10.y*.

③ *build_deb.sh* is the master builder. If needed, it will download the cross compilers needed to compile the kernel (gcc is the current cross compiler). If there is a kernel at `~/linux-dev`, it will use it; otherwise, it will download a copy to `ti-linux-kernel-dev/ignore/linux-src`. It will then patch the kernel so that it will run on the Bone.

Note: *build_deb.sh* may ask you to install additional files. Just run **sudo apt install *files*** to install them.

After the kernel is patched, you'll see a screen similar to *Kernel configuration menu*, on which you can configure the kernel.

You can use the arrow keys to navigate. No changes need to be made, so you can just press the right arrow and Enter to start the kernel compiling. The entire process took about 25 minutes on my 8-core host.

The `ti-linux-kernel-dev/KERNEL` directory contains the source code for the kernel. The `ti-linux-kernel-dev/deploy` directory contains the compiled kernel and the files needed to run it.

Installing the Kernel on the Bone

The **./build_deb.sh** script creates a single .deb file that contains all the files needed for the new kernel. You find it here:

```
host$ cd ti-linux-kernel-dev/deploy
host$ ls -sh
total 40M
7.7M linux-headers-5.10.168-ti-r62_1xross_armhf.deb  8.0K linux-upstream_
↪1xross_armhf.buildinfo
```

(continues on next page)

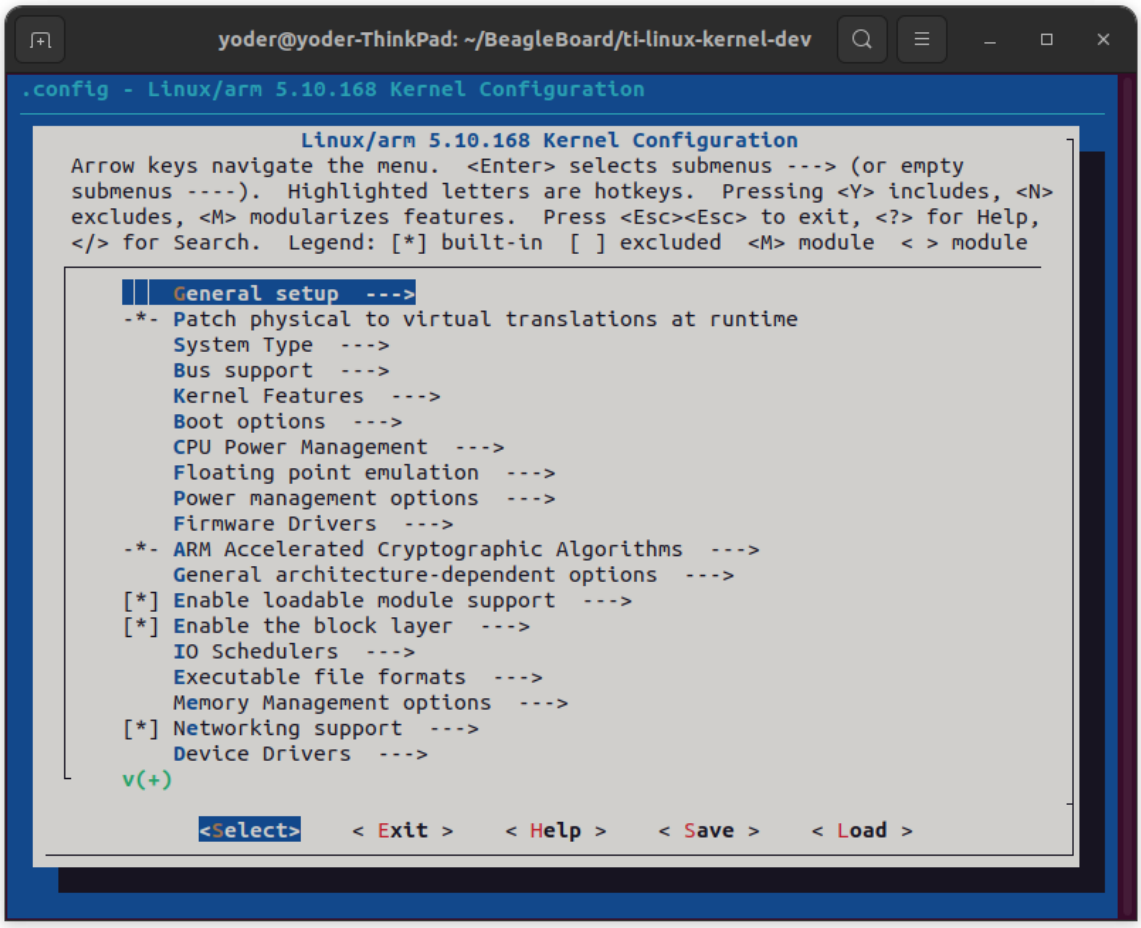


Fig. 15.68: Kernel configuration menu

(continued from previous page)

```
 33M linux-image-5.10.168-ti-r62_1xross_armhf.deb    4.0K linux-upstream_
↪1xross_armhf.changes
1.1M linux-libc-dev_1xross_armhf.deb
```

The **linux-image-** file is the one we want. It contains over 3000 files.

```
host$ dpkg -c linux-image-5.10.168-ti-r62_1xross_armhf.deb | wc
   3251   19506  379250
```

The **dpkg** command lists all the files in the .deb file and the wc counts all the lines in the output. You can see those files with:

```
host$ dpkg -c linux-image-5.10.168-ti-r62_1xross_armhf.deb | less
drwxr-xr-x root/root         0 2023-06-12 12:57 ./
drwxr-xr-x root/root         0 2023-06-12 12:57 ./boot/
-rw-r--r-- root/root   4763113 2023-06-12 12:57 ./boot/System.map-5.10.168-
↪ti-r62
-rw-r--r-- root/root    191331 2023-06-12 12:57 ./boot/config-5.10.168-ti-r62
drwxr-xr-x root/root         0 2023-06-12 12:57 ./boot/dtbs/
drwxr-xr-x root/root         0 2023-06-12 12:57 ./boot/dtbs/5.10.168-ti-r62/
-rwxr-xr-x root/root     90644 2023-06-12 12:57 ./boot/dtbs/5.10.168-ti-r62/
↪am335x-baltos-ir2110.dtb
-rwxr-xr-x root/root     91362 2023-06-12 12:57 ./boot/dtbs/5.10.168-ti-r62/
↪am335x-baltos-ir3220.dtb
-rwxr-xr-x root/root     91633 2023-06-12 12:57 ./boot/dtbs/5.10.168-ti-r62/
↪am335x-baltos-ir5221.dtb
-rwxr-xr-x root/root     88684 2023-06-12 12:57 ./boot/dtbs/5.10.168-ti-r62/
↪am335x-base0033.dtb
```

You can see it's putting things in the **/boot** directory.

Note: You can also look into the other two .deb files and see what they install.

Move the **linux-image-** file to your Bone.

```
host$ scp linux-image-5.10.168-ti-r62_1xross_armhf.deb bone:.
```

You might have to use debian@192.168.7.2 for bone if you haven't set everything up.

Now ssh to the bone.

```
host$ ssh bone
bone$ ls -sh
bin  exercises linux-image-5.10.168-ti-r62_1xross_armhf.deb
```

Now install it.

```
bone$ sudo dpkg --install linux-image-5.10.168-ti-r62_1xross_armhf.deb
```

Wait a while. (Mine took almore 2 minutes.) Once done check /boot.

```
bone$ ls -sh /boot
total 40M
160K config-4.19.94-ti-r50        4.0K SOC.sh                      4.0K uEnv.
↪txt.orig
180K config-5.10.168-ti-r62       3.5M System.map-4.19.94-ti-r50   9.7M␣
↪vmlinuz-4.19.94-ti-r50
4.0K dtbs                         4.1M System.map-5.10.168-ti-r62  8.6M␣
↪vmlinuz-5.10.168-ti-r62
6.4M initrd.img-4.19.94-ti-r50    4.0K uboot
6.8M initrd.img-5.10.168-ti-r62   4.0K uEnv.txt
```

You see the new kernel files along with the old files. Check uEnv.txt.

```
bone$ head /boot/uEnv.txt
#Docs: http://elinux.org/Beagleboard:U-boot_partitioning_layout_2.0
# uname_r=4.19.94-ti-r50
uname_r=5.10.168-ti-r62
```

I added the commented out uname_r line to make it easy to switch between versions of the kernel.

Reboot and test out the new kernel.

```
bone$ sudo reboot
```

Installin a Cross Compiler

Problem You want to compile on your host computer and run on the Beagle.

Solution Run the following:

32-bit

64-bit

```
host$ sudo apt install gcc-arm-linux-gnueabihf
```

```
host$ sudo apt install gcc-aarch64-linux-gnu
```

Note: From now on use **arm** if you are using a 32-bit machine and **aarch64** if you are using a 64-bit machine.

This installs a cross compiler, but you need to set up a couple of things so that it can be found. At the command prompt, enter **arm-<TAB><TAB>** to see what was installed.

```
host$ arm-<TAB><TAB>
arm-linux-gnueabihf-addr2line      arm-linux-gnueabihf-gcc-nm         arm-
↪linux-gnueabihf-ld.bfd
arm-linux-gnueabihf-ar             arm-linux-gnueabihf-gcc-nm-11      arm-
↪linux-gnueabihf-ld.gold
arm-linux-gnueabihf-as             arm-linux-gnueabihf-gcc-ranlib     arm-
↪linux-gnueabihf-lto-dump-11
arm-linux-gnueabihf-c++filt        arm-linux-gnueabihf-gcc-ranlib-11  arm-
↪linux-gnueabihf-nm
arm-linux-gnueabihf-cpp            arm-linux-gnueabihf-gcov           arm-
↪linux-gnueabihf-objcopy
arm-linux-gnueabihf-cpp-11         arm-linux-gnueabihf-gcov-11        arm-
↪linux-gnueabihf-objdump
arm-linux-gnueabihf-dwp            arm-linux-gnueabihf-gcov-dump      arm-
↪linux-gnueabihf-ranlib
arm-linux-gnueabihf-elfedit        arm-linux-gnueabihf-gcov-dump-11   arm-
↪linux-gnueabihf-readelf
arm-linux-gnueabihf-gcc            arm-linux-gnueabihf-gcov-tool      arm-
↪linux-gnueabihf-size
arm-linux-gnueabihf-gcc-11         arm-linux-gnueabihf-gcov-tool-11   arm-
↪linux-gnueabihf-strings
arm-linux-gnueabihf-gcc-ar         arm-linux-gnueabihf-gprof          arm-
↪linux-gnueabihf-strip
arm-linux-gnueabihf-gcc-ar-11      arm-linux-gnueabihf-ld
```

What you see are all the cross-development tools.

Setting Up Variables

Now, set up a couple of variables to know which compiler you are using:

```
host$ export ARCH=arm
host$ export CROSS_COMPILE=arm-linux-gnueabihf-
```

These lines set up the standard environmental variables so that you can determine which cross-development tools to use. Test the cross compiler by adding *Simple helloWorld.c to test cross compiling (helloWorld.c)* to a file named _helloWorld.c_.

Listing 15.57: Simple helloWorld.c to test cross compiling (helloWorld.c)

```
#include <stdio.h>

int main(int argc, char **argv) {
  printf("Hello, World! \n");
}
```

helloWorld.c

You can then cross-compile by using the following commands:

```
host$ ${CROSS_COMPILE}gcc helloWorld.c
host$ file a.out
a.out: ELF 32-bit LSB executable, ARM, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.31,
BuildID[sha1]=0x10182364352b9f3cb15d1aa61395aeede11a52ad, not stripped
```

The *file* command shows that *a.out* was compiled for an ARM processor.

Applying Patches

Problem You have a patch file that you need to apply to the kernel.

Solution *Simple kernel patch file (hello.patch)* shows a patch file that you can use on the kernel.

Listing 15.58: Simple kernel patch file (hello.patch)

```
From eaf4f7ea7d540bc8bb57283a8f68321ddb4401f4 Mon Sep 17 00:00:00 2001
From: Jason Kridner <jdk@ti.com>
Date: Tue, 12 Feb 2013 02:18:03 +0000
Subject: [PATCH] hello: example kernel modules

---
 hello/Makefile   |    7 +++++++
 hello/hello.c    |   18 ++++++++++++++++++
 2 files changed, 25 insertions(+), 0 deletions(-)
 create mode 100644 hello/Makefile
 create mode 100644 hello/hello.c

diff --git a/hello/Makefile b/hello/Makefile
new file mode 100644
index 0000000..4b23da7
--- /dev/null
+++ b/hello/Makefile
@@ -0,0 +1,7 @@
+obj-m := hello.o
+
+PWD   := $(shell pwd)
```

(continues on next page)

(continued from previous page)

```
+KDIR  := ${PWD}/..
+
+default:
+	make -C $(KDIR) SUBDIRS=$(PWD) modules
diff --git a/hello/hello.c b/hello/hello.c
new file mode 100644
index 0000000..157d490
--- /dev/null
+++ b/hello/hello.c
@@ -0,0 +1,22 @@
+#include <linux/module.h>	/* Needed by all modules */
+#include <linux/kernel.h>	/* Needed for KERN_INFO */
+#include <linux/init.h>		/* Needed for the macros */
+
+static int __init hello_start(void)
+{
+    printk(KERN_INFO "Loading hello module...\n");
+    printk(KERN_INFO "Hello, World!\n");
+    return 0;
+}
+
+static void __exit hello_end(void)
+{
+    printk(KERN_INFO "Goodbye Boris\n");
+}
+
+module_init(hello_start);
+module_exit(hello_end);
+
+MODULE_AUTHOR("Boris Houndleroy");
+MODULE_DESCRIPTION("Hello World Example");
+MODULE_LICENSE("GPL");
```

`hello.patch`

Here's how to use it:

- Install the kernel sources (*Compiling the Kernel*).
- Change to the kernel directory (+cd ti-linux-kernel-dev/KERNEL+).
- Add *Simple kernel patch file (hello.patch)* to a file named `hello.patch` in the `ti-linux-kernel-dev/KERNEL` directory.
- Run the following commands:

```
host$ cd ti-linux-kernel-dev/KERNEL
host$ patch -p1 &lt; hello.patch
patching file hello/Makefile
patching file hello/hello.c
```

The output of the *patch* command apprises you of what it's doing. Look in the `hello` directory to see what was created:

```
host$ cd hello
host$ ls
hello.c  Makefile
```

Building and Installing Kernel Modules shows how to build and install a module, and *Creating Your Own Patch File* shows how to create your own patch file.

Creating Your Own Patch File

Problem You made a few changes to the kernel, and you want to share them with your friends.

Solution Create a patch file that contains just the changes you have made. Before making your changes, check out a new branch:

```
host$ cd ti-linux-kernel-dev/KERNEL
host$ git status
# On branch master
nothing to commit (working directory clean)
```

Good, so far no changes have been made. Now, create a new branch:

```
host$ git checkout -b hello1
host$ git status
# On branch hello1
nothing to commit (working directory clean)
```

You've created a new branch called `hello1` and checked it out. Now, make whatever changes to the kernel you want. I did some work with a simple character driver that we can use as an example:

```
host$ cd ti-linux-kernel-dev/KERNEL/drivers/char/
host$ git status
# On branch hello1
# Changes not staged for commit:
#   (use "git add file..." to update what will be committed)
#   (use "git checkout -- file..." to discard changes in working directory)
#
#   modified:   Kconfig
#   modified:   Makefile
#
# Untracked files:
#   (use "git add file..." to include in what will be committed)
#
#   examples/
no changes added to commit (use "git add" and/or "git commit -a")
```

Add the files that were created and commit them:

```
host$ git add Kconfig Makefile examples
host$ git status
# On branch hello1
# Changes to be committed:
#   (use "git reset HEAD file..." to unstage)
#
#   modified:   Kconfig
#   modified:   Makefile
#   new file:   examples/Makefile
#   new file:   examples/hello1.c
#
host$ git commit -m "Files for hello1 kernel module"
[hello1 99346d5] Files for hello1 kernel module
4 files changed, 33 insertions(+)
create mode 100644 drivers/char/examples/Makefile
create mode 100644 drivers/char/examples/hello1.c
```

Finally, create the patch file:

```
host$ git format-patch master --stdout &gt; hello1.patch
```

15.1.8 Real-Time I/O

Sometimes, when BeagleBone Black interacts with the physical world, it needs to respond in a timely manner. For example, your robot has just detected that one of the driving motors needs to turn a bit faster. Systems that can respond quickly to a real event are known as `real-time` systems. There are two broad categories of real-time systems: soft and hard.

In a `soft real-time` system, the real-time requirements should be met `most` of the time, where `most` depends on the system. A video playback system is a good example. The goal might be to display 60 frames per second, but it doesn't matter much if you miss a frame now and then. In a 100 percent `hard real-time` system, you can never fail to respond in time. Think of an airbag deployment system on a car. You can't even be 50 ms late.

Systems running Linux generally can't do 100 percent hard real-time processing, because Linux gets in the way. However, the Bone has an ARM processor running Linux and two additional 32-bit programmable real-time units (PRUs Ti AM33XX PRUSSv2) available to do real-time processing. Although the PRUs can achieve 100 percent hard real-time, they take some effort to use.

This chapter shows several ways to do real-time input/output (I/O), starting with the effortless, yet slower JavaScript and moving up with increasing speed (and effort) to using the PRUs.

Note: In this chapter, as in the others, we assume that you are logged in as *debian* (as indicated by the *bone$* prompt). This gives you quick access to the general-purpose input/output (GPIO) ports but you may have to use *sudo* some times.

I/O with Python and JavaScript

Problem You want to read an input pin and write it to the output as quickly as possible with JavaScript.

Solution *Reading the Status of a Pushbutton or Magnetic Switch (Passive On/Off Sensor)* shows how to read a pushbutton switch and *Toggling an External LED* controls an external LED. This recipe combines the two to read the switch and turn on the LED in response to it. To make this recipe, you will need:

- Breadboard and jumper wires
- Pushbutton switch
- 220R resistor
- LED

Wire up the pushbutton and LED as shown in *Diagram for wiring a pushbutton and LED with the LED attached to P9_14*.

The code in *Monitoring a pushbutton (pushLED.py)* reads GPIO port *P9_42*, which is attached to the pushbutton, and turns on the LED attached to *P9_12* when the button is pushed.

Python

c

JavaScript

Listing 15.59: Monitoring a pushbutton (pushLED.py)

```
#!/usr/bin/env python
# ////////////////////////////////////
# //        pushLED.py
# //        Blinks an LED attached to P9_12 when the button at P9_42 is␣
 ↪pressed
# //        Wiring:
# //        Setup:
```

(continues on next page)

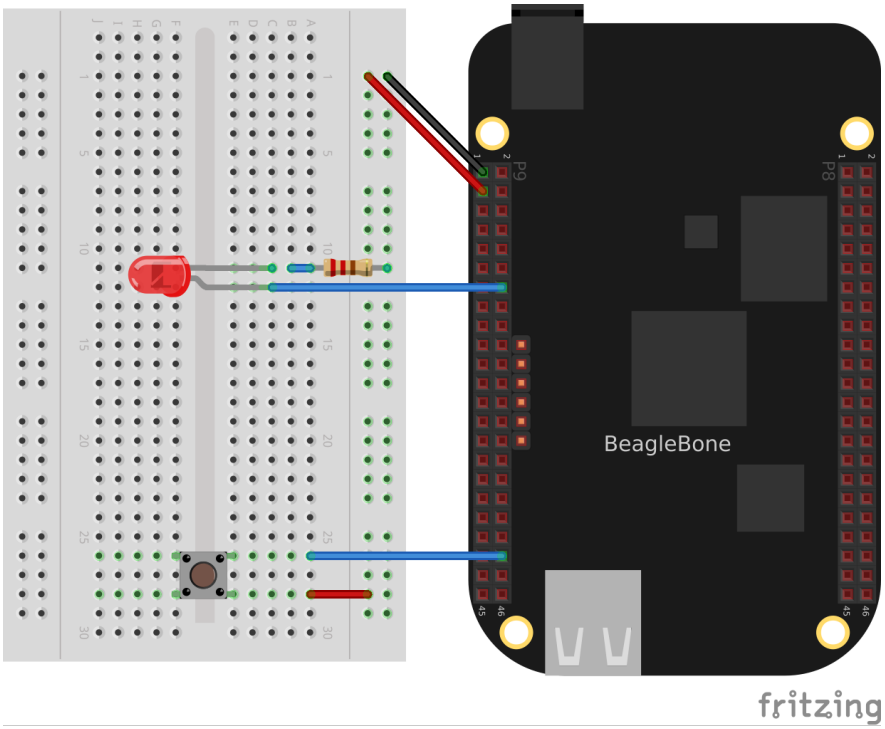


Fig. 15.69: Diagram for wiring a pushbutton and LED with the LED attached to P9_14

(continued from previous page)

```
# //         See:
# ////////////////////////////////////
import time
import os

ms = 50    # Read time in ms

LED="50"    # Look up P9.14 using gpioinfo | grep -e chip -e P9.14.  chip 1, line 18 maps to 50
button="7" # P9_42 mapps to 7

GPIOPATH="/sys/class/gpio/"

# Make sure LED is exported
if (not os.path.exists(GPIOPATH+"gpio"+LED)):
    f = open(GPIOPATH+"export", "w")
    f.write(LED)
    f.close()

# Make it an output pin
f = open(GPIOPATH+"gpio"+LED+"/direction", "w")
f.write("out")
f.close()

# Make sure button is exported
if (not os.path.exists(GPIOPATH+"gpio"+button)):
    f = open(GPIOPATH+"export", "w")
    f.write(button)
    f.close()

# Make it an output pin
f = open(GPIOPATH+"gpio"+button+"/direction", "w")
f.write("in")
```

(continues on next page)

(continued from previous page)

```
f.close()

# Read every ms
fin  = open(GPIOPATH+"gpio"+button+"/value", "r")
fout = open(GPIOPATH+"gpio"+LED+"/value", "w")

while True:
    fin.seek(0)
    fout.seek(0)
    fout.write(fin.read())
    time.sleep(ms/1000)
```

pushLED.py

Listing 15.60: Code for reading a switch and blinking an LED (push-LED.c)

```
////////////////////////////////////
//        blinkLED.c
//        Blinks the P9_14 pin based on the P9_42 pin
//        Wiring:
//        Setup:
//        See:
////////////////////////////////////
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define MAXSTR 100

int main() {
  FILE *fpbutton, *fpLED;
  char LED[] = "50";   // Look up P9.14 using gpioinfo | grep -e chip -e P9.14.  chip 1, line 18 maps to 50
  char button[] = "7"; // Look up P9.42 using gpioinfo | grep -e chip -e P9.42.  chip 0, line 7 maps to 7
  char GPIOPATH[] = "/sys/class/gpio";
  char path[MAXSTR] = "";

  // Make sure LED is exported
  snprintf(path, MAXSTR, "%s%s%s", GPIOPATH, "/gpio", LED);
  if (!access(path, F_OK) == 0) {
    snprintf(path, MAXSTR, "%s%s", GPIOPATH, "/export");
    fpLED = fopen(path, "w");
    fprintf(fpLED, "%s", LED);
    fclose(fpLED);
  }

  // Make it an output LED
  snprintf(path, MAXSTR, "%s%s%s%s", GPIOPATH, "/gpio", LED, "/direction");
  fpLED = fopen(path, "w");
  fprintf(fpLED, "out");
  fclose(fpLED);

  // Make sure bbuttonutton is exported
  snprintf(path, MAXSTR, "%s%s%s", GPIOPATH, "/gpio", button);
  if (!access(path, F_OK) == 0) {
    snprintf(path, MAXSTR, "%s%s", GPIOPATH, "/export");
    fpbutton = fopen(path, "w");
    fprintf(fpbutton, "%s", button);
    fclose(fpbutton);
  }

```

(continues on next page)

(continued from previous page)

```
  // Make it an input button
  snprintf(path, MAXSTR, "%s%s%s%s", GPIOPATH, "/gpio", button, "/direction
↪");
  fpbutton = fopen(path, "w");
  fprintf(fpbutton, "in");
  fclose(fpbutton);

  // I don't know why I can open the LED outside the loop and use fseek␣
↪before
  //  each read, but I can't do the same for the button.  It appears it needs
  //  to be opened every time.
  snprintf(path, MAXSTR, "%s%s%s%s", GPIOPATH, "/gpio", LED,    "/value");
  fpLED    = fopen(path, "w");

  char state = '0';

  while (1) {
    snprintf(path, MAXSTR, "%s%s%s%s", GPIOPATH, "/gpio", button, "/value");
    fpbutton = fopen(path, "r");
    fseek(fpLED, 0L, SEEK_SET);
    fscanf(fpbutton, "%c", &state);
    printf("state: %c\n", state);
    fprintf(fpLED, "%c", state);
    fclose(fpbutton);
    usleep(250000);   // sleep time in microseconds
  }
}
```

```
bone$ gcc -o pushLED pushLED.c -lgpiod
bone$ ./pushLED
1
1
0
0
0
1
^C
```

pushLED.c

Listing 15.61: Monitoring a pushbutton (pushLED.js)

```
#!/usr/bin/env node
////////////////////////////////////
//        pushLED.js
//        Blinks an LED attached to P9_12 when the button at P9_42 is pressed
//        Wiring:
//        Setup:
//        See:
////////////////////////////////////
const fs = require("fs");

const ms = 500   // Read time in ms

const LED="50";   // Look up P9.14 using gpioinfo | grep -e chip -e P9.14. ␣
↪chip 1, line 18 maps to 50
const button="7"; // P9_42 mapps to 7

GPIOPATH="/sys/class/gpio/";

// Make sure LED is exported
```

(continues on next page)

(continued from previous page)

```
if(!fs.existsSync(GPIOPATH+"gpio"+LED)) {
    fs.writeFileSync(GPIOPATH+"export", LED);
}
// Make it an output pin
fs.writeFileSync(GPIOPATH+"gpio"+LED+"/direction", "out");

// Make sure button is exported
if(!fs.existsSync(GPIOPATH+"gpio"+button)) {
    fs.writeFileSync(GPIOPATH+"export", button);
}
// Make it an input pin
fs.writeFileSync(GPIOPATH+"gpio"+button+"/direction", "in");

// Read every ms
setInterval(flashLED, ms);

function flashLED() {
    var data = fs.readFileSync(GPIOPATH+"gpio"+button+"/value").slice(0, -1);
    console.log('data = ' + data);
    fs.writeFileSync(GPIOPATH+"gpio"+LED+"/value", data);
 }
```

`pushLED.js`

Add the code to a file named `pushLED.py` and run it by using the following commands:

```
bone$ chmod *x pushLED.py
bone$ ./pushLED.py
Hit ^C to stop
0
0
1
1
^C
```

Press ^C (Ctrl-C) to stop the code.

I/O with devmem2

Problem Your C code isn't responding fast enough to the input signal. You want to read the GPIO registers directly.

Solution The solution is to use a simple utility called *devmem2*, with which you can read and write registers from the command line.

Warning: This solution is much more involved than the previous ones. You need to understand binary and hex numbers and be able to read the AM335x Technical Reference Manual.

First, download and install *devmem2*:

```
bone$ wget http://bootlin.com/pub/mirror/devmem2.c
bone$ gcc -o devmem2 devmem2.c
bone$ sudo mv devmem2 /usr/bin
```

This solution will read a pushbutton attached to *P9_42* and flash an LED attached to *P9_13*. Note that this is a change from the previous solutions that makes the code used here much simpler. Wire up your Bone as shown in *Diagram for wiring a pushbutton and LED with the LED attached to P9_13*.

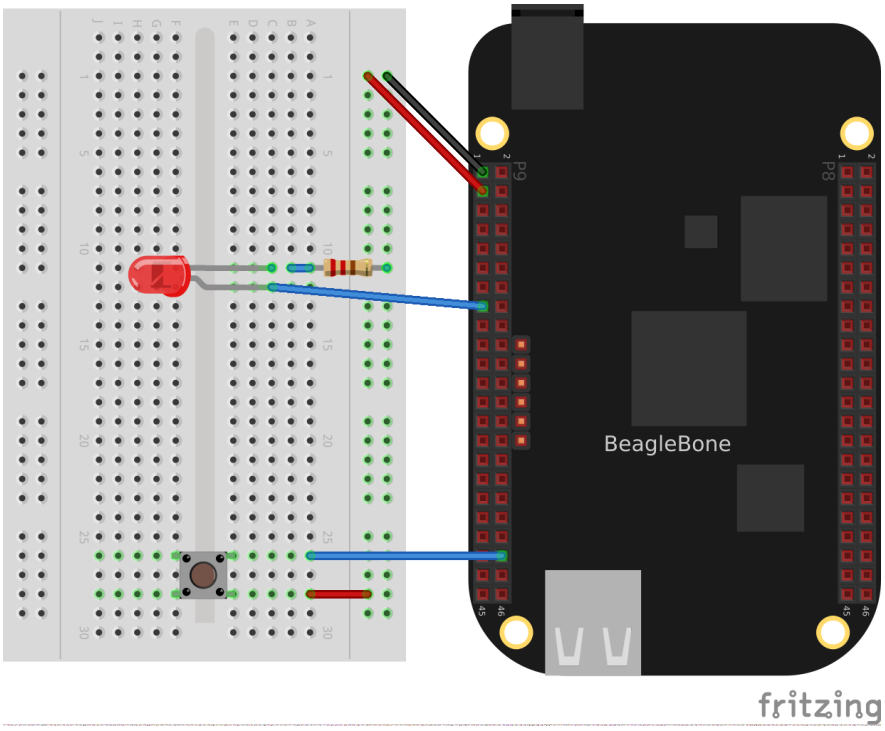


Fig. 15.70: Diagram for wiring a pushbutton and LED with the LED attached to P9_13

Now, flash the LED attached to *P9_13* using the Linux *sysfs* interface (*Controlling GPIOs by Using SYSFS Entries*). To do this, first look up which GPIO number *P9_13* is attached to by referring to *Mapping from header pin to internal GPIO number*. Finding *P9_13* at GPIO 31, export GPIO 31 and make it an output:

```
bone$ cd cd /sys/class/gpio/
bone$ echo 31 > export
bone$ cd gpio31
bone$ echo out > direction
bone$ echo 1 > value
bone$ echo 0 > value
```

The LED will turn on when *1* is echoed into *value* and off when *0* is echoed.

Now that you know the LED is working, look up its memory address. This is where things get very detailed. First, download the AM335x Technical Reference Manual. Look up *GPIO0* in the Memory Map chapter (sensors). Table 2-2 indicates that *GPIO0* starts at address *0x44E0_7000*. Then go to Section 25.4.1, "GPIO Registers." This shows that *GPIO_DATAIN* has an offset of *0x138*, *GPIO_CLEARDATAOUT* has an offset of *0x190*, and *GPIO_SETDATAOUT* has an offset of *0x194*.

This means you read from address *0x44E0_7000* + *0x138* = *0x44E0_7138* to see the status of the LED:

```
bone$ sudo devmem2 0x44E07138
/dev/mem opened.
Memory mapped at address 0xb6f8e000.
Value at address 0x44E07138 (0xb6f8e138): 0xC000C404
```

The returned value *0xC000C404* (*1100 0000 0000 0000 1100 0100 0000 0100* in binary) has bit 31 set to *1*, which means the LED is on. Turn the LED off by writing *0x80000000* (*1000 0000 0000 0000 0000 0000 0000 0000* binary) to the *GPIO_CLEARDATA* register at *0x44E0_7000* + *0x190* = *0x44E0_7190*:

```
bone$ sudo devmem2 0x44E07190 w 0x80000000
/dev/mem opened.
Memory mapped at address 0xb6fd7000.
Value at address 0x44E07190 (0xb6fd7190): 0x80000000
```

(continues on next page)

(continued from previous page)

```
Written 0x80000000; readback 0x0
```

The LED is now off.

You read the pushbutton switch in a similar way. *Mapping from header pin to internal GPIO number* says *P9_42* is GPIO 7, which means bit 7 is the state of *P9_42*. The *devmem2* in this example reads *0x0*, which means all bits are *0*, including GPIO 7. Section 25.4.1 of the Technical Reference Manual instructs you to use offset *0x13C* to read *GPIO_DATAOUT*. Push the pushbutton and run *devmem2*:

```
bone$ sudo devmem2 0x44e07138
/dev/mem opened.
Memory mapped at address 0xb6fe2000.
Value at address 0x44E07138 (0xb6fe2138): 0x4000C484
```

Here, bit 7 is set in *0x4000C484*, showing the button is pushed.

This is much more tedious than the previous methods, but it's what's necessary if you need to minimize the time to read an input. *I/O with C and mmap()* shows how to read and write these addresses from C.

I/O with C and mmap()

Problem Your C code isn't responding fast enough to the input signal.

Solution In smaller processors that aren't running an operating system, you can read and write a given memory address directly from C. With Linux running on Bone, many of the memory locations are hardware protected, so you can't accidentally access them directly.

This recipe shows how to use *mmap()* (memory map) to map the GPIO registers to an array in C. Then all you need t o do is access the array to read and write the registers.

Warning: This solution is much more involved than the previous ones. You need to understand binary and hex numbers and be able to read the AM335x Technical Reference Manual.

This solution will read a pushbutton attached to *P9_42* and flash an LED attached to *P9_13*. Note that this is a change from the previous solutions that makes the code used here much simpler.

Tip: See *I/O with devmem2* for details on mapping the GPIO numbers to memory addresses.

Add the code in *Memory address definitions (pushLEDmmap.h)* to a file named `pushLEDmmap.h`.

Listing 15.62: Memory address definitions (pushLEDmmap.h)

```
1  // From: http://stackoverflow.com/questions/13124271/driving-beaglebone-gpio
2  // -through-dev-mem
3  // user contributions licensed under cc by-sa 3.0 with attribution required
4  // http://creativecommons.org/licenses/by-sa/3.0/
5  // http://blog.stackoverflow.com/2009/06/attribution-required/
6  // Author: madscientist159 (http://stackoverflow.com/users/3000377/
   ↪madscientist159)
7
8  #ifndef _BEAGLEBONE_GPIO_H_
9  #define _BEAGLEBONE_GPIO_H_
10
11 #define GPIO0_START_ADDR 0x44e07000
12 #define GPIO0_END_ADDR   0x44e08000
13 #define GPIO0_SIZE (GPIO0_END_ADDR - GPIO0_START_ADDR)
14
```

(continues on next page)

(continued from previous page)

```
#define GPIO1_START_ADDR 0x4804C000
#define GPIO1_END_ADDR   0x4804D000
#define GPIO1_SIZE (GPIO1_END_ADDR - GPIO1_START_ADDR)

#define GPIO2_START_ADDR 0x41A4C000
#define GPIO2_END_ADDR   0x41A4D000
#define GPIO2_SIZE (GPIO2_END_ADDR - GPIO2_START_ADDR)

#define GPIO3_START_ADDR 0x41A4E000
#define GPIO3_END_ADDR   0x41A4F000
#define GPIO3_SIZE (GPIO3_END_ADDR - GPIO3_START_ADDR)

#define GPIO_DATAIN 0x138
#define GPIO_SETDATAOUT 0x194
#define GPIO_CLEARDATAOUT 0x190

#define GPIO_03  (1<<3)
#define GPIO_07  (1<<7)
#define GPIO_31  (1<<31)
#define GPIO_60  (1<<28)
#endif
```

`pushLEDmmap.h`

Add the code in *Code for directly reading memory addresses (pushLEDmmap.c)* to a file named `pushLEDmmap.c`.

Listing 15.63: Code for directly reading memory addresses (pushLEDmmap.c)

```
// From: http://stackoverflow.com/questions/13124271/driving-beaglebone-gpio
// -through-dev-mem
// user contributions licensed under cc by-sa 3.0 with attribution required
// http://creativecommons.org/licenses/by-sa/3.0/
// http://blog.stackoverflow.com/2009/06/attribution-required/
// Author: madscientist159 (http://stackoverflow.com/users/3000377/
↪madscientist159)
//
// Read one gpio pin and write it out to another using mmap.
// Be sure to set -O3 when compiling.
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <signal.h>    // Defines signal-handling functions (i.e. trap Ctrl-
↪C)
#include "pushLEDmmap.h"

// Global variables
int keepgoing = 1;    // Set to 0 when Ctrl-c is pressed

// Callback called when SIGINT is sent to the process (Ctrl-C)
void signal_handler(int sig) {
    printf( "\nCtrl-C pressed, cleaning up and exiting...\n" );
        keepgoing = 0;
}

int main(int argc, char *argv[]) {
    volatile void *gpio_addr;
    volatile unsigned int *gpio_datain;
    volatile unsigned int *gpio_setdataout_addr;
    volatile unsigned int *gpio_cleardataout_addr;
```

(continues on next page)

(continued from previous page)

```
31
32      // Set the signal callback for Ctrl-C
33      signal(SIGINT, signal_handler);
34
35      int fd = open("/dev/mem", O_RDWR);
36
37      printf("Mapping %X - %X (size: %X)\n", GPIO0_START_ADDR, GPIO0_END_ADDR,
38                                             GPIO0_SIZE);
39
40      gpio_addr = mmap(0, GPIO0_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
41                       GPIO0_START_ADDR);
42
43      gpio_datain             = gpio_addr + GPIO_DATAIN;
44      gpio_setdataout_addr    = gpio_addr + GPIO_SETDATAOUT;
45      gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;
46
47      if(gpio_addr == MAP_FAILED) {
48          printf("Unable to map GPIO\n");
49          exit(1);
50      }
51      printf("GPIO mapped to %p\n", gpio_addr);
52      printf("GPIO SETDATAOUTADDR mapped to %p\n", gpio_setdataout_addr);
53      printf("GPIO CLEARDATAOUT mapped to %p\n", gpio_cleardataout_addr);
54
55      printf("Start copying GPIO_07 to GPIO_31\n");
56      while(keepgoing) {
57              if(*gpio_datain & GPIO_07) {
58              *gpio_setdataout_addr= GPIO_31;
59              } else {
60              *gpio_cleardataout_addr = GPIO_31;
61              }
62          //usleep(1);
63      }
64
65      munmap((void *)gpio_addr, GPIO0_SIZE);
66      close(fd);
67      return 0;
68  }
```

pushLEDmmap.c

Now, compile and run the code:

```
bone$ gcc -O3 pushLEDmmap.c -o pushLEDmmap
bone$ sudo ./pushLEDmmap
Mapping 44E07000 - 44E08000 (size: 1000)
GPIO mapped to 0xb6fac000
GPIO SETDATAOUTADDR mapped to 0xb6fac194
GPIO CLEARDATAOUT mapped to 0xb6fac190
Start copying GPIO_07 to GPIO_31
^C
Ctrl-C pressed, cleaning up and exiting...
```

The code is in a tight *while* loop that checks the status of GPIO 7 and copies it to GPIO 31.

Tighter Delay Bounds with the PREEMPT_RT Kernel

Problem You want to run real-time processes on the Beagle, but the OS is slowing things down.

Solution The Kernel can be compiled with PREEMPT_RT enabled which reduces the delay from when a thread is scheduled to when it runs.

Switching to a PREEMPT_RT kernel is rather easy, but be sure to follow the steps in the Discussion to see how much the latencies are reduced.

- First see which kernel you are running:

```
bone$ uname -a
Linux breadboard-home 5.10.120-ti-r47 #1bullseye SMP PREEMPT Tue Jul 12␣
↪18:59:38 UTC 2022 armv7l GNU/Linux
```

I'm running a 5.10 kernel. Remember the whole string, *5.10.120-ti-r47*, for later.

- Go to kernel update and look for *5.10*.

v5.10.x-ti branch:

```
bbb.io-kernel-5.10-ti-am335x - BeagleBoard.org 5.10-ti for am335x
bbb.io-kernel-5.10-ti-am57xx - BeagleBoard.org 5.10-ti for am57xx
```

v5.10.x-ti-rt branch:

```
bbb.io-kernel-5.10-ti-rt-am335x - BeagleBoard.org 5.10-ti-rt for am335x
bbb.io-kernel-5.10-ti-rt-am57xx - BeagleBoard.org 5.10-ti-rt for am57xx
```

Fig. 15.71: The regular and RT kernels

In *The regular and RT kernels* you see the reular kernel on top and the RT below.

- We want the RT one.

```
bone$ sudo apt update
bone$ sudo apt install bbb.io-kernel-5.10-ti-rt-am335x
```

Note: Use the *am57xx* if you are using the BeagleBoard AI or AI64.

- Before rebooting, edit */boot/uEnv.txt* to start with:

```
#Docs: http://elinux.org/Beagleboard:U-boot_partitioning_layout_2.0

# uname_r=5.10.120-ti-r47
uname_r=5.10.120-ti-rt-r47
#uuid=
#dtb=
```

uname_r tells the boot loader which kernel to boot. Here we've commented out the regular kernel and left in the RT kernel. Next time you boot you'll be running the RT kernel. Don't reboot just yet. Let's gather some latency data first.

Bootlin's preempt_rt workshop looks like a good workshop on PREEMPT RT. Their slides say:

- One way to implement a multi-task Real-Time Operating System is to have a preemptible system
- Any task can be interrupted at any point so that higher priority tasks can run
- Userspace preemption already exists in Linux

- The Linux Kernel also supports real-time scheduling policies
- However, code that runs in kernel mode isn't fully preemptible
- The Preempt-RT patch aims at making all code running in kernel mode preemptible

The workshop goes into many details on how to get real-time performance on Linux. Checkout their slides and labs. Though you can skip the first lab since we present a simpler way to get the RT kernel running.

Cyclictest

cyclictest is one tool for measuring the latency from when a thread is schduled and when it runs. The *code/rt* directory in the git repo has some scripts for gathering latency data and plotting it. Here's how to run the scripts.

- First look in *rt/install.sh* to see what to install.

Listing 15.64: rt/install.sh

```
sudo apt install rt-tests
# You can run gnuplot on the host
sudo apt install gnuplot
```

rt/install.sh

- Open up another window and start something that will create a load on the Bone, then run the following:

```
bone$ time sudo ./hist.gen > nort.hist
```

hist.gen shows what's being run. It defaults to 100,000 loops, so it takes a while. The data is saved in *nort.hist*, which stands for no RT histogram.

Listing 15.65: hist.gen

```
#!/bin/sh
# This code is from Julia Cartwright julia@kernel.org

cyclictest -m -S -p 90 -h 400 -l "${1:-100000}"
```

rt/hist.gen

Note: If you get an error:

Unable to change scheduling policy! Probably missing capabilities, either run as root or increase RLIMIT_RTPRIO limits

try running ./setup.sh. If that doesn't work try:

```
bone$ sudo bash
bone# ulimit -r unlimited
bone# ./hist.gen > nort.hist
bone# exit
```

- Now you are ready to reboot into the RT kernel and run the test again.

```
bone$ reboot
```

- After rebooting:

```
bone$ uname -a
Linux breadboard-home 5.10.120-ti-rt-r47 #1bullseye SMP PREEMPT RT Tue Jul␣
↪12 18:59:38 UTC 2022 armv7l GNU/Linux
```

Congratulations you are running the RT kernel.

Note: If the Beagle appears to be running (the LEDs are flashing) but you are having trouble connecting via *ssh 192.168.7.2*, you can try connecting using the approach shown in *Viewing and Debugging the Kernel and u-boot Messages at Boot Time*.

Now run the script again (note it's being saved in *rt.hist* this time.)

```
bone$ time sudo ./hist.gen > rt.hist
```

Note: At this point yoou can edit */boot/uEnt.txt* to boot the non RT kernel and reboot.

Now it's time to plot the results.

```
bone$ gnuplot hist.plt
```

This will generate the file *cyclictest.png* which contains your plot. It should look like:

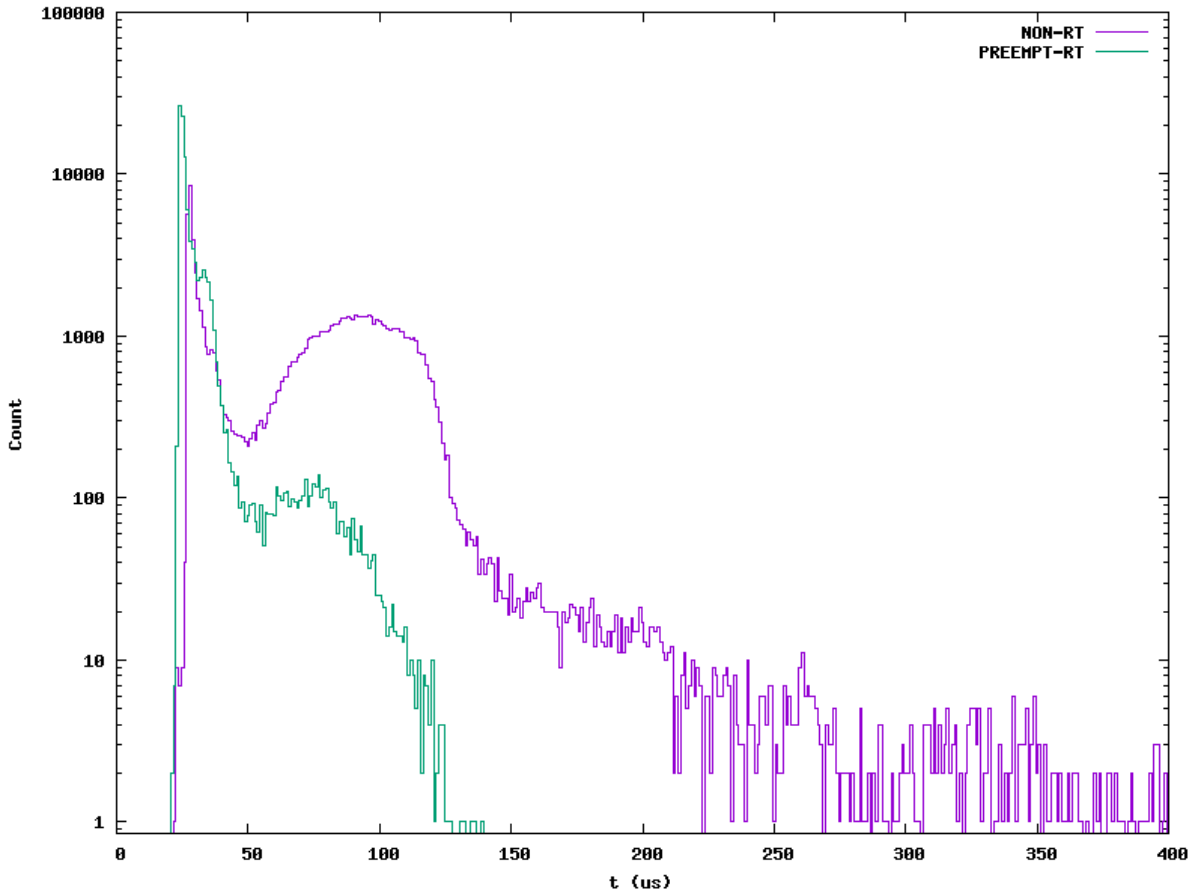


Fig. 15.72: Histogram of Non-RT and RT kernels running cyclictest

Notice the NON-RT data have much longer latenices. They may not happen often (fewer than 10 times in each bin), but they are occurring and may be enough to miss a real-time deadline.

The PREEMPT-RT times are all under a 150 us.

I/O with simpPRU

Problem You require better timing than running C on the ARM can give you.

Solution The AM335x processor on the Bone has an ARM processor that is running Linux, but it also has two 32-bit PRUs that are available for processing I/O. It takes a fair amount of understanding to program the PRU. Fortunately, simpPRU is an intuitive language for PRU which compiles down to PRU C. This solution shows how to use it.

Background

simpPRU

15.1.9 Capes

Previous chapters of this book show a variety of ways to interface BeagleBone Black to the physical world by using a breadboard and wiring to the +P8+ and +P9+ headers. This is a great approach because it's easy to modify your circuit to debug it or try new things. At some point, though, you might want a more permanent solution, either because you need to move the Bone and you don't want wires coming loose, or because you want to share your hardware with the masses.

You can easily expand the functionality of the Bone by adding a cape. A cape is simply a board–often a printed circuit board (PCB) that connects to the **P8** and **P9** headers and follows a few standard pin usages. You can stack up to four capes onto the Bone. Capes can range in size covering a few pins to much larger than the Bone.

Todo: Add cape examples of various sizes

This chapter shows how to attach a couple of capes, move your design to a protoboard, then to a PCB, and finally on to mass production.

Todo: Update display cape example

Connecting Multiple Capes

Problem You want to use more than one cape at a time.

Solution First, look at each cape that you want to stack mechanically. Are they all using stacking headers like the ones shown in *Stacking headers*? No more than one should be using non-stacking headers.

Note that larger LCD panels might provide expansion headers, such as the ones shown in *LCD Backside*, rather than the stacking headers, and that those can also be used for adding additional capes.

LCD Backside

Note: Back side of LCD7 cape, *LCD Backside* was originally posted by CircuitCo at http://elinux.org/File:BeagleBone-LCD-Backside.jpg under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

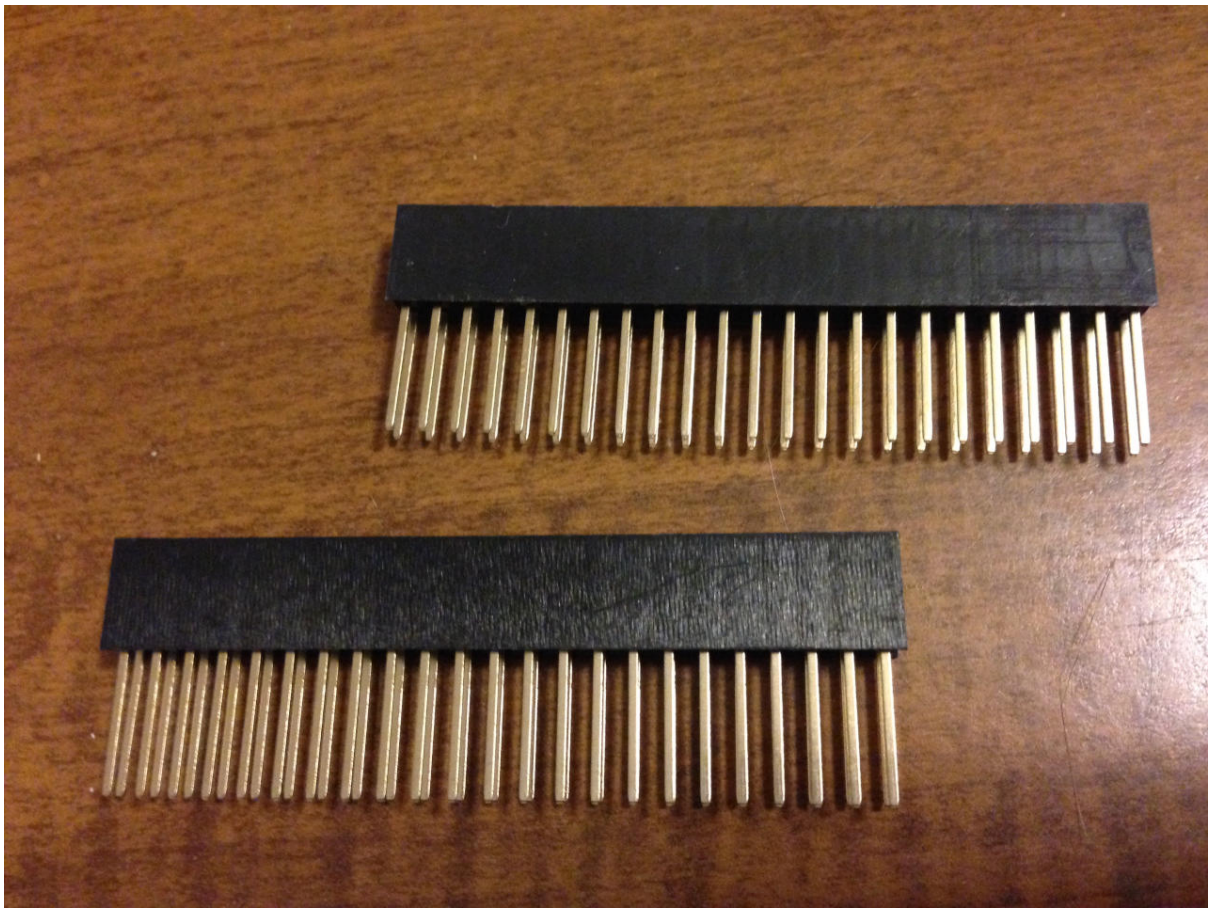
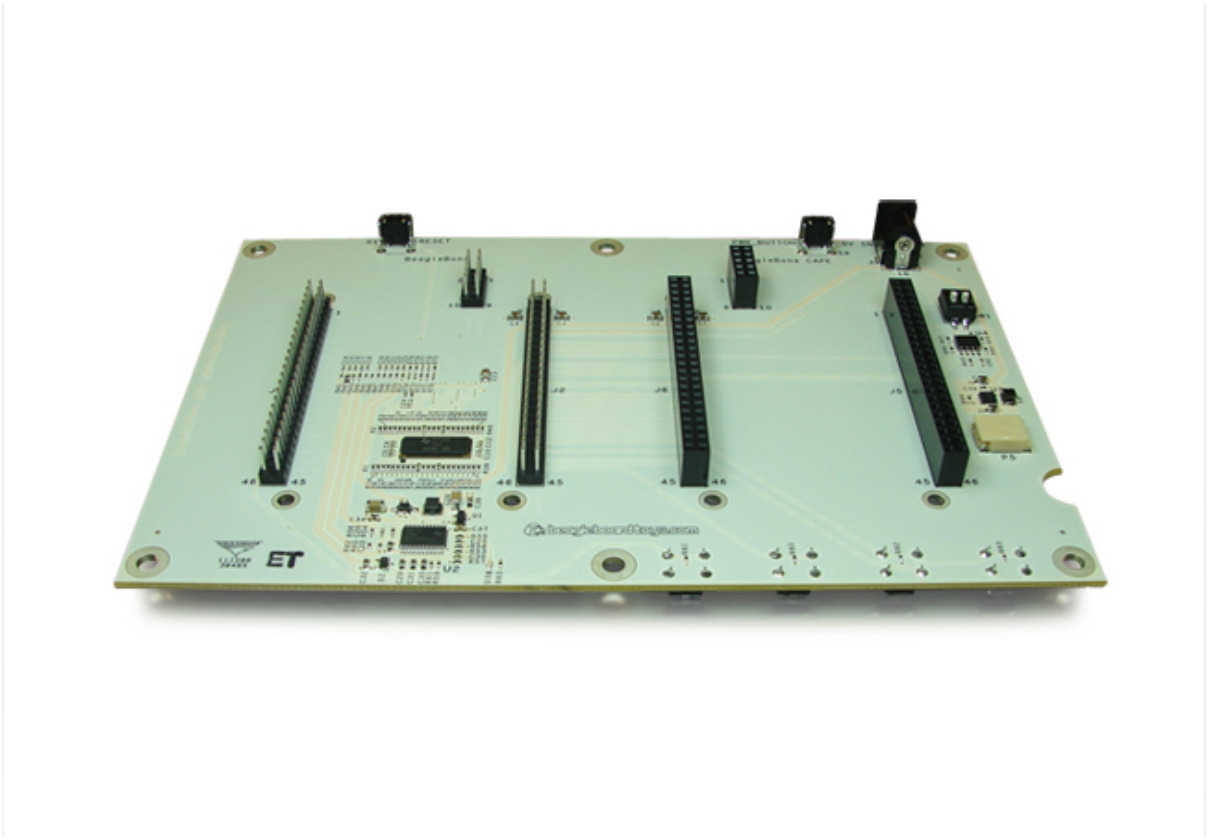


Fig. 15.73: Stacking headers

Note: #TODO# One of the 4D Systems LCD capes would make a better example for an LCD cape. The CircuitCo cape is no longer available.



Next, take a note of each pin utilized by each cape. The BeagleBone Capes catalog provides a graphical representation for the pin usage of most capes, as shown in *Audio cape pins* for the Circuitco Audio Cape.

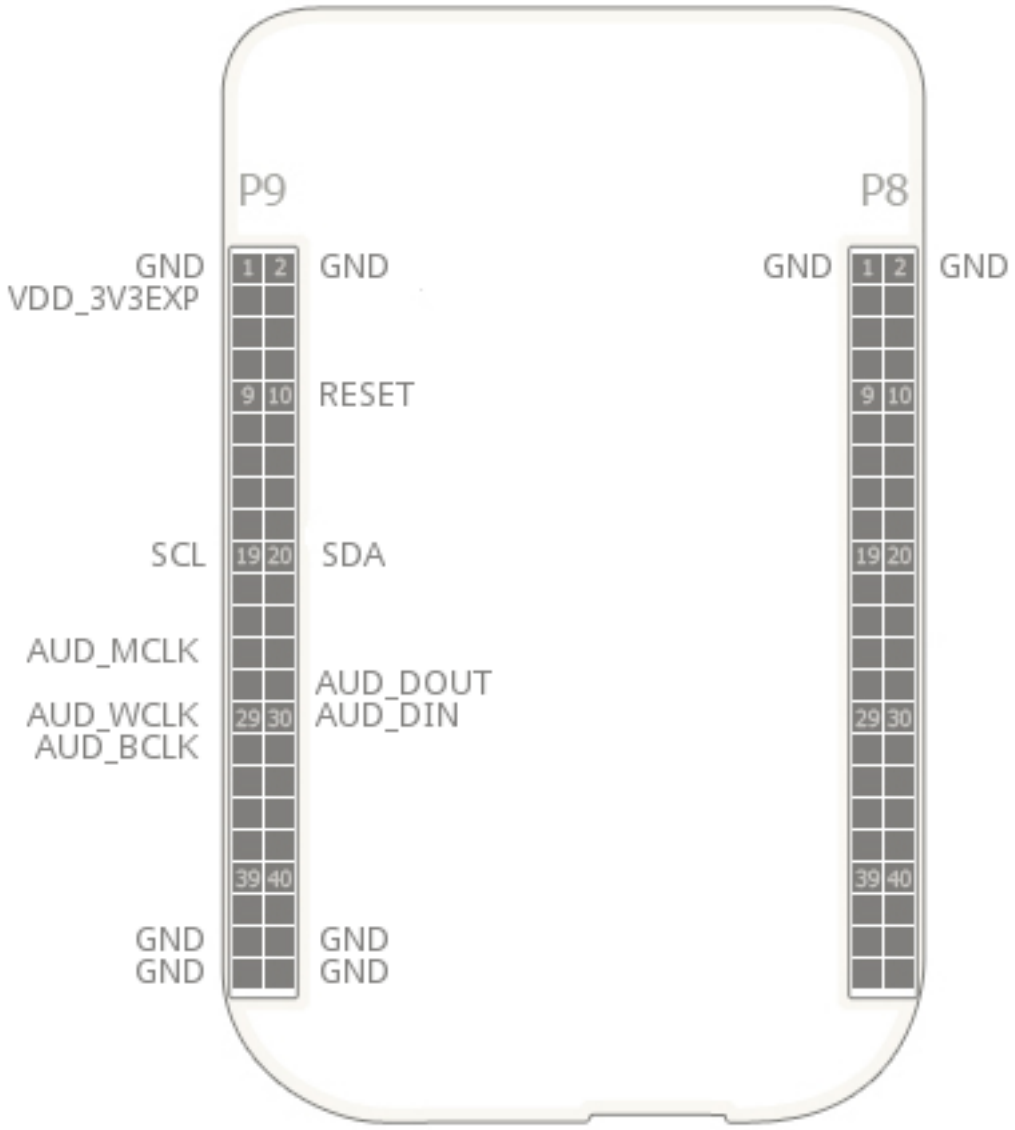
Note: #TODO# Bela would make a better example for an audio cape. The CircuitCo cape is no longer available.

Audio cape pins

Note: Pins utilized by CircuitCo Audio Cape, *Audio cape pins* was originally posted by Djackson at http://elinux.org/File:Audio_pins_revb.png under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

In most cases, the same pin should never be used on two different capes, though in some cases, pins can be shared. Here are some exceptions:

- **GND**
  - The ground (*GND*) pins should be shared between the capes, and there's no need to worry about consumed resources on those pins.
- **VDD_3V3**
  - The 3.3 V power supply (*VDD_3V3*) pins can be shared by all capes to supply power, but the total combined consumption of all the capes should be less than 500 mA (250 mA per *VDD_3V3* pin).


P9
P8
GND
VDD_3V3EXP
GND
RESET
SCL
SDA
AUD_MCLK
AUD_DOUT
AUD_WCLK
AUD_DIN
AUD_BCLK
GND
GND
GND
GND
GND
1 2
9 10
19 20
29 30
39 40

- **VDD_5V** The 5.0 V power supply (*VDD_5V*) pins can be shared by all capes to supply power, but the total combined consumption of all the capes should be less than 2 A (1 A per +VDD_5V+ pin). It is possible for one, and only one, of the capes to *provide* power to this pin rather than consume it, and it should provide at least 3 A to ensure proper system function. Note that when no voltage is applied to the DC connector, nor from a cape, these pins will not be powered, even if power is provided via USB.
- **SYS_5V** The regulated 5.0 V power supply (*SYS_5V*) pins can be shared by all capes to supply power, but the total combined consumption of all the capes should be less than 500 mA (250 mA per *SYS_5V* pin).
- **VADC and AGND**
  - The ADC reference voltage pins can be shared by all capes.
- **I2C2_SCL and I2C2_SDA**
  - I^2C is a shared bus, and the *I2C2_SCL* and *I2C2_SDA* pins default to having this bus enabled for use by cape expansion ID EEPROMs.

Moving from a Breadboard to a Protoboard

Problem You have your circuit working fine on the breadboard, but you want a more reliable solution.

Solution Solder your components to a protoboard.

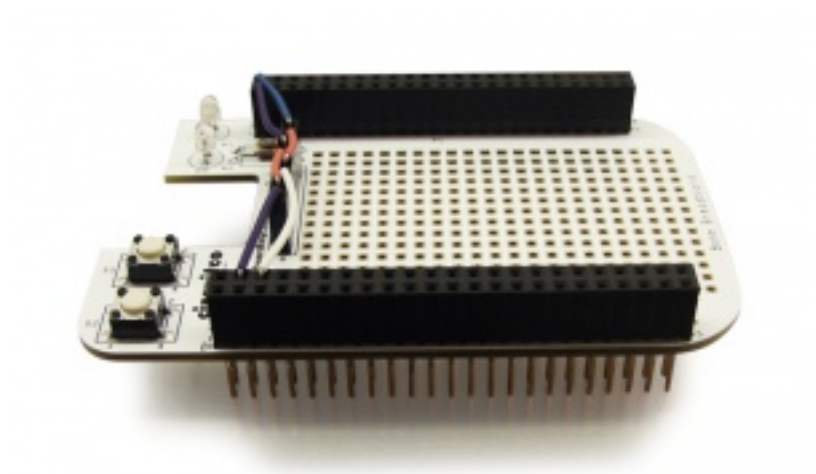
To make this recipe, you will need:

- Protoboard
- Soldering iron
- Your other components

Many places make premade circuit boards that are laid out like the breadboard we have been using. The Adafruit Proto Cape Kit is one protoboard option.

BeagleBone Breadboard

Note: This was originally posted by William Traynor at http://elinux.org/File:BeagleBone-Breadboard.jpg under a Creative Commons Attribution-ShareAlike 3.0 Unported License



You just solder your parts on the protoboard as you had them on the breadboard.

Creating a Prototype Schematic

Problem You've wired up a circuit on a breadboard. How do you turn that prototype into a schematic others can read and that you can import into other design tools?

Solution In *Fritzing tips*, we introduced Fritzing as a useful tool for drawing block diagrams. Fritzing can also do circuit schematics and printed-circuit layout. For example, *A simple robot controller diagram (quickBot.fzz)* shows a block diagram for a simple robot controller (quickBot.fzz is the name of the Fritzing file used to create the diagram).

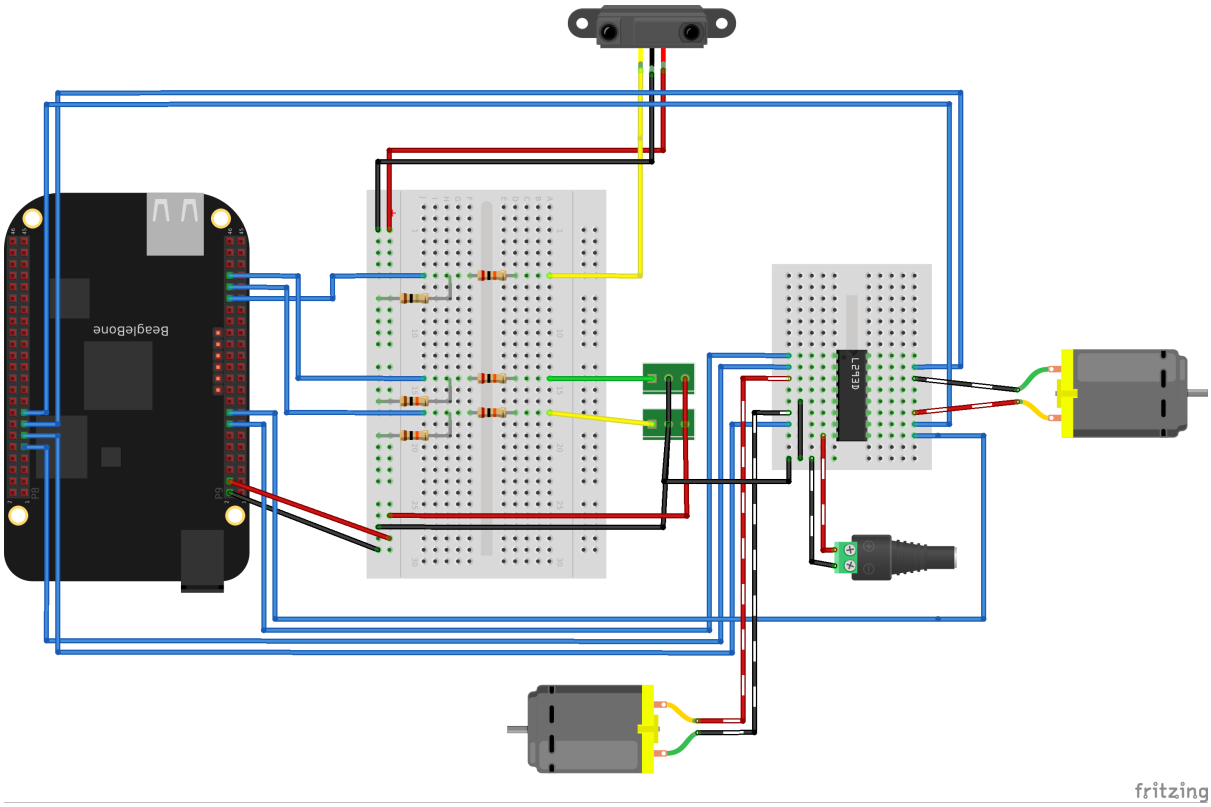


Fig. 15.74: A simple robot controller diagram (quickBot.fzz)

The controller has an H-bridge to drive two DC motors (*Controlling the Speed and Direction of a DC Motor*), an IR range sensor, and two headers for attaching analog encoders for the motors. Both the IR sensor and the encoders have analog outputs that exceed 1.8 V, so each is run through a voltage divider (two resistors) to scale the voltage to the correct range (see *Reading a Distance Sensor (Variable Pulse Width Sensor)* for a voltage divider example).

Automatically generated schematic shows the schematic automatically generated by Fritzing. It's a mess. It's up to you to fix it.

Cleaned-up schematic shows my cleaned-up schematic. I did it by moving the parts around until it looked better.

You might find that you want to create your design in a more advanced design tool, perhaps because it has the library components you desire, it integrates better with other tools you are using, or it has some other feature (such as simulation) of which you'd like to take advantage.

Verifying Your Cape Design

Problem You've got a design. How do you quickly verify that it works?

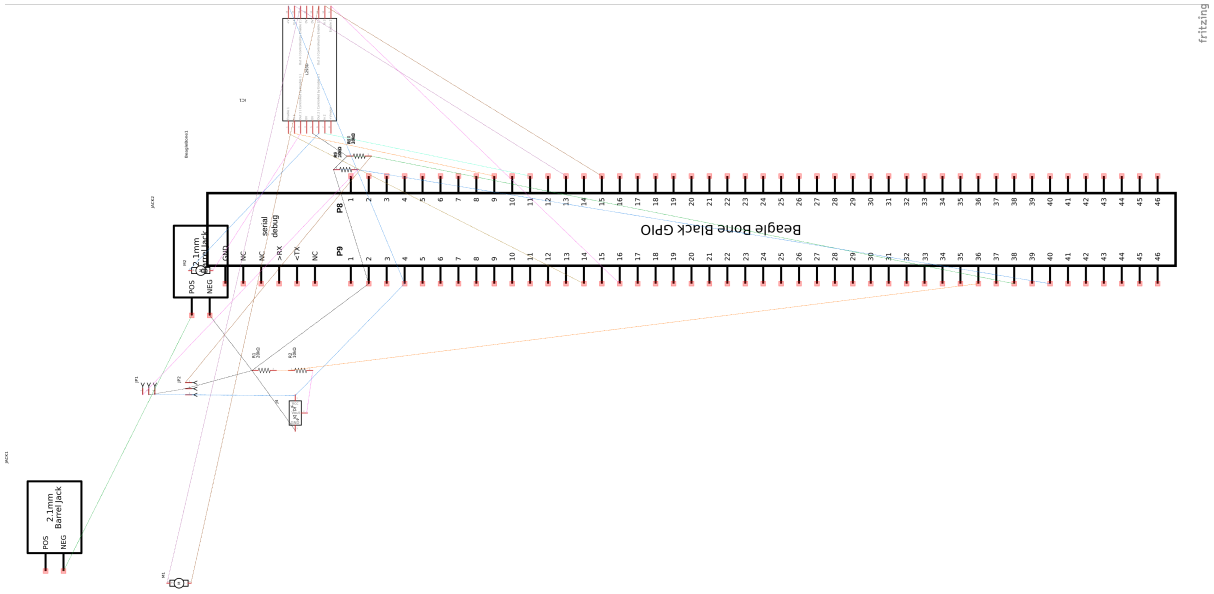


Fig. 15.75: Automatically generated schematic

Solution To make this recipe, you will need:

- An oscilloscope

Break down your design into functional subcomponents and write tests for each. Use components you already know are working, such as the onboard LEDs, to display the test status with the code in *Testing the quickBot motors interface (quickBot_motor_test.js)*.

Testing the quickBot motors interface (quickBot_motor_test.js)

```
#!/usr/bin/env node
var b = require('bonescript');
var M1_SPEED    = 'P9_16'; // 
var M1_FORWARD  = 'P8_15';
var M1_BACKWARD = 'P8_13';
var M2_SPEED    = 'P9_14';
var M2_FORWARD  = 'P8_9';
var M2_BACKWARD = 'P8_11';
var freq = 50; // 
var fast = 0.95;
var slow = 0.7;
var state = 0; // 

b.pinMode(M1_FORWARD, b.OUTPUT); // 
b.pinMode(M1_BACKWARD, b.OUTPUT);
b.pinMode(M2_FORWARD, b.OUTPUT);
b.pinMode(M2_BACKWARD, b.OUTPUT);
b.analogWrite(M1_SPEED, 0, freq); // 
b.analogWrite(M2_SPEED, 0, freq);

updateMotors(); // 

function updateMotors() {
    //console.log("Setting state = " + state); // 
    updateLEDs(state);
    switch(state) { // 
        case 0:
        default:
```

(continues on next page)

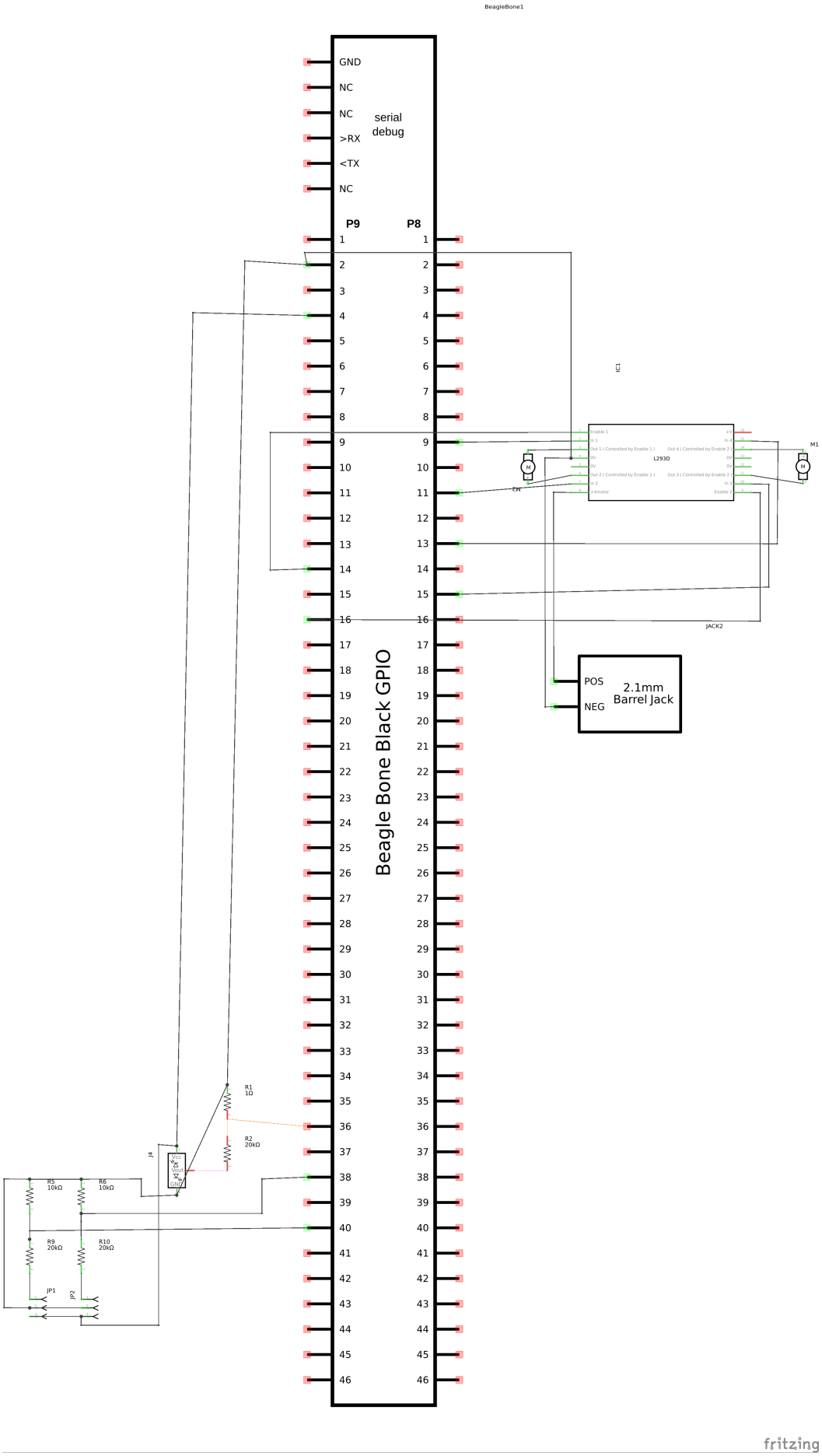


Fig. 15.76: Cleaned-up schematic

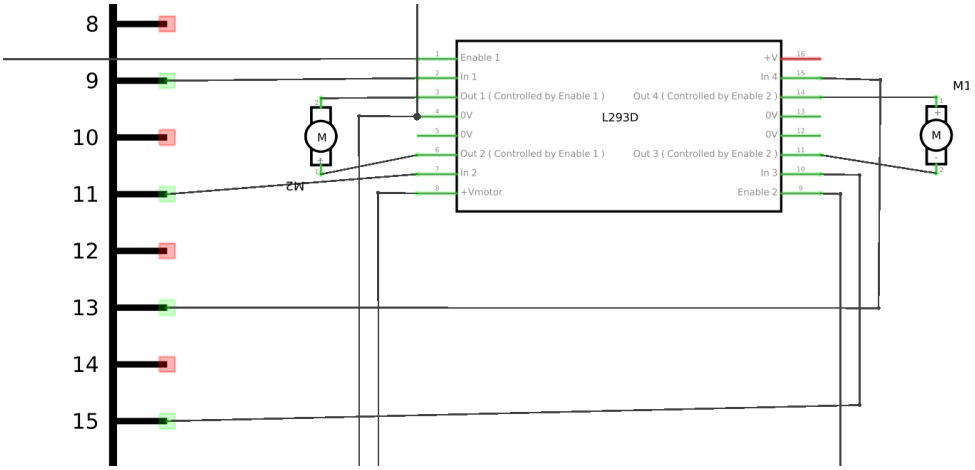


Fig. 15.77: Zoomed-in schematic

(continued from previous page)

```
            M1_set(0); // 
            M2_set(0);
            state = 1; // 
            break;
        case 1:
            M1_set(slow);
            M2_set(slow);
            state = 2;
            break;
        case 2:
            M1_set(slow);
            M2_set(-slow);
            state = 3;
            break;
        case 3:
            M1_set(-slow);
            M2_set(slow);
            state = 4;
            break;
        case 4:
            M1_set(fast);
            M2_set(fast);
            state = 0;
            break;
    }
    setTimeout(updateMotors, 2000); // 
}

function updateLEDs(state) { // 
    switch(state) {
    case 0:
        b.digitalWrite("USR0", b.LOW);
        b.digitalWrite("USR1", b.LOW);
        b.digitalWrite("USR2", b.LOW);
        b.digitalWrite("USR3", b.LOW);
        break;
    case 1:
        b.digitalWrite("USR0", b.HIGH);
        b.digitalWrite("USR1", b.LOW);
        b.digitalWrite("USR2", b.LOW);
        b.digitalWrite("USR3", b.LOW);
        break;
```

(continues on next page)

(continued from previous page)

```
    case 2:
        b.digitalWrite("USR0", b.LOW);
        b.digitalWrite("USR1", b.HIGH);
        b.digitalWrite("USR2", b.LOW);
        b.digitalWrite("USR3", b.LOW);
        break;
    case 3:
        b.digitalWrite("USR0", b.LOW);
        b.digitalWrite("USR1", b.LOW);
        b.digitalWrite("USR2", b.HIGH);
        b.digitalWrite("USR3", b.LOW);
        break;
    case 4:
        b.digitalWrite("USR0", b.LOW);
        b.digitalWrite("USR1", b.LOW);
        b.digitalWrite("USR2", b.LOW);
        b.digitalWrite("USR3", b.HIGH);
        break;
    }
}

function M1_set(speed) { // ②
    speed = (speed > 1) ? 1 : speed; // ②
    speed = (speed < -1) ? -1 : speed;
    b.digitalWrite(M1_FORWARD, b.LOW);
    b.digitalWrite(M1_BACKWARD, b.LOW);
    if(speed > 0) {
        b.digitalWrite(M1_FORWARD, b.HIGH);
    } else if(speed < 0) {
        b.digitalWrite(M1_BACKWARD, b.HIGH);
    }
    b.analogWrite(M1_SPEED, Math.abs(speed), freq); // ⑤
}

function M2_set(speed) {
    speed = (speed > 1) ? 1 : speed;
    speed = (speed < -1) ? -1 : speed;
    b.digitalWrite(M2_FORWARD, b.LOW);
    b.digitalWrite(M2_BACKWARD, b.LOW);
    if(speed > 0) {
        b.digitalWrite(M2_FORWARD, b.HIGH);
    } else if(speed < 0) {
        b.digitalWrite(M2_BACKWARD, b.HIGH);
    }
    b.analogWrite(M2_SPEED, Math.abs(speed), freq);
```

① Define each pin as a variable. This makes it easy to change to another pin if you decide that is necessary.

② Make other simple parameters variables. Again, this makes it easy to update them. When creating this test, I found that the PWM frequency to drive the motors needed to be relatively low to get over the kickback shown in *quickBot motor test showing kickback*. I also found that I needed to get up to about 70 percent duty cycle for my circuit to reliably start the motors turning.

③ Use a simple variable such as *state* to keep track of the test phase. This is used in a *switch* statement to jump to the code to configure for that test phase and updated after configuring for the current phase in order to select the next phase. Note that the next phase isn't entered until after a two-second delay, as specified in the call to *setTimeout()*.

④ Perform the initial setup of all the pins.

⑤ The first time a PWM pin is used, it is configured with the update frequency. It is important to set this just once to the right frequency, because other PWM channels might use the same PWM controller, and attempts to reset the PWM frequency might fail. The *pinMode()* function doesn't have an argument for providing the

update frequency, so use the *analogWrite()* function, instead. You can review using the PWM in *Controlling a Servo Motor*.

⑥ *updateMotors()* is the test function for the motors and is defined after all the setup and initialization code. The code calls this function every two seconds using the *setTimeout()* JavaScript function. The first call is used to prime the loop.

⑦ The call to *console.log()* was initially here to observe the state transitions in the debug console, but it was replaced with the *updateLEDs()* call. Using the *USER* LEDs makes it possible to note the state transitions without having visibility of the debug console. *updateLEDs()* is defined later.

⑧ The *M1_set()* and *M2_set()* functions are defined near the bottom and do the work of configuring the motor drivers into a particular state. They take a single argument of *speed*, as defined between *-1* (maximum reverse), *0* (stop), and *1* (maximum forward).

⑨ Perform simple bounds checking to ensure that speed values are between *-1* and *1*.

⑩ The *analogWrite()* call uses the absolute value of *speed*, making any negative numbers a positive magnitude.

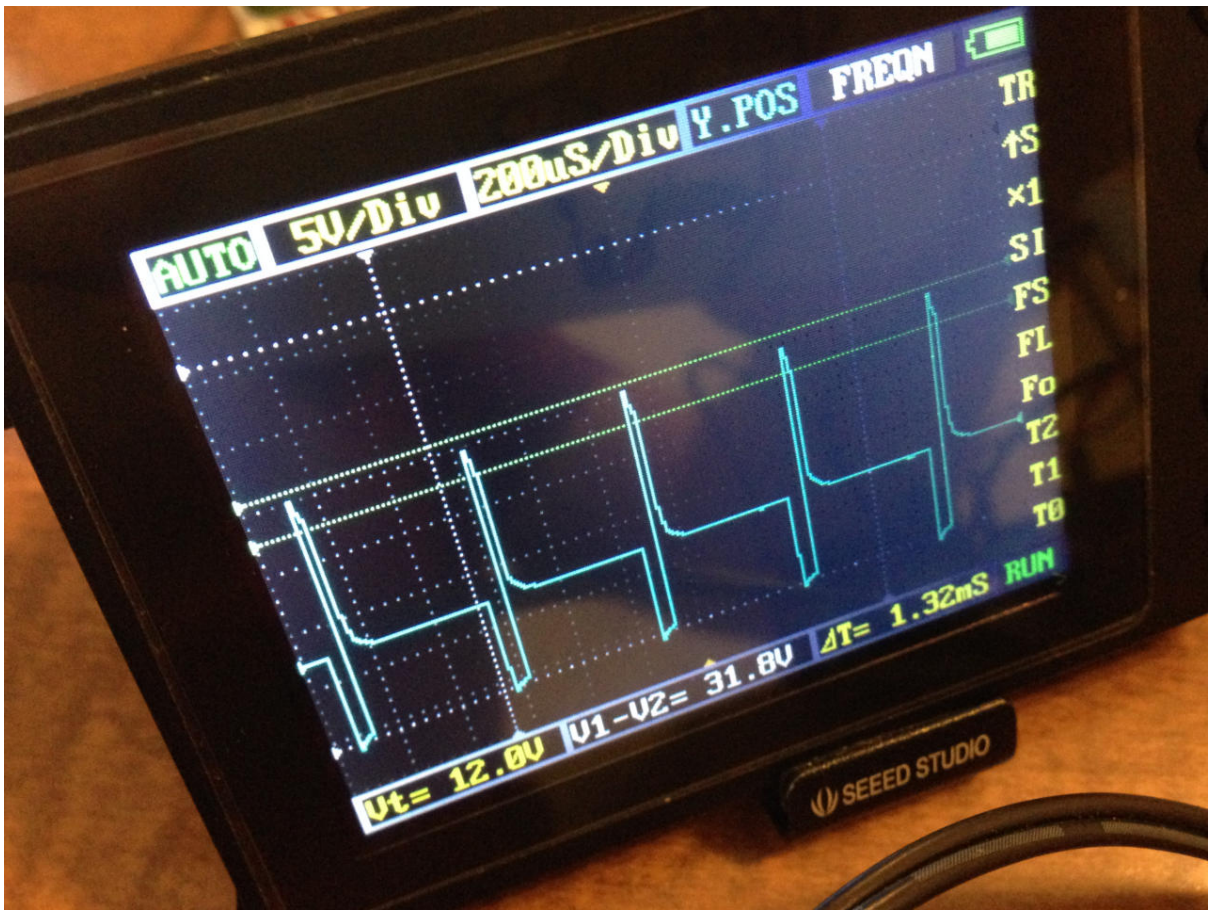


Fig. 15.78: quickBot motor test showing kickback

Using the solution in *Basics*, you can untether from your coding station to test your design at your lab workbench, as shown in *quickBot motor test code under scope*.

SparkFun provides a useful guide to using an oscilloscope. You might want to check it out if you've never used an oscilloscope before. Looking at the stimulus you'll generate *before* you connect up your hardware will help you avoid surprises.

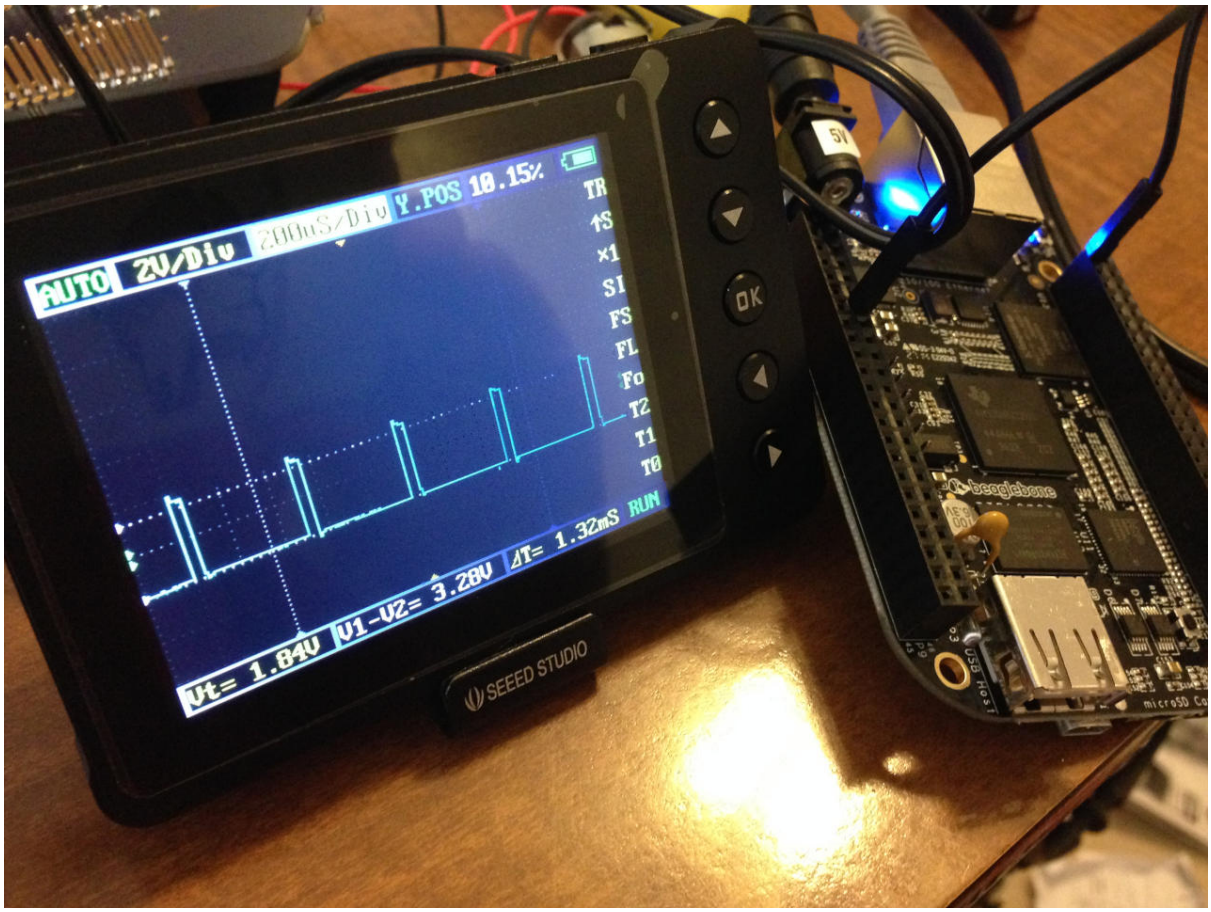


Fig. 15.79: quickBot motor test code under scope

Laying Out Your Cape PCB

Problem You've generated a diagram and schematic for your circuit and verified that they are correct. How do you create a PCB?

Solution If you've been using Fritzing, all you need to do is click the PCB tab, and there's your board. Well, almost. Much like the schematic view shown in *Creating a Prototype Schematic*, you need to do some layout work before it's actually usable. I just moved the components around until they seemed to be grouped logically and then clicked the Autoroute button. After a minute or two of trying various layouts, Fritzing picked the one it determined to be the best. *Simple robot PCB* shows the results.

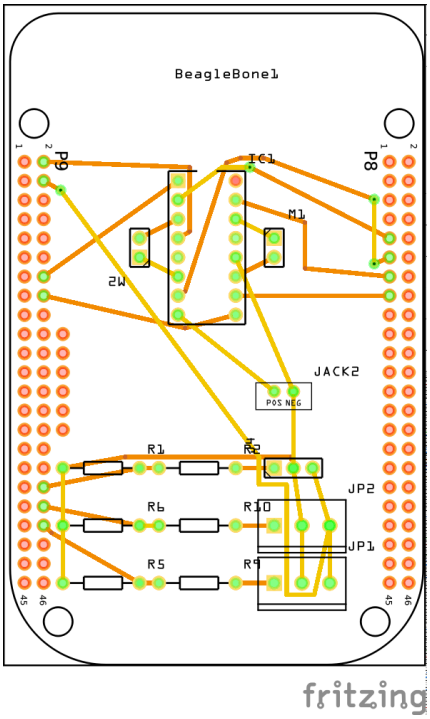


Fig. 15.80: Simple robot PCB

The Fritzing pre-fab web page has a few helpful hints, including checking the widths of all your traces and cleaning up any questionable routing created by the autorouter.

The PCB in *Simple robot PCB* is a two-sided board. One color (or shade of gray in the printed book) represents traces on one side of the board, and the other color (or shade of gray) is the other side. Sometimes, you'll see a trace come to a small circle and then change colors. This is where it is switching sides of the board through what's called a *via*. One of the goals of PCB design is to minimize the number of vias.

Simple robot PCB wasn't my first try or my last. My approach was to see what was needed to hook where and move the components around to make it easier for the autorouter to carry out its job.

Note: There are entire books and websites dedicated to creating PCB layouts. Look around and see what you can find. SparkFun's guide to making PCBs is particularly useful.

Customizing the Board Outline

One challenge that slipped my first pass review was the board outline. The part we installed in *Fritzing tips* is meant to represent BeagleBone Black, not a cape, so the outline doesn't have the notch cut out of it for the Ethernet connector.

The Fritzing custom PCB outline page describes how to create and use a custom board outline. Although it is possible to use a drawing tool like Inkscape, I chose to use the SVG path command directly to create *Outline SVG for BeagleBone cape (beaglebone_cape_boardoutline.svg)*.

Listing 15.66: Outline SVG for BeagleBone cape (beaglebone_cape_boardoutline.svg)

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
 width="306"  height="193.5"> <!-- ① -->
 <g id="board"> <!-- ② -->
  <path fill="#338040" id="boardoutline" d="M 22.5,0 l 0,56 L 72,56
   q 5,0 5,5 l 0,53.5 q 0,5 -5,5 L 0,119.5 L 0,171 Q 0,193.5 22.5,193.5
   l 238.5,0 c 24.85281,0 45,-20.14719 45,-45 L 306,45
   C 306,20.14719 285.85281,0 261,0 z"/> <!-- ③ -->
 </g>
</svg>
```

① This is a standard SVG header. The width and height are set based on the BeagleBone outline provided in the Adafruit library.

② Fritzing requires the element to be within a layer called *board*

③ Fritzing requires the color to be *#338040* and the layer to be called *boardoutline*. The units end up being 1/90 of an inch. That is, take the numbers in the SVG code and divide by 90 to get the numbers from the System Reference Manual.

The measurements are taken from the *BeagleBone Black Mechanical* section of the *BeagleBone Black System Reference Manual*, as shown in *Cape dimensions*.

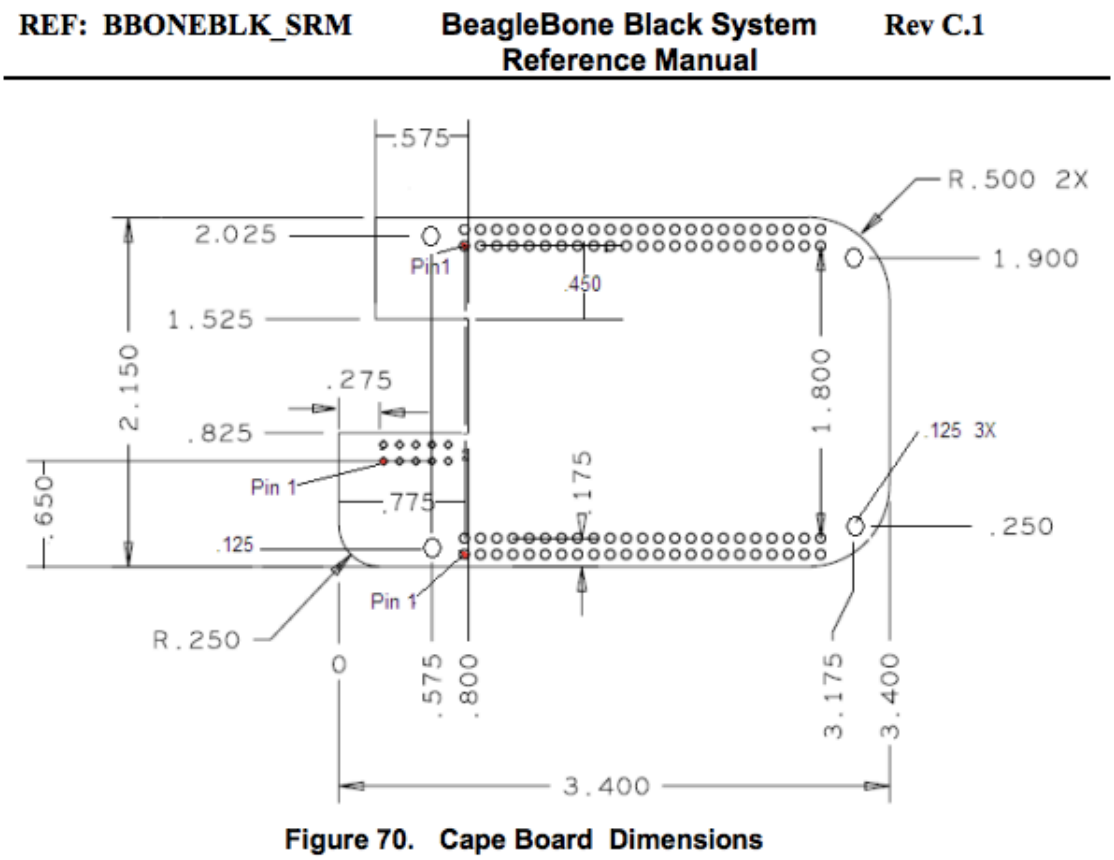


Figure 70. Cape Board Dimensions

Fig. 15.81: Cape dimensions

You can observe the rendered output of *Outline SVG for BeagleBone cape (beaglebone_cape_boardoutline.svg)* quickly by opening the file in a web browser, as shown in *Rendered cape outline in Chrome*.

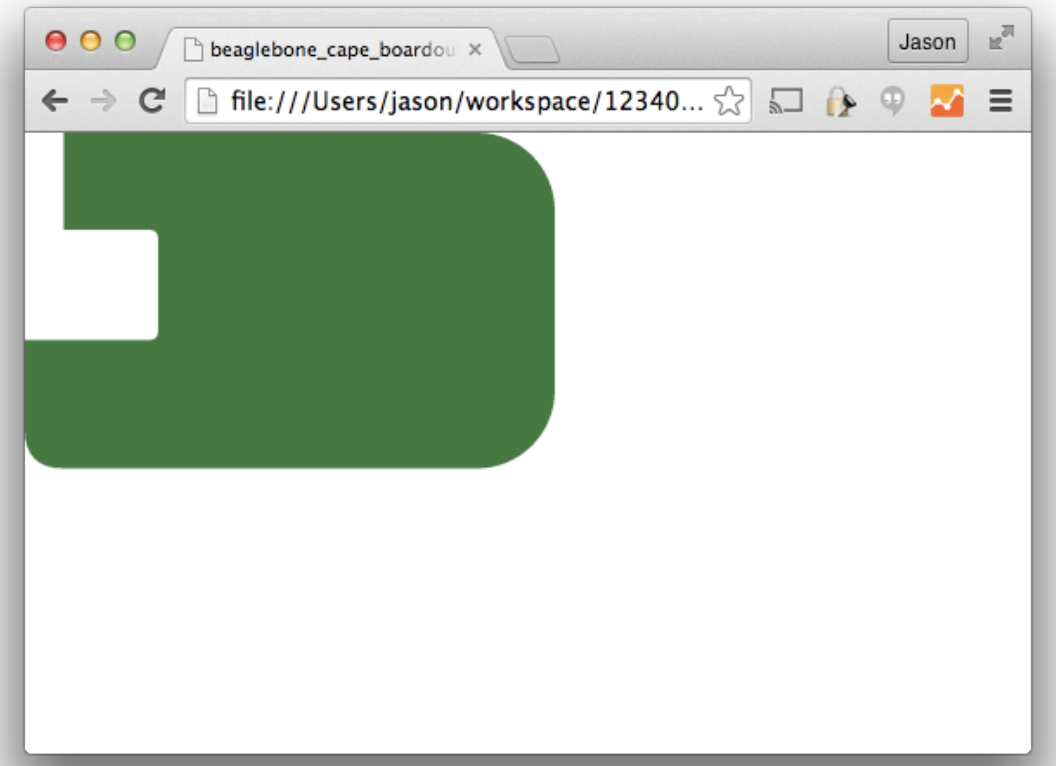


Fig. 15.82: Rendered cape outline in Chrome

Fritzing tips

After you have the SVG outline, you'll need to select the PCB in Fritzing and select a custom shape in the Inspector box. Begin with the original background, as shown in *PCB with original board, without notch for Ethernet connector*.

Hide all but the Board Layer (*PCB with all but the Board Layer hidden*).

Select the PCB1 object and then, in the Inspector pane, scroll down to the "load image file" button (*Clicking :load image file: with PCB1 selected*).

Navigate to the *beaglebone_cape_boardoutline.svg* file created in *Outline SVG for BeagleBone cape (beaglebone_cape_boardoutline.svg)*, as shown in *Selecting the .svg file*.

Turn on the other layers and line up the Board Layer with the rest of the PCB, as shown in *PCB Inspector*.

Now, you can save your file and send it off to be made, as described in *Producing a Prototype*.

PCB Design Alternatives

There are other free PCB design programs. Here are a few.

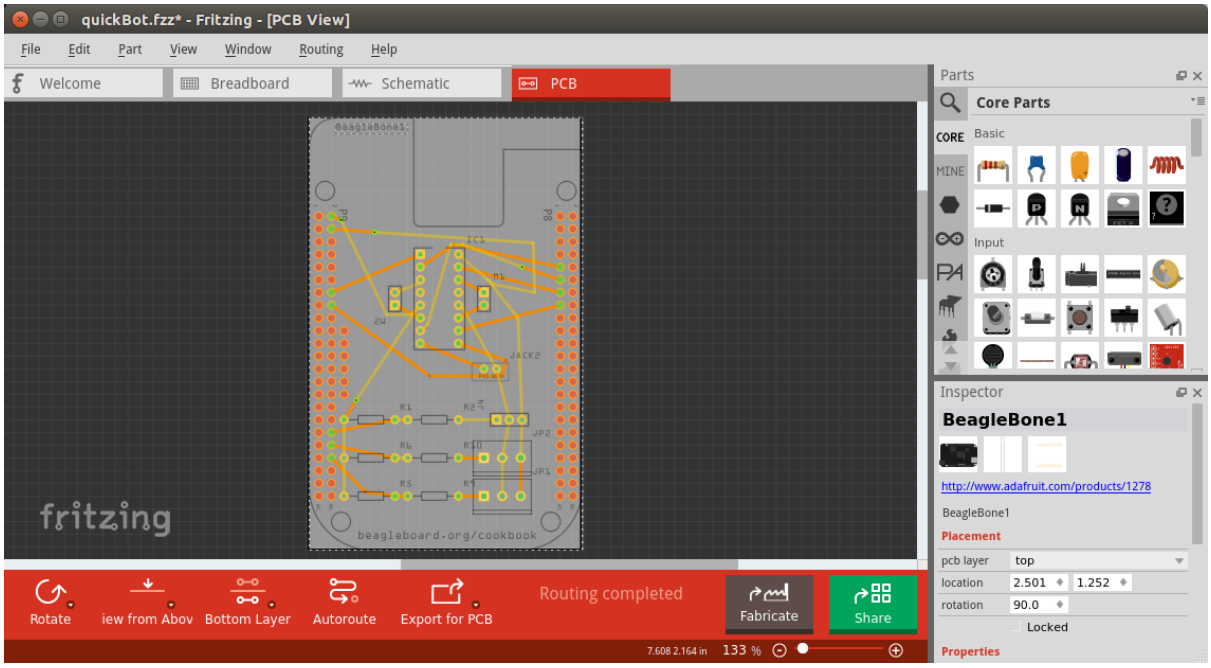


Fig. 15.83: PCB with original board, without notch for Ethernet connector

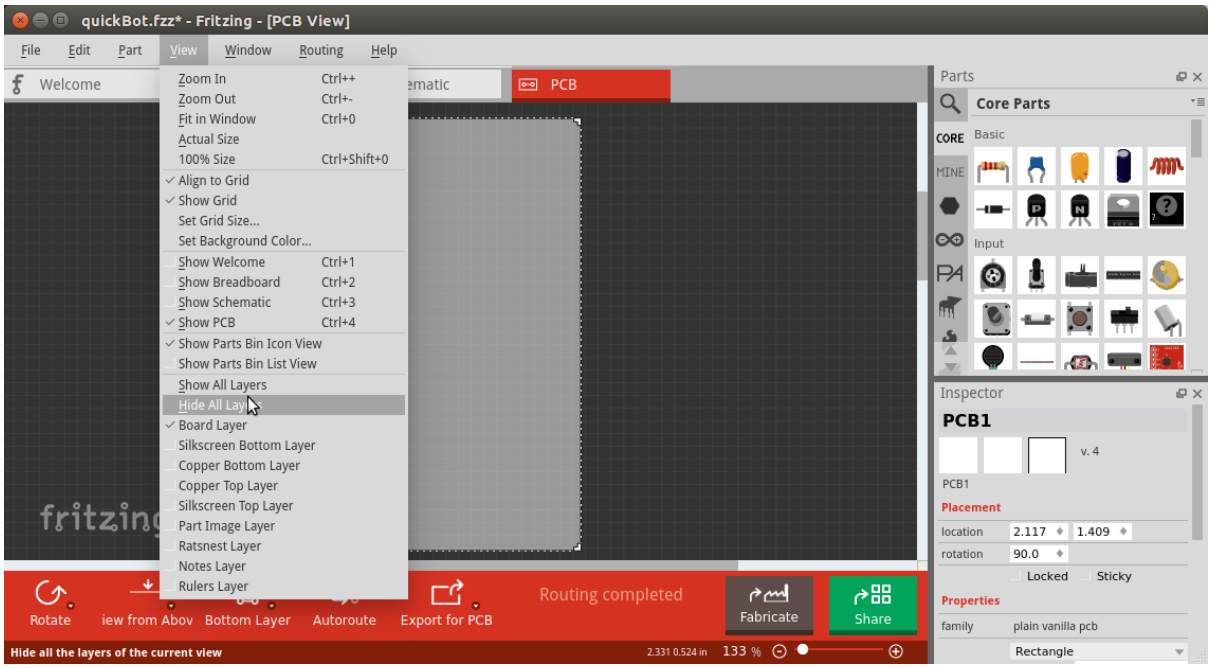


Fig. 15.84: PCB with all but the Board Layer hidden

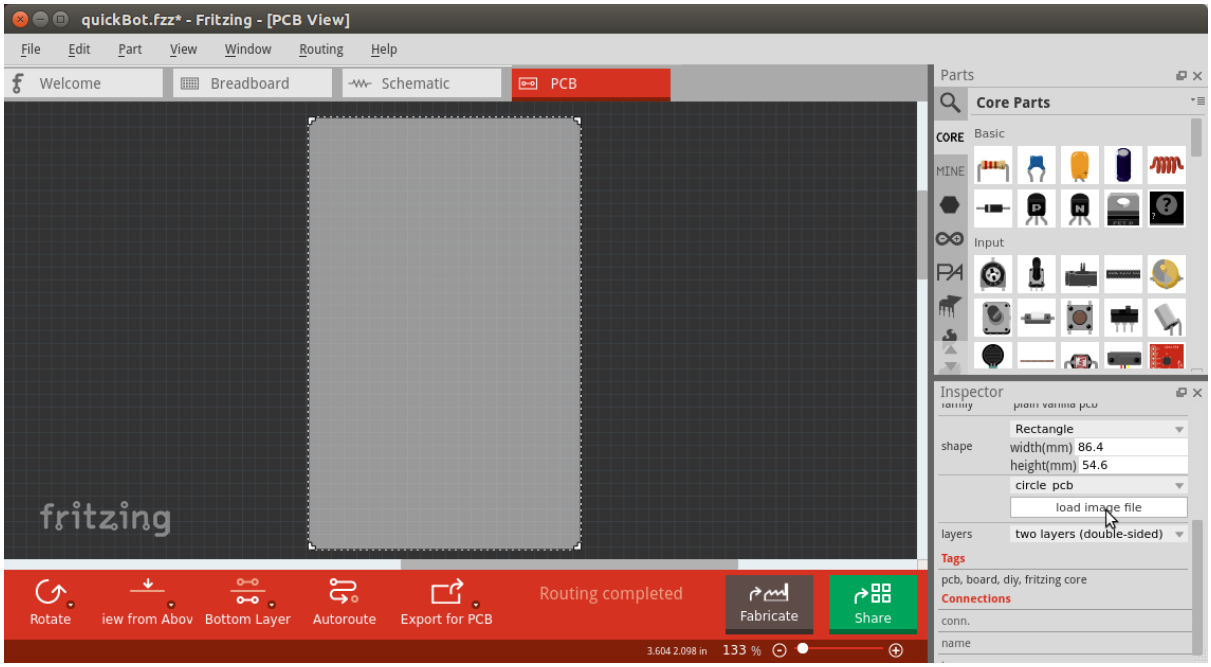


Fig. 15.85: Clicking :load image file: with PCB1 selected

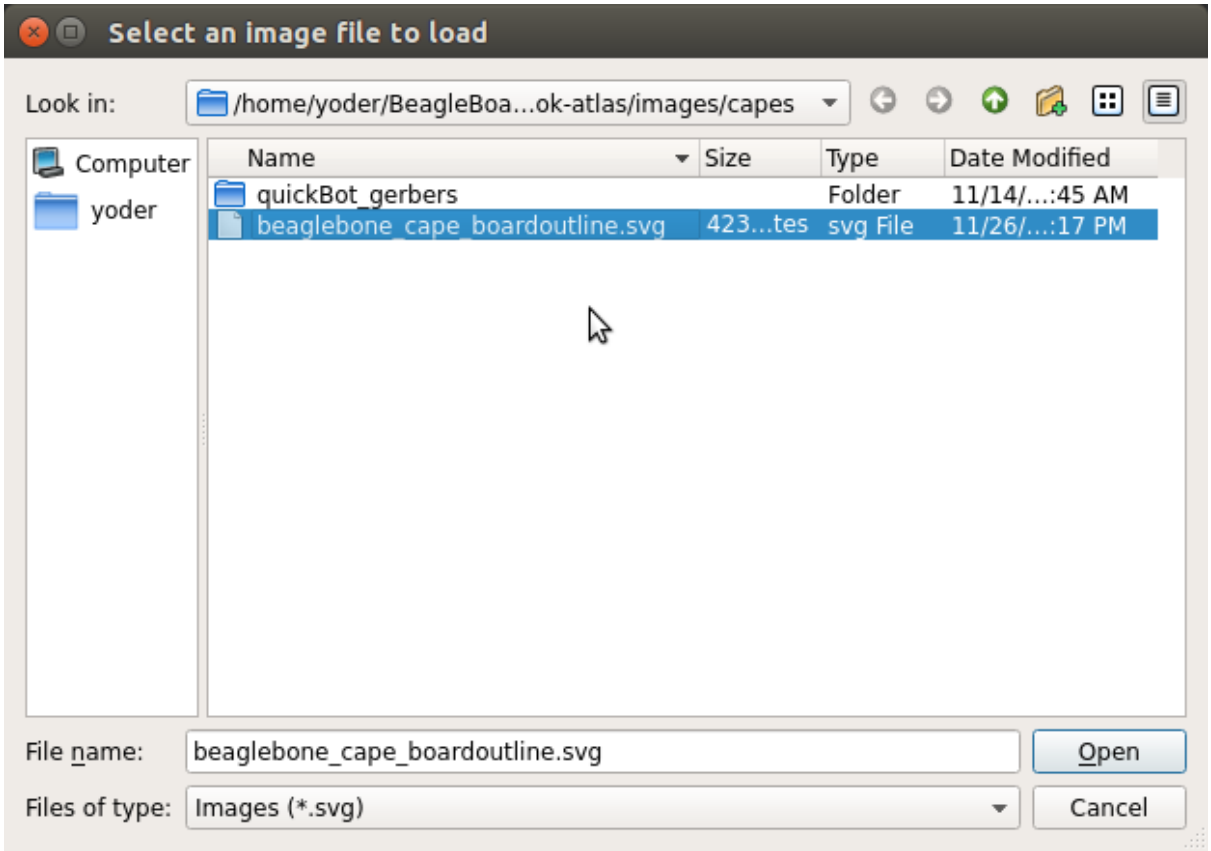


Fig. 15.86: Selecting the .svg file

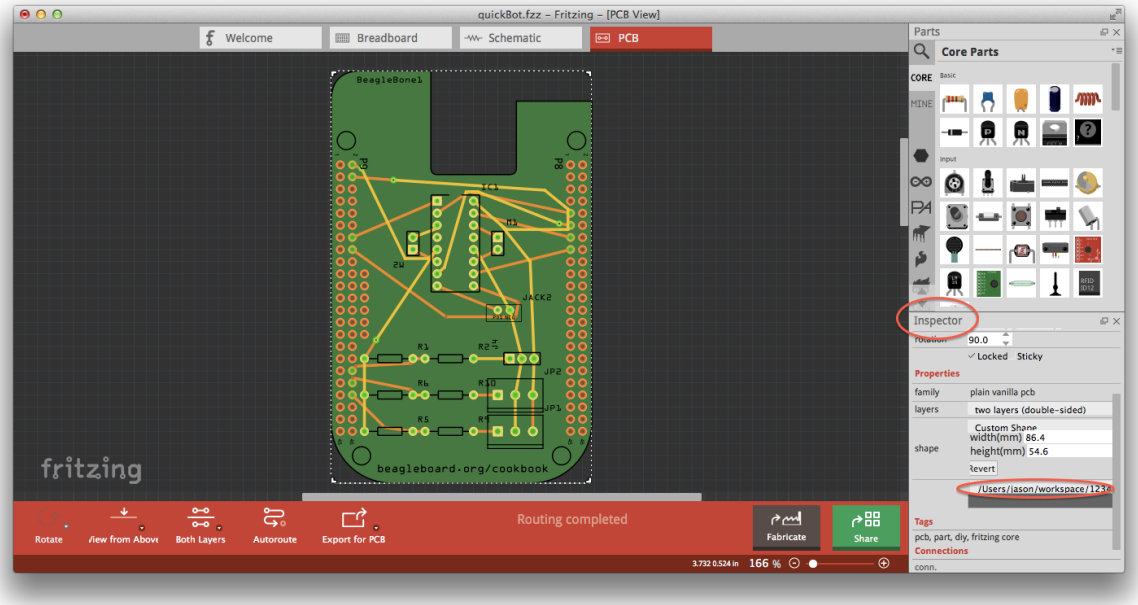


Fig. 15.87: PCB Inspector

EAGLE Eagle PCB and DesignSpark PCB are two popular design programs. Many capes (and other PCBs) are designed with Eagle PCB, and the files are available. For example, the MiniDisplay cape has the schematic shown in *Schematic for the MiniDisplay cape* and PCB shown in *PCB for MiniDisplay cape*.

Note: #TODO#: The MiniDisplay cape is not currently available, so this example should be updated.

A good starting point is to take the PCB layout for the MiniDisplay and edit it for your project. The connectors for **P8** and **P9** are already in place and ready to go.

Eagle PCB is a powerful system with many good tutorials online. The free version runs on Windows, Mac, and Linux, but it has three limitations:

- The usable board area is limited to 100 x 80 mm (4 x 3.2 inches).
- You can use only two signal layers (Top and Bottom).
- The schematic editor can create only one sheet.

You can install Eagle PCB on your Linux host by using the following command:

```
host$ sudo apt install eagle
Reading package lists... Done
Building dependency tree
Reading state information... Done
...
Setting up eagle (6.5.0-1) ...
Processing triggers for libc-bin (2.19-0ubuntu6.4) ...
host$ eagle
```

You'll see the startup screen shown in *Eagle PCB startup screen*.

Click "Run as Freeware." When my Eagle started, it said it needed to be updated. To update on Linux, follow the link provided by Eagle and download *eagle-lin-7.2.0.run* (or whatever version is current.). Then run the following commands:

```
host$ chmod +x eagle-lin-7.2.0.run
host$ ./eagle-lin-7.2.0.run
```

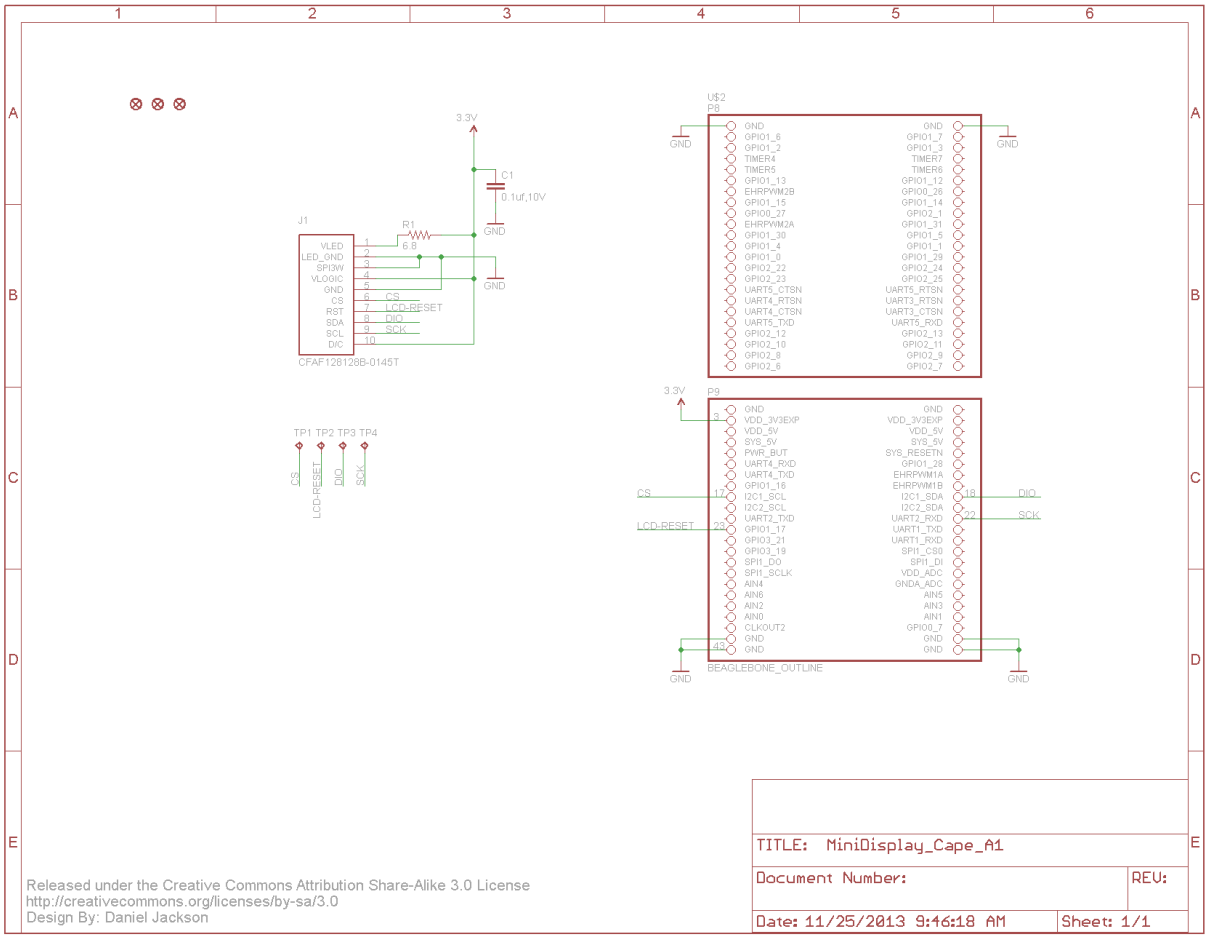


Fig. 15.88: Schematic for the MiniDisplay cape

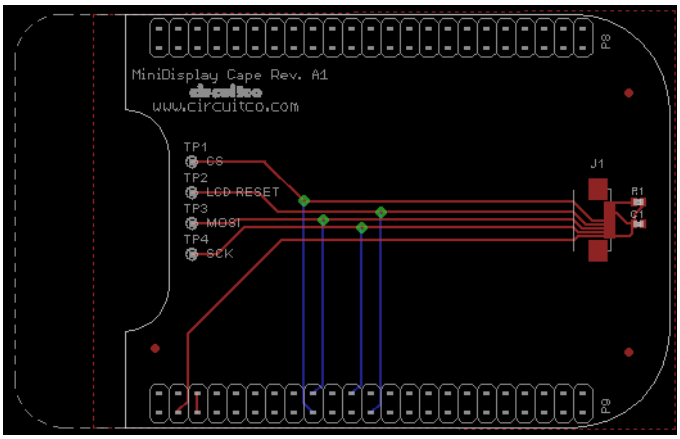


Fig. 15.89: PCB for MiniDisplay cape

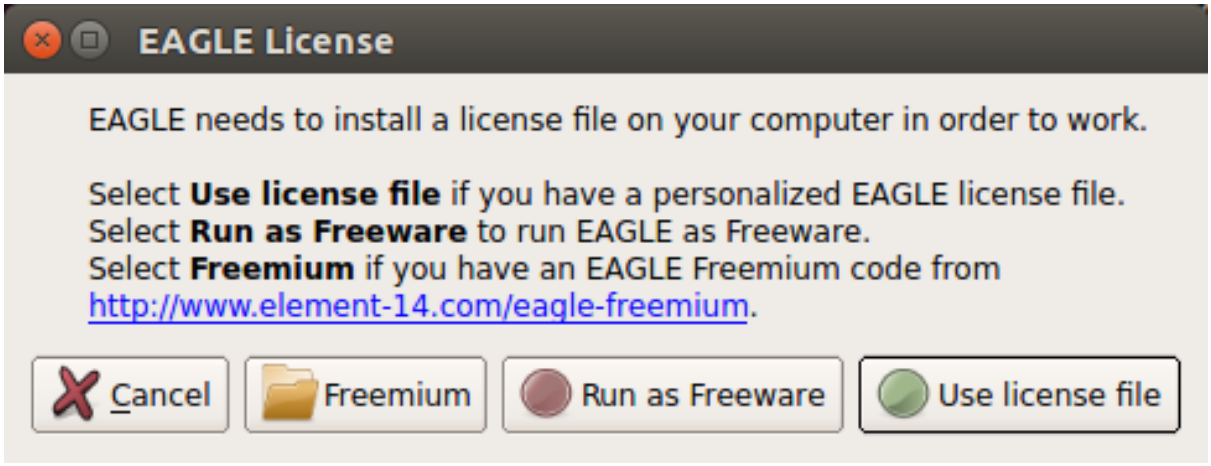


Fig. 15.90: Eagle PCB startup screen

A series of screens will appear. Click Next. When you see a screen that looks like *The Eagle installation destination directory*, note the Destination Directory.

Continue clicking Next until it's installed. Then run the following commands (where *~/eagle-7.2.0* is the path you noted in *The Eagle installation destination directory*):

```
host$ cd /usr/bin
host$ sudo rm eagle
host$ sudo ln -s ~/eagle-7.2.0/bin/eagle .
host$ cd
host$ eagle
```

The *ls* command links *eagle* in */usr/bin*, so you can run +eagle+ from any directory. After *eagle* starts, you'll see the start screen shown in *The Eagle start screen*.

Ensure that the correct version number appears.

If you are moving a design from Fritzing to Eagle, see *Migrating a Fritzing Schematic to Another Tool* for tips on converting from one to the other.

DesignSpark PCB The free DesignSpark doesn't have the same limitations as Eagle PCB, but it runs only on Windows. Also, it doesn't seem to have the following of Eagle at this time.

Upverter In addition to free solutions you run on your desktop, you can also work with a browser-based tool called Upverter. With Upverter, you can collaborate easily, editing your designs from anywhere on the Internet. It also provides many conversion options and a PCB fabrication service.

Note: Don't confuse Upverter with Upconverter (*Migrating a Fritzing Schematic to Another Tool*). Though their names differ by only three letters, they differ greatly in what they do.

Kicad Unlike the previously mentioned free (no-cost) solutions, Kicad is open source and provides some features beyond those of Fritzing. Notably, CircuitHub site (discussed in *Putting Your Cape Design into Production*) provides support for uploading Kicad designs.

Migrating a Fritzing Schematic to Another Tool

Problem You created your schematic in Fritzing, but it doesn't integrate with everything you need. How can you move the schematic to another tool?



Fig. 15.91: The Eagle installation destination directory

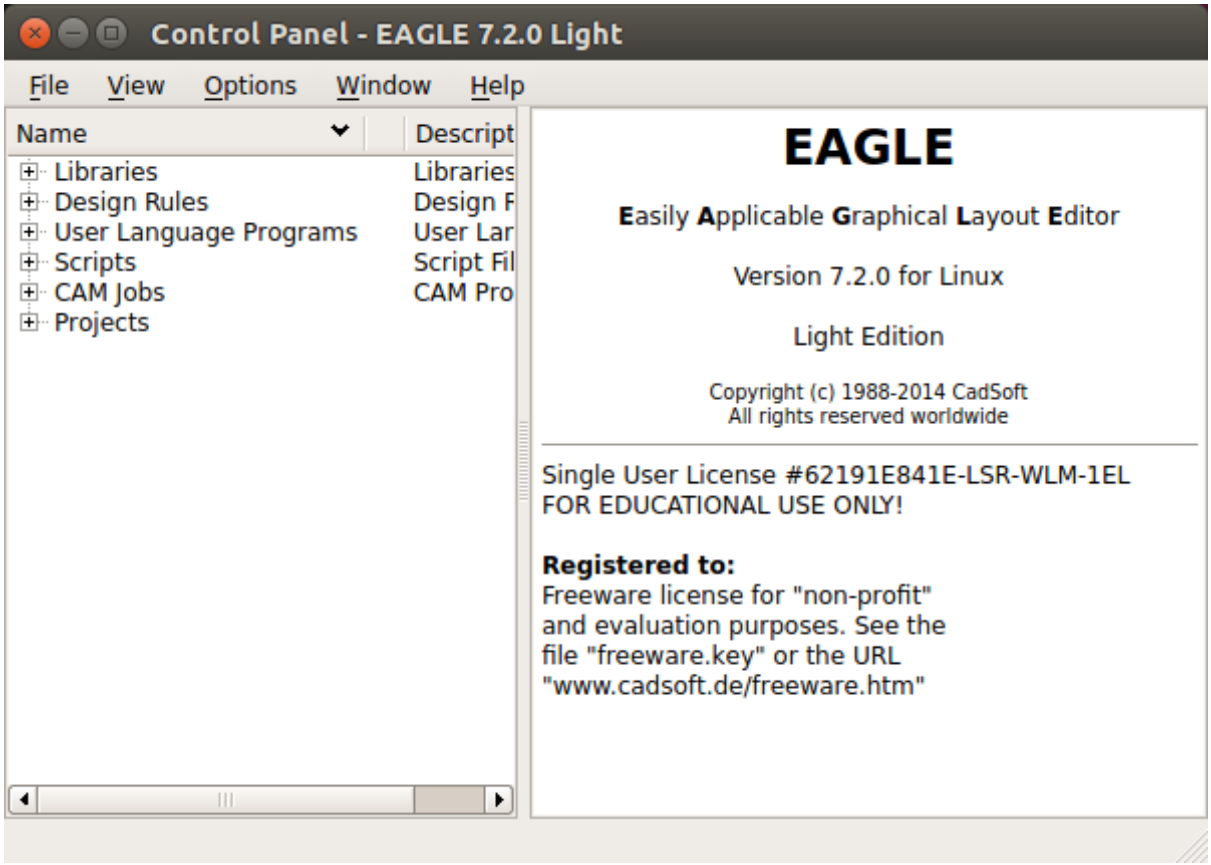


Fig. 15.92: The Eagle start screen

Solution Use the Upverter schematic-file-converter Python script. For example, suppose that you want to convert the Fritzing file for the diagram shown in *A simple robot controller diagram (quickBot.fzz)*. First, install Upverter.

I found it necessary to install +libfreetype6+ and +freetype-py+ onto my system, but you might not need this first step:

```
host$ sudo apt install libfreetype6
Reading package lists... Done
Building dependency tree
Reading state information... Done
libfreetype6 is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 154 not upgraded.
host$ sudo pip install freetype-py
Downloading/unpacking freetype-py
Running setup.py egg_info for package freetype-py

Installing collected packages: freetype-py
Running setup.py install for freetype-py

Successfully installed freetype-py
Cleaning up...
```

Note: All these commands are being run on the Linux-based host computer, as shown by the **host$** prompt. Log in as a normal user, not +root+.

Now, install the `schematic-file-converter` tool:

```
host$ git clone git@github.com:upverter/schematic-file-converter.git
Cloning into 'schematic-file-converter'...
remote: Counting objects: 22251, done.
remote: Total 22251 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (22251/22251), 39.45 MiB | 7.28 MiB/s, done.
Resolving deltas: 100% (14761/14761), done.
Checking connectivity... done.
Checking out files: 100% (16880/16880), done.
host$ cd schematic-file-converter
host$ sudo python setup.py install
.
.
.
Extracting python_upconvert-0.8.9-py2.7.egg to \
    /usr/local/lib/python2.7/dist-packages
Adding python-upconvert 0.8.9 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/python_upconvert-0.8.9-py2.
↪7.egg
Processing dependencies for python-upconvert==0.8.9
Finished processing dependencies for python-upconvert==0.8.9
host$ cd ..
host$ python -m upconvert.upconverter -h
usage: upconverter.py [-h] [-i INPUT] [-f TYPE] [-o OUTPUT] [-t TYPE]
                      [-s SYMDIRS [SYMDIRS ...]] [--unsupported]
                      [--raise-errors] [--profile] [-v] [--formats]

optional arguments:
-h, --help            show this help message and exit
-i INPUT, --input INPUT
                        read INPUT file in
-f TYPE, --from TYPE  read input file as TYPE
-o OUTPUT, --output OUTPUT
                        write OUTPUT file out
-t TYPE, --to TYPE    write output file as TYPE
-s SYMDIRS [SYMDIRS ...], --sym-dirs SYMDIRS [SYMDIRS ...]
                        specify SYMDIRS to search for .sym files (for gEDA
                        only)
--unsupported         run with an unsupported python version
--raise-errors        show tracebacks for parsing and writing errors
--profile             collect profiling information
-v, --version         print version information and quit
--formats             print supported formats and quit
```

At the time of this writing, Upverter supports the following file types:

File type	Support
openjson	i/o
kicad	i/o
geda	i/o
eagle	i/o
eaglexml	i/o
fritzing	in only schematic only
gerber	i/o
specctra	i/o
image	out only
ncdrill	out only
bom (csv)	out only
netlist (csv)	out only

After Upverter is installed, run the file (`quickBot.fzz`) that generated *A simple robot controller diagram (quickBot.fzz)* through Upverter:

```
host$ python -m upconvert.upconverter -i quickBot.fzz \
-f fritzing -o quickBot-eaglexml.sch -t eaglexml --unsupported
WARNING: RUNNING UNSUPPORTED VERSION OF PYTHON (2.7 > 2.6)
DEBUG:main:parsing quickBot.fzz in format fritzing
host$ ls -l
total 188
-rw-rw-r-- 1 ubuntu  63914 Nov 25 19:47 quickBot-eaglexml.sch
-rw-r--r-- 1 ubuntu 122193 Nov 25 19:43 quickBot.fzz
drwxrwxr-x 9 ubuntu   4096 Nov 25 19:42 schematic-file-converter
```

Output of Upverter conversion shows the output of the conversion.

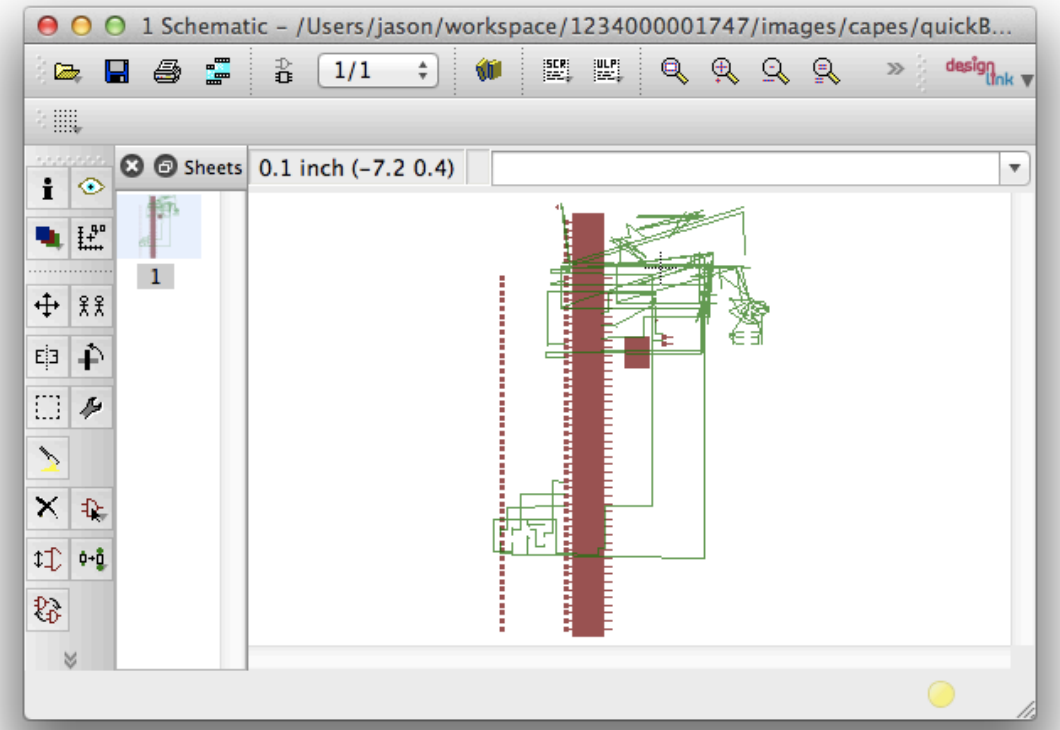


Fig. 15.93: Output of Upverter conversion

No one said it would be pretty!

I found that Eagle was more generous at reading in the **eaglexml** format than the **eagle** format. This also made it easier to hand-edit any translation issues.

Producing a Prototype

Problem You have your PCB all designed. How do you get it made?

Solution To make this recipe, you will need:

- A completed design
- Soldering iron
- Oscilloscope

- Multimeter
- Your other components

Upload your design to OSH Park and order a few boards. *The OSH Park QuickBot Cape shared project page* shows a resulting shared project page for the quickBot cape created in *Laying Out Your Cape PCB*. We'll proceed to break down how this design was uploaded and shared to enable ordering fabricated PCBs.

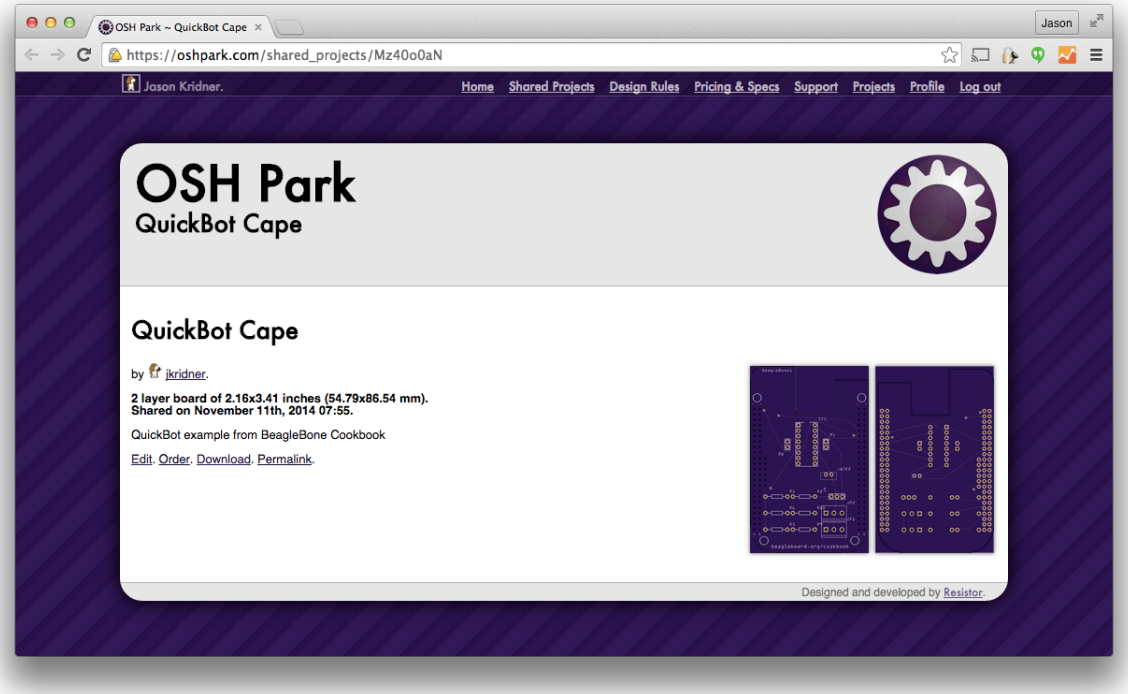


Fig. 15.94: The OSH Park QuickBot Cape shared project page

Within Fritzing, click the menu next to "Export for PCB" and choose "Extended Gerber," as shown in *Choosing "Extended Gerber" in Fritzing*. You'll need to choose a directory in which to save them and then compress them all into a Zip file. The WikiHow article on creating Zip files might be helpful if you aren't very experienced at making these.

Things on the OSH Park website are reasonably self-explanatory. You'll need to create an account and upload the Zip file containing the Gerber files you created. If you are a cautious person, you might choose to examine the Gerber files with a Gerber file viewer first. The Fritzing fabrication FAQ offers several suggestions, including gerbv for Windows and Linux users.

When your upload is complete, you'll be given a quote, shown images for review, and presented with options for accepting and ordering. After you have accepted the design, your list of accepted designs will also include the option of enabling sharing of your designs so that others can order a PCB, as well. If you are looking to make some money on your design, you'll want to go another route, like the one described in *Putting Your Cape Design into Production*. *QuickBot PCB* shows the resulting PCB that arrives in the mail.

Now is a good time to ensure that you have all of your components and a soldering station set up as in *Moving from a Breadboard to a Protoboard*, as well as an oscilloscope, as used in *Verifying Your Cape Design*.

When you get your board, it is often informative to "buzz out" a few connections by using a multimeter. If you've never used a multimeter before, the SparkFun or Adafruit tutorials might be helpful. Set your meter to continuity testing mode and probe between points where the headers are and where they should be connecting to your components. This would be more difficult and less accurate after you solder down your components, so it is a good idea to keep a bare board around just for this purpose.

You'll also want to examine your board mechanically before soldering parts down. You don't want to waste components on a PCB that might need to be altered or replaced.

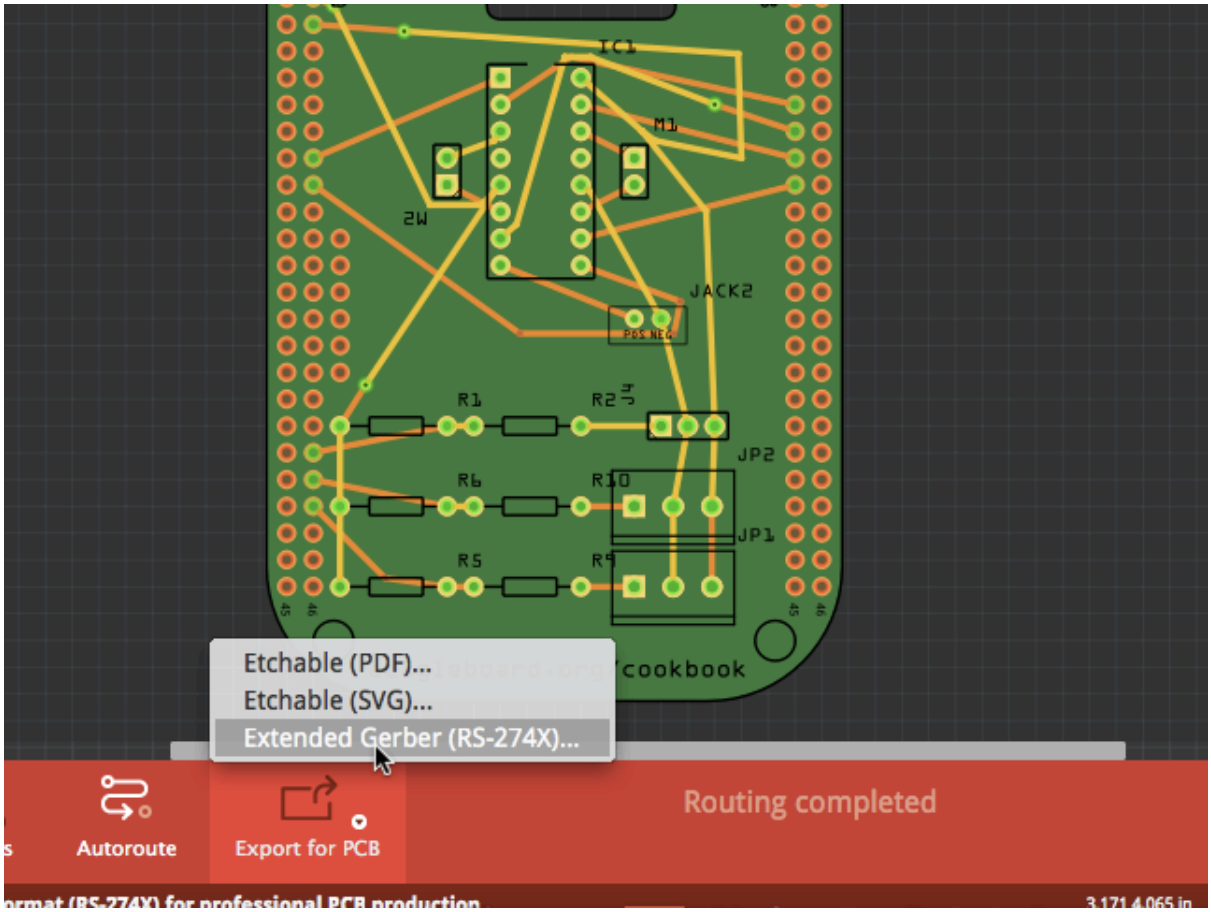


Fig. 15.95: Choosing “Extended Gerber” in Fritzing

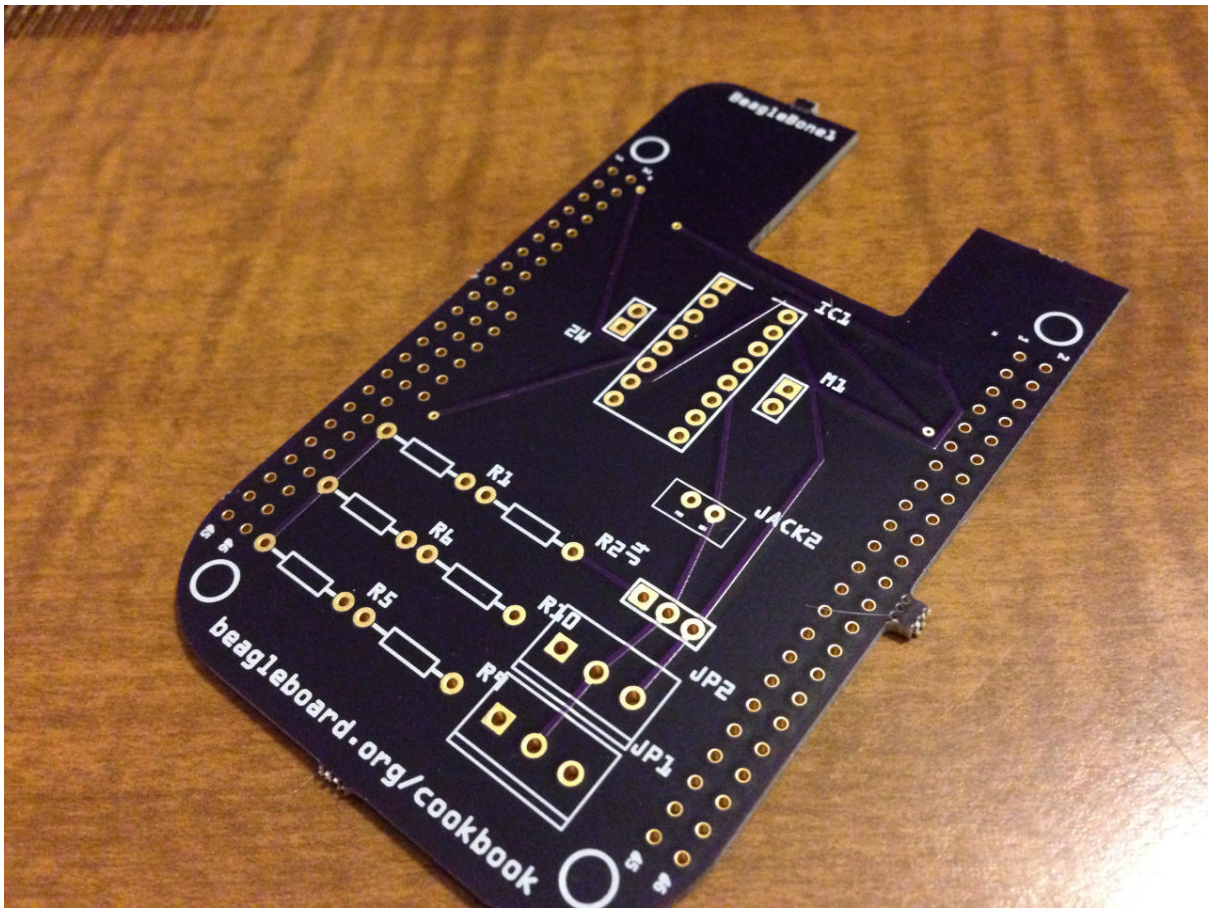


Fig. 15.96: QuickBot PCB

When you begin assembling your board, it is advisable to assemble it in functional subsections, if possible, to help narrow down any potential issues. *QuickBot motors under test* shows the motor portion wired up and running the test in *Testing the quickBot motors interface (quickBot_motor_test.js)*.

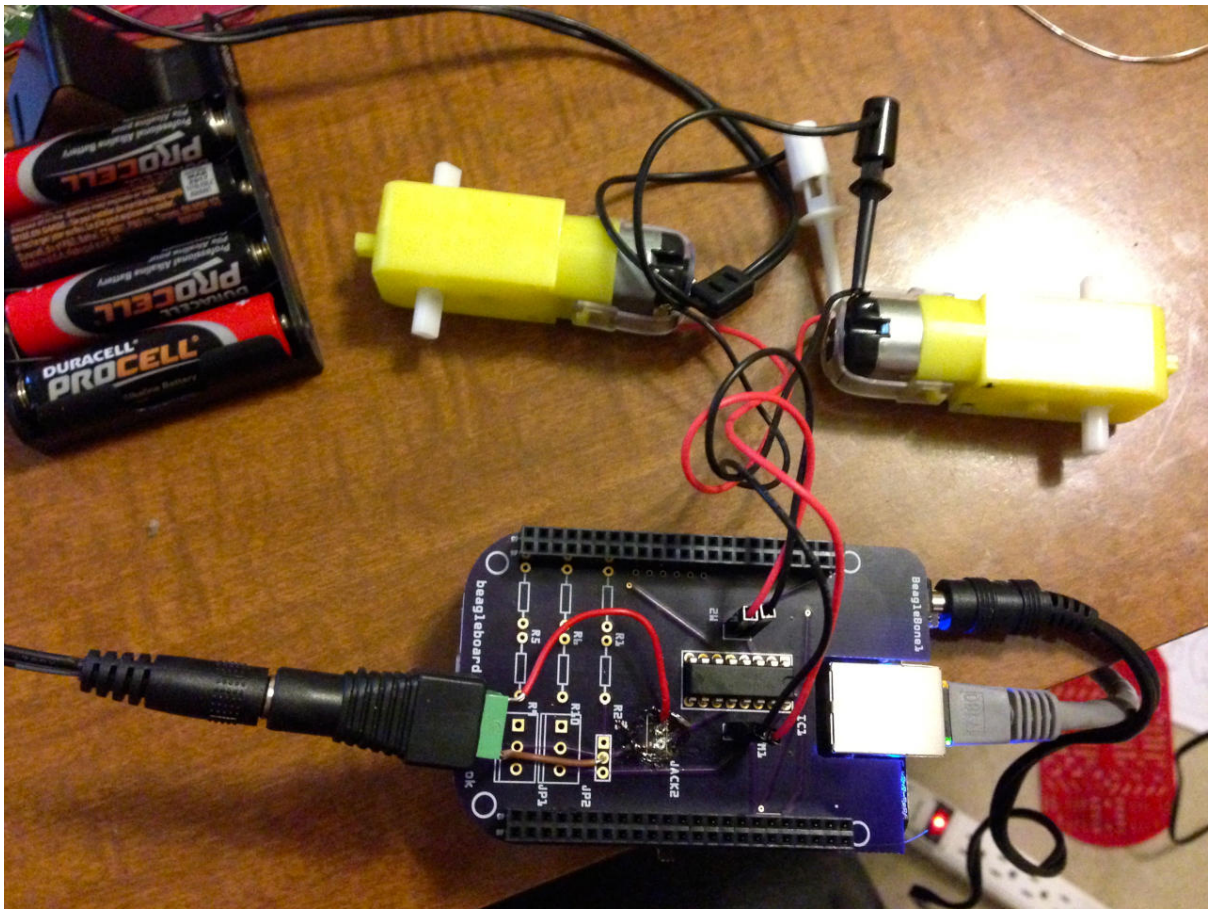


Fig. 15.97: QuickBot motors under test

Continue assembling and testing your board until you are happy. If you find issues, you might choose to cut traces and use point-to-point wiring to resolve your issues before placing an order for a new PCB. Better right the second time than the third!

Creating Contents for Your Cape Configuration EEPROM

Problem Your cape is ready to go, and you want it to automatically initialize when the Bone boots up.

Solution Complete capes have an I^2C EEPROM on board that contains configuration information that is read at boot time. Adventures in BeagleBone Cape EEPROMs gives a helpful description of two methods for programming the EEPROM. How to Roll your own BeagleBone Capes is a good four-part series on creating a cape, including how to wire and program the EEPROM.

Note: The current effort to document how to enable software for a cape is ongoing at https://docs.beagleboard.org/latest/boards/capes.

Putting Your Cape Design into Production

Problem You want to share your cape with others. How do you scale up?

Solution CircuitHub offers a great tool to get a quick quote on assembled PCBs. To make things simple, I downloaded the CircuitCo MiniDisplay Cape Eagle design materials and uploaded them to CircuitHub.

After the design is uploaded, you'll need to review the parts to verify that CircuitHub has or can order the right ones. Find the parts in the catalog by changing the text in the search box and clicking the magnifying glass. When you've found a suitable match, select it to confirm its use in your design, as shown in *CircuitHub part matching*.

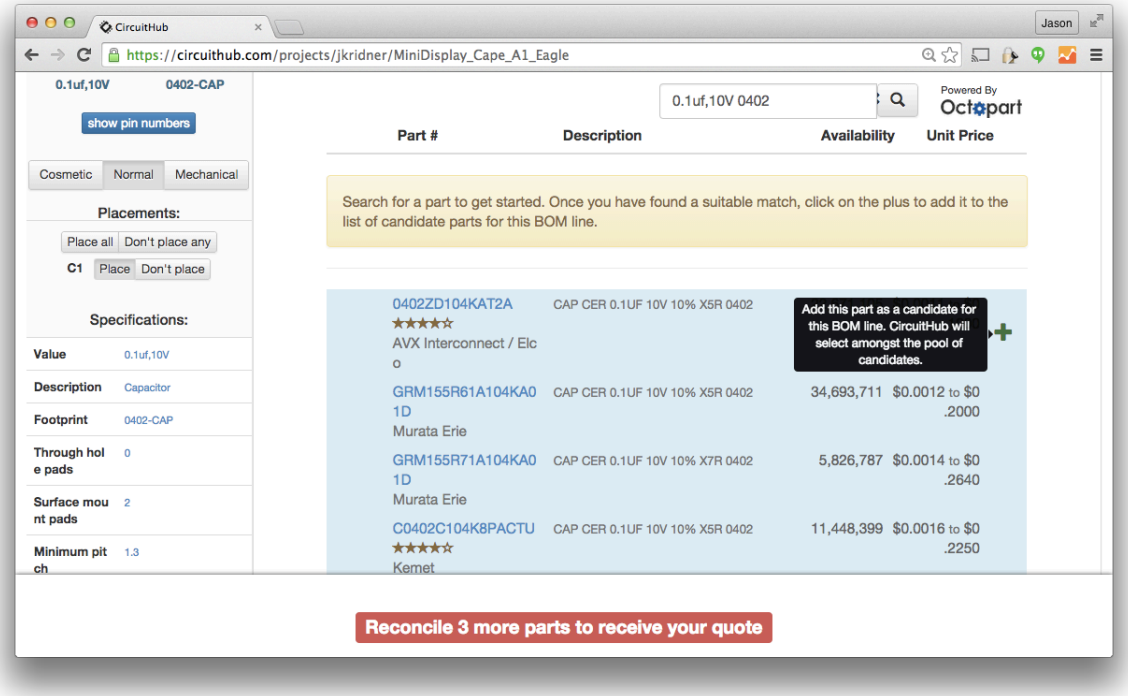


Fig. 15.98: CircuitHub part matching

When you've selected all of your parts, a quote tool appears at the bottom of the page, as shown in *CircuitHub quote generation*.

Checking out the pricing on the MiniDisplay Cape (without including the LCD itself) in *CircuitHub price examples (all prices USD)*, you can get a quick idea of how increased volume can dramatically impact the per-unit costs.

Table 15.5: CircuitHub price examples (all prices USD)

Quantity	1	10	100	1000	10,000
PCB	$208.68	$21.75	$3.30	$0.98	$0.90
Parts	$11.56	$2.55	$1.54	$1.01	$0.92
Assembly	$249.84	$30.69	$7.40	$2.79	$2.32
Per unit	$470.09	$54.99	$12.25	$4.79	$4.16
Total	$470.09	$550.00	$1,225.25	$4,796.00	$41,665.79

Checking the Crystalfontz web page for the LCD, you can find the prices for the LCDs as well, as shown in *LCD pricing (USD)*.

Table 15.6: LCD pricing (USD)

Quantity	1	10	100	1000	10,000
Per unit	$12.12	$7.30	$3.86	$2.84	$2.84
Total	$12.12	$73.00	$386.00	$2,840.00	$28,400.00

To enable more cape developers to launch their designs to the market, CircuitHub has launched a group buy campaign site. You, as a cape developer, can choose how much markup you need to be paid for your work and launch the campaign to the public. Money is only collected if and when the desired target quantity is reached,

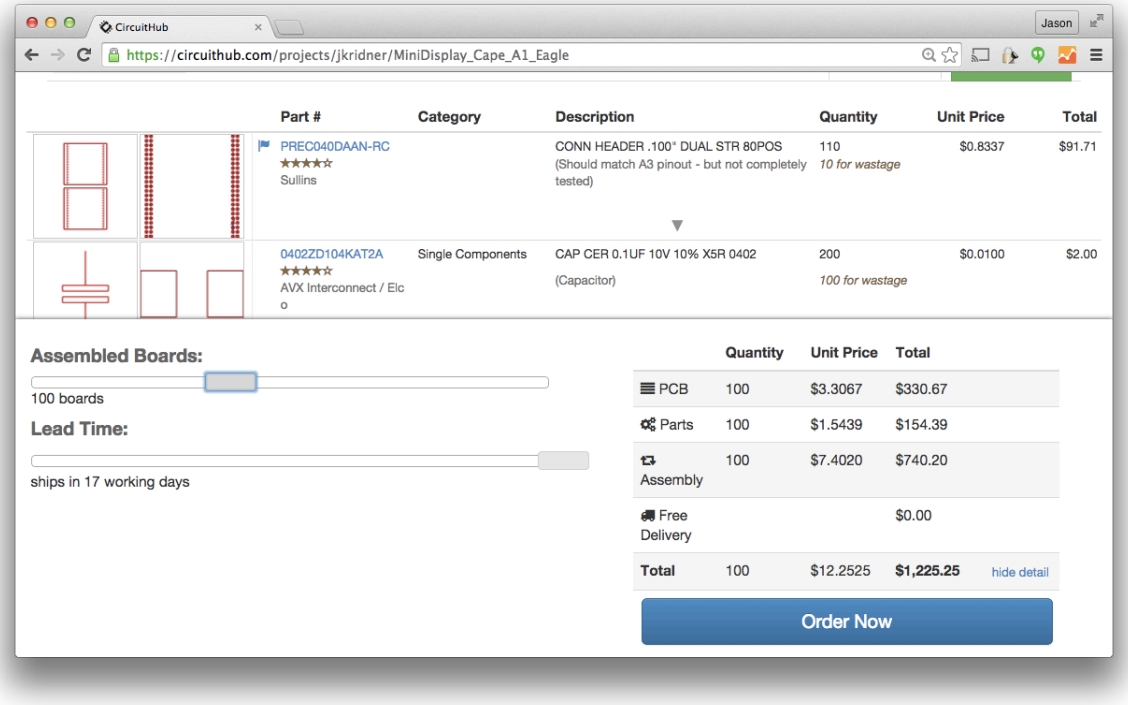


Fig. 15.99: CircuitHub quote generation

so there's no risk that the boards will cost too much to be affordable. This is a great way to cost-effectively launch your boards to market!

There's no real substitute for getting to know your contract manufacturer, its capabilities, communication style, strengths, and weaknesses. Look around your town to see if anyone is doing this type of work and see if they'll give you a tour.

Note: Don't confuse CircuitHub and CircuitCo. CircuitCo is closed.

15.1.10 Parts and Suppliers

The following tables list where you can find the parts used in this book. We have listed only one or two sources here, but you can often find a given part in many places.

Table 15.7: United States suppliers

Supplier	Website	Notes
Adafruit	http://www.adafruit.com	Good for modules and parts
Amazon	http://www.amazon.com/	Carries everything
Digikey	http://www.digikey.com/	Wide range of components
MakerShed	http://www.makershed.com/	Good for modules, kits, and tools
SeeedStudio	https://www.seeedstudio.com/SBC-Beaglebone-Original-c-2031.html?	Low-cost modules
SparkFun	http://www.sparkfun.com	Good for modules and parts

Table 15.8: Other suppliers

Supplier	Website	Notes
Element14	http://element14.com/BeagleBone	World-wide BeagleBoard.org-compliant clone of BeagleBone Black, carries many accessories

Prototyping Equipment

Many of the hardware projects in this book use jumper wires and a breadboard. We prefer the preformed wires that lie flat on the board. *Jumper wires* lists places with jumper wires, and *Breadboards* shows where you can get breadboards.

Table 15.9: Jumper wires

Supplier	Website
Amazon	http://www.amazon.com/Elenco-Piece-Pre-formed-Jumper-Wire/dp/B0002H7AIG
Digikey	http://www.digikey.com/product-detail/en/TW-E012-000/438-1049-ND/643115
SparkFun	https://www.sparkfun.com/products/124

Table 15.10: Breadboards

Supplier	Website
Amazon	http://www.amazon.com/s/ref=nb_sb_noss_1?url=search-alias%3Dtoys-and-games&field-keywords=breadboards&sprefix=breadboards%2Ctoys-and-games
Digikey	https://www.digikey.com/en/products/filter/solderless-breadboards/638
SparkFun	https://www.sparkfun.com/search/results?term=breadboard
CircuitCo	https://elinux.org/BeagleBoneBreadboard (no longer manufactured, but design available)

If you want something more permanent, try Adafruit's Perma-Proto Breadboard, laid out like a breadboard.

Resistors

We use 220 , 1k, 4.7k, 10k, 20k, and 22 kΩ resistors in this book. All are 0.25 W. The easiest way to get all these, and many more, is to order SparkFun's Resistor Kit. It's a great way to be ready for future projects, because it has 500 resistors.

If you don't need an entire kit of resistors, you can order a la carte from a number of places. DigiKey has more than a quarter million through-hole resistors at good prices, but make sure you are ordering the right one.

You can find the 10 kΩ trimpot (or variable resistor) at SparkFun 10k POT or Adafruit 10k POT.

Flex resistors (sometimes called *flex sensors* or *bend sensors*) are available at SparkFun flex resistors and Adafruit flex resistors.

Transistors and Diodes

The 2N3904 is a common NPN transistor that you can get almost anywhere. Even Amazon NPN transitor has it. Adafruit NPN transitor has a nice 10-pack. SparkFun NPN transitor lets you buy them one at a time. DigiKey NPN transitor will gladly sell you 100,000.

The 1N4001 is a popular 1A diode. Buy one at SparkFun diode, 10 at Adafruit diode, or 10,000 at DigiKey diode.

Integrated Circuits

The PCA9306 is a small integrated circuit (IC) that converts voltage levels between 3.3 V and 5 V. You can get it cheaply in large quantities from DigiKey PCA9306, but it's in a very small, hard-to-use, surface-mount package. Instead, you can get it from SparkFun PCA9306 on a Breakout board, which plugs into a breadboard.

The L293D is an H-bridge IC with which you can control large loads (such as motors) in both directions. SparkFun L293D, Adafruit L293D, and DigiKey L293D all have it in a DIP package that easily plugs into a breadboard.

The ULN2003 is a 7 darlington NPN transistor IC array used to drive motors one way. You can get it from DigiKey ULN2003. A possible substitution is ULN2803 available from SparkFun ULN2003 and Adafruit ULN2003.

The TMP102 is an I^{2}C-based digital temperature sensor. You can buy them in bulk from DigiKey TMP102, but it's too small for a breadboard. SparkFun TMP102 sells it on a breakout board that works well with a breadboard.

The DS18B20 is a one-wire digital temperature sensor that looks like a three-terminal transistor. Both SparkFun DS18B20 and Adafruit DS18B20 carry it.

Opto-Electronics

LEDs are *light-emitting diodes*. LEDs come in a wide range of colors, brightnesses, and styles. You can get a basic red LED at SparkFun red LED, Adafuit red LED, and DigiKey red LED.

Many places carry bicolor LED matrices, but be sure to get one with an I^{2}C interface. Adafruit LED matrix is where I got mine.

Capes

There are a number of sources for capes for BeagleBone Black. BeagleBoard.org capes page keeps a current list.

Miscellaneous

Here are some things that don't fit in the other categories.

Table 15.11: Miscellaneous

3.3 V FTDI cable	SparkFun FTDI cable, Adafruit FTDI cable
USB WiFi adapter	Adafruit WiFi adapter
HDMI cable	SparkFun HDMI cable
Micro HDMI to HDMI cable	Adafruit HDMI to microHDMI cable
HDMI to DVI Cable	SparkFun HDMI to DVI cable
HDMI monitor	Amazon HDMI monitor
Powered USB hub	Amazon power USB hub, Adafruit power USB hub
Soldering iron	SparkFun soldering iron, Adafruit soldering iron
Oscilloscope	Adafruit oscilloscope
Multimeter	SparkFun multimeter, Adafruit multimeter
PowerSwitch Tail II	SparkFun PowerSwitch Tail II, Adafruit PowerSwitch Tail II
Servo motor	SparkFun servo motor, Adafruit servo motor
5 V power supply	SparkFun 5V power supply, Adafruit 5V power supply
3 V to 5 V motor	SparkFun 3V-5V motor, Adafruit 3V-5V motor
3 V to 5 V bipolar stepper motor	SparkFun 3V-5V bipolar stepper motor, Adafruit 3V-5V bipolar stepper motor
3 V to 5 V unipolar stepper motor	Adafruit 3V-5V unipolar stepper motor
Pushbutton switch	SparkFun pushbutton switch, Adafruit pushbutton switch
Magnetic reed switch	SparkFun magnetic reed switch
LV-MaxSonar-EZ1 Sonar Range Finder	SparkFun LV-MaxSonar-EZ1, Amazon LV-MaxSonar-EZ1
HC-SR04 Ultrsonic Range Sensor	Amazon HC-SR04
Rotary encoder	SparkFun rotary encoder, Adafruit rotary encoder
GPS receiver	SparkFun GPS, Adafruit GPS
BLE USB dongle	Adafruit BLE USB dongle
Syba SD-CM-UAUD USB Stereo Audio Adapter	Amazon USB audio adapter
Sabrent External Sound Box USB-SBCV	Amazon USB audio adapter (alt)
Vantec USB External 7.1 Channel Audio Adapter	Amazon USB audio adapter (alt2)

15.1.11 Misc

Here are bits and pieces of ideas that are being developed.

BeagleConnect Freedom

Here are some notes on how to setup and use the Connect.

First get the flasher image from: https://www.beagleboard.org/distros/beagleplay-home-assistant-webinar-demo-image

Flash the eMMC (which also loads the cc1352 with the correct firmware)

Here's Jason's demo at the 2023 EOSS: https://youtu.be/ZT9GEs3_ZYU?t=2195

```
debian@BeaglePlay:~$ sudo beagleconnect-start-gateway
[sudo] password for debian:
setting up wpanusb gateway for IEEE 802154 CHANNEL 1(906 Mhz)
RTNETLINK answers: File exists
RTNETLINK answers: Device or resource busy
PING 2001:db8::1(2001:db8::1) from fe80::212:4b00:29c4:cdee%lowpan0 lowpan0: 56 d
bytes
64 bytes from 2001:db8::1: icmp_seq=1 ttl=64 time=69.5 ms
64 bytes from 2001:db8::1: icmp_seq=2 ttl=64 time=66.2 ms
64 bytes from 2001:db8::1: icmp_seq=3 ttl=64 time=37.5 ms
64 bytes from 2001:db8::1: icmp_seq=4 ttl=64 time=70.8 ms
64 bytes from 2001:db8::1: icmp_seq=5 ttl=64 time=37.6 ms

--- 2001:db8::1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 37.528/56.326/70.823/15.411 ms
```

Fig. 15.100: beagleconnect-start-gateway

```
bone$ sudo beagleconnect-start-gateway
setting up wpanusb gateway for IEEE 802154 CHANNEL 1(906 Mhz)
RTNETLINK answers: File exists
RTNETLINK answers: Device or resource busy
PING 2001:db8::1(2001:db8::1) from fe80::212:4b00:29b9:9884%lowpan0 lowpan0:␣
↪56 data bytes
64 bytes from 2001:db8::1: icmp_seq=1 ttl=64 time=70.0 ms
64 bytes from 2001:db8::1: icmp_seq=2 ttl=64 time=66.6 ms
64 bytes from 2001:db8::1: icmp_seq=3 ttl=64 time=37.6 ms
64 bytes from 2001:db8::1: icmp_seq=4 ttl=64 time=37.6 ms
64 bytes from 2001:db8::1: icmp_seq=5 ttl=64 time=37.6 ms

--- 2001:db8::1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 37.559/49.868/70.035/15.084 ms
```

Useful Links https://docs.micropython.org/en/latest/zephyr/quickref.html

https://docs.zephyrproject.org/latest/boards/arm/beagle_bcf/doc/index.html

micropython Examples Here is the output from running the examples from here: https://docs.beagleboard.org/latest/boards/beagleconnect/freedom/demos-and-tutorials/using-micropython.html

Plug the BeagleConnect Freedom into the USB on the Play.

```
bone:~$ sudo systemd-resolve --set-mdns=yes --interface=lowpan0
bone:~$ avahi-browse -r -t _zephyr._tcp
+ lowpan0 IPv6 zephyr                                        _zephyr._tcp   ␣
↪     local
= lowpan0 IPv6 zephyr                                        _zephyr._tcp   ␣
↪     local
hostname = [zephyr.local]
address = [2001:db8::1]
port = [12345]
```

(continues on next page)

(continued from previous page)

```
txt = []
bone:~$ avahi-resolve -6 -n zephyr.local
zephyr.local    2001:db8::1
bone:~$ mcumgr conn add bcf0 type="udp" connstring="[2001:db8::1%lowpan0]:1337
↪"
Connection profile bcf0 successfully added
bone:~$ mcumgr -c bcf0 image list
Images:
image=0 slot=0
    version: hu.hu.hu
    bootable: true
    flags: active confirmed
    hash: 16a97391d2570eae80667cfd8c475cb051d4a4a600430b64cb52b59f5db4ce22
Split status: N/A (0)
bone:~$ mcumgr -c bcf0 shell exec "device list"
status=0

devices:
- GPIO_0 (READY)
- random@40028000 (READY)
- UART_1 (READY)
- UART_0 (READY)
- i2c@40002000 (READY)
- I2C_0S (READY)
requires: GPIO_0
requires: i2c@40002000
- flash-controller@40030000 (READY)
- spi@40000000 (READY)
requires: GPIO_0
- ieee802154g (READY)
- gd25q16c@0 (READY)
requires: spi@40000000
- leds (READY)
- HDC2010-HUMIDITY (READY)
requires: I2C_0S
-
bone:~$ mcumgr -c bcf0 shell exec "net iface"
status=0

Hostname: zephyr


Interface 0x20002de4 (IEEE 802.15.4) [1]
========================================
Link addr : 3D:9A:B9:29:00:4B:12:00
MTU       : 125
Flags     : AUTO_START,IPv6
IPv6 unicast addresses (max 3):
        fe80::3f9a:b929:4b:1200 autoconf preferred infinite
        2001:db8::1 manual preferred infinite
IPv6 multicast addresses (max 4):
        ff02::1
        ff02::1:ff4b:1200
        ff02::1:ff00:1
bone:~$ tio /dev/ttyACM0
```

Press the RST button on the Connect.

```
I: gd25q16c@0: SFDP v 1.6 AP ff with 2 PH
I: PH0: ff00 rev 1.6: 16 DW @ 30
I: gd25q16c@0: 2 MiBy flash
```

(continues on next page)

(continued from previous page)

```
I: PH1: ffc8 rev 1.0: 3 DW @ 90
*** Booting Zephyr OS build zephyr-v3.2.0-3470-g14e193081b1f ***
I: Starting bootloader
I: Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
I: Boot source: primary slot
I: Swap type: none
I: Bootloader chainload address offset: 0x20000
I: Jumping to the first image slot


[00:00:00.001,464] <inf> spi_nor: gd25q16c@0: SFDP v 1.6 AP ff with 2 PH
[00:00:00.001,464] <inf> spi_nor: PH0: ff00 rev 1.6: 16 DW @ 30
[00:00:00.001,983] <inf> spi_nor: gd25q16c@0: 2 MiBy flash
[00:00:00.002,014] <inf> spi_nor: PH1: ffc8 rev 1.0: 3 DW @ 90
uart:~$ build time: Feb 22 2023 08:09:25MicroPython v1.19.1 on 2023-02-22;␣
↪zephyr-beagleconnect_freedom with unknown-cpu
Type "help()" for more information.
>>>
```

Setting up shortcuts to make life easier

We'll be ssh'ing from the host to the bone often, here are some shortcuts I use so instead of typing ssh debian@192.168.7.2 and a password every time. I can enter *ssh bone* and no password.

First edit */etc/hosts* and add a couple of lines.

```
host$ sudo nano /etc/hosts
```

You may use whatever editor you want. I suggest nano since it's easy to figure out. Add the following to the end of */etc/hosts* and quit the editor.

```
192.168.7.2 bone
```

Now you can connect with

```
host$ ssh debian@bone
```

Let's make it so you don't have to enter *debian*. On your host computer, put the following in *~/.ssh/config* (Note: ~ is a shortcut for your home directory.)

```
Host bone
User debian
UserKnownHostsFile /dev/null
StrictHostKeyChecking no
```

These say that whenever you login to bone, login as *debian*. Now you can enter.

```
host$ ssh bone
```

One last thing, let's make it so you don't have to add a password. Back to your host.

```
host$ ssh-keygen
```

Accept all the defaults and then

```
host$ ssh-copy-id bone
```

Now all you have to enter is

```
host$ ssh bone
```

and no password is required. If you, especially virtual machine users, get an error says "sign_and_send_pubkey: signing failed: agent refused operation", you can solve this by entering

```
host$ ssh-add
```

which adds the private key identities to the authentication agent. Then you should be able to *ssh bone* without problems.

Setting up a root login

By default the image we are running doesn't allow a root login. You can always sudo from debian, but sometimes it's nice to login as root. Here's how to setup root so you can login from your host without a password.

```
host$ ssh bone

bone$ sudo -i

root@bone# nano /etc/ssh/sshd_config
```

Search for the line

#PermitRootLogin prohibit-password

and change it to

PermitRootLogin yes

(The # symbol indicates a comment and must be removed in order for the setting to take effect.)

Save the file and quit the editor. Restart ssh so it will reread the file.

root@bone# systemctl restart sshd

And assign a password to root.

root@bone# passwd

Now open another window on your host computer and enter:

host$ ssh-copy-id root@bone

and enter the root password. Test it with:

host$ ssh root@bone

You should be connected without a password. Now go back to the Bone and turn off the root password access.

root@bone# nano /etc/ssh/sshd_config

Restore the line:

#PermitRootLogin prohibit-password

and restart sshd.

```
root@bone# systemctl restart sshd
root@bone# exit
bone$ exit
```

You should now be able to go back to your host computer and login as root on the bone without a password.

host$ ssh root@bone

You have access to your bone without passwords only from you host computer. Try it from another computer and see what happens

Wireshark

Wireshark is a network protocol analyzer that can be run on the Beagle or the host computer to see what's happening on the network.

Running Wireshark on the Beagle If you have X11 installed on the Beagle and you are running Linux on your host you can run Wireshark on the Beagle and have it display on the host.

Tip: A quick way to see if you have X windows installed is to ssh to your Beagle. At the prompt enter `xfce` then enter <TAB><TAB>. If you see a list of completions, you have X installed.

---

1. First ssh to the Beagle using the *-X* flag.

```
host$ ssh -X debian@10.0.5.10

bone$ sudo apt update
bone$ sudo apt install wireshark
bone$ sudo usermod -a -G wireshark debian
bone$ exit

host$ ssh -X debian@10.0.5.10
host$ wireshark
```

The *-X* flag sets the *DISPLAY* variable on the Beagle so it knows where to display the Beagle's graphical data on the host. We then install wireshark and add debian to the wireshark group. We then log out and log back in again to be sure we are in the wireshark group. Finally we start wireshark.

You should see something like *Wireshark start screen*.

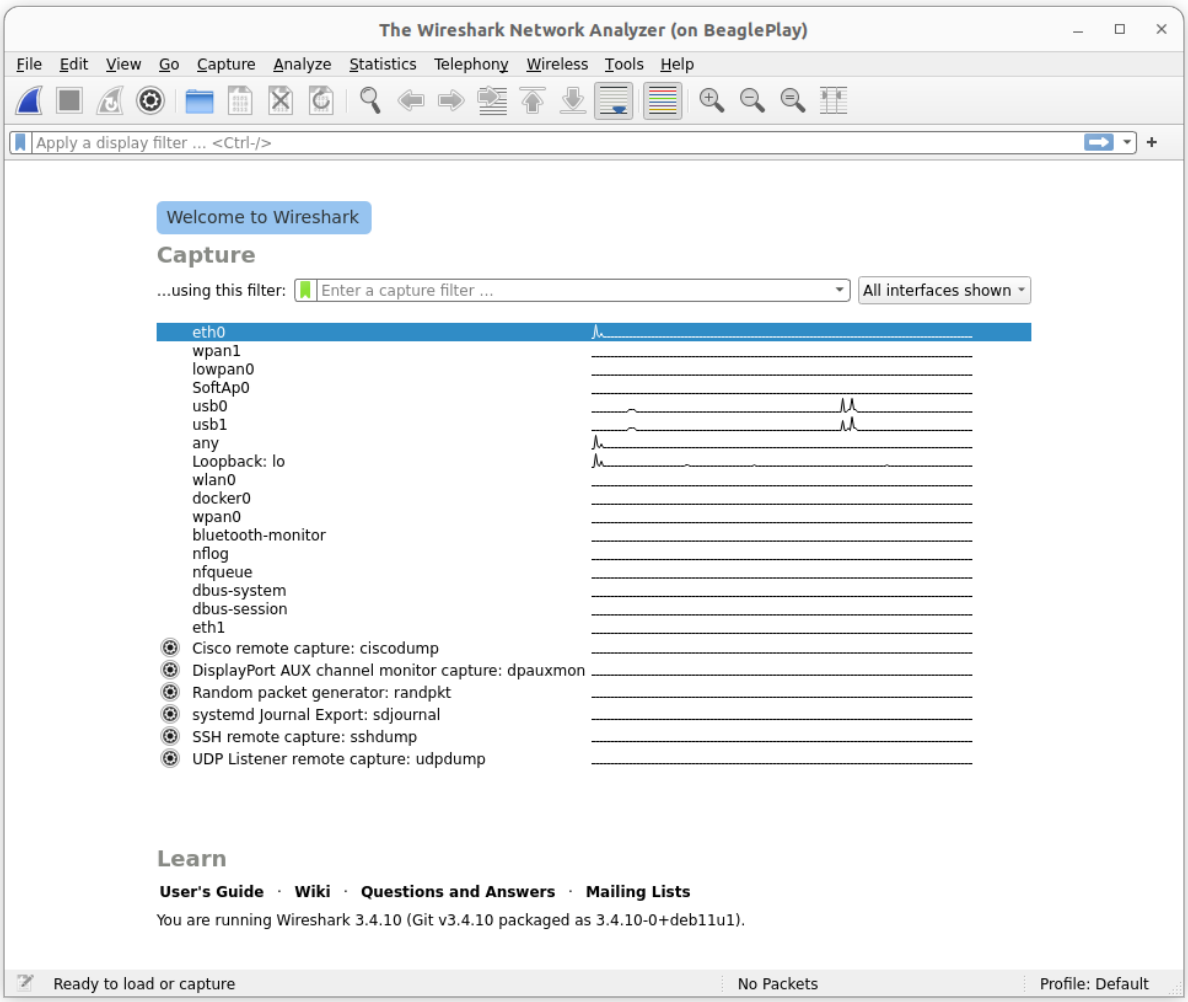


Fig. 15.101: Wireshark start screen

Running Wireshark on the host If you don't have X11 installed on the Beagle, you can run wireshark on your host computer and capture the packets on the Beagle. These instructions come from: https://serverfault.com/questions/362529/how-can-i-sniff-the-traffic-of-remote-machine-with-wireshark

First login to the Beagle and install tcpdump. Use your Beagle's IP address.

```
host$ ssh 192.168.7.2
bone$ sudo apt update
bone$ sudo apt install tcpdump
bone$ exit
```

Next, create a named pipe and have wireshark read from it.

```
host$ mkfifo /tmp/remote
host$ wireshark -k -i /tmp/remote
```

Then, run tcpdump over ssh on your remote machine and redirect the packets to the named pipe:

```
host$ ssh root@192.168.7.2 "tcpdump -s 0 -U -n -w - -i any not port 22" > /
↪tmp/remote
```

Tip: For this to work you will need to follow in instructions in *Setting up a root login*.

Sharking the wpan radio Now that you have Wireshark set up, you can view traffice from the Play's wpan radio. First, set up the network by running:

```
bone:~$ beagleconnect-start-gateway
```

Go to Wireshark and in the field that says *Apply a display filter...* enter, `wpan || 6lowpan || ipv6`. This will dispaly three types of packets. Be sure to hit Enter.

Now generate some traffic:

```
bone:~$ ping6 -I lowpan0 2001:db8::1 -c 5 -p ca11ab1ebeef
```

You can see the pattern `ca11ab1ebeef` appears in the packets.

Converting a tmp117 to a tmp114

Problem You have a tmp114 temperature sensor and you need a driver for it.

Solution Find a similar driver and convert it to the tmp114.

Let's first see if there is a driver for it already. Run the following on the bone using the tab key in place of <tab>.

```
bone$ modinfo tmp<tab><tab>
tmp006  tmp007  tmp102  tmp103  tmp108  tmp401  tmp421  tmp513
bone$ modinfo tmp
```

Here you see a list of modules that match *tmp*, unfortunately *tmp114* is not there. Let's see if there are any matches in */lib/modules*.

```
bone$ find /lib/modules/ -iname "*tmp*"
/lib/modules/5.10.168-ti-arm64-r104/kernel/drivers/iio/temperature/tmp006.ko.
↪xz
/lib/modules/5.10.168-ti-arm64-r104/kernel/drivers/iio/temperature/tmp007.ko.
↪xz
```

(continues on next page)

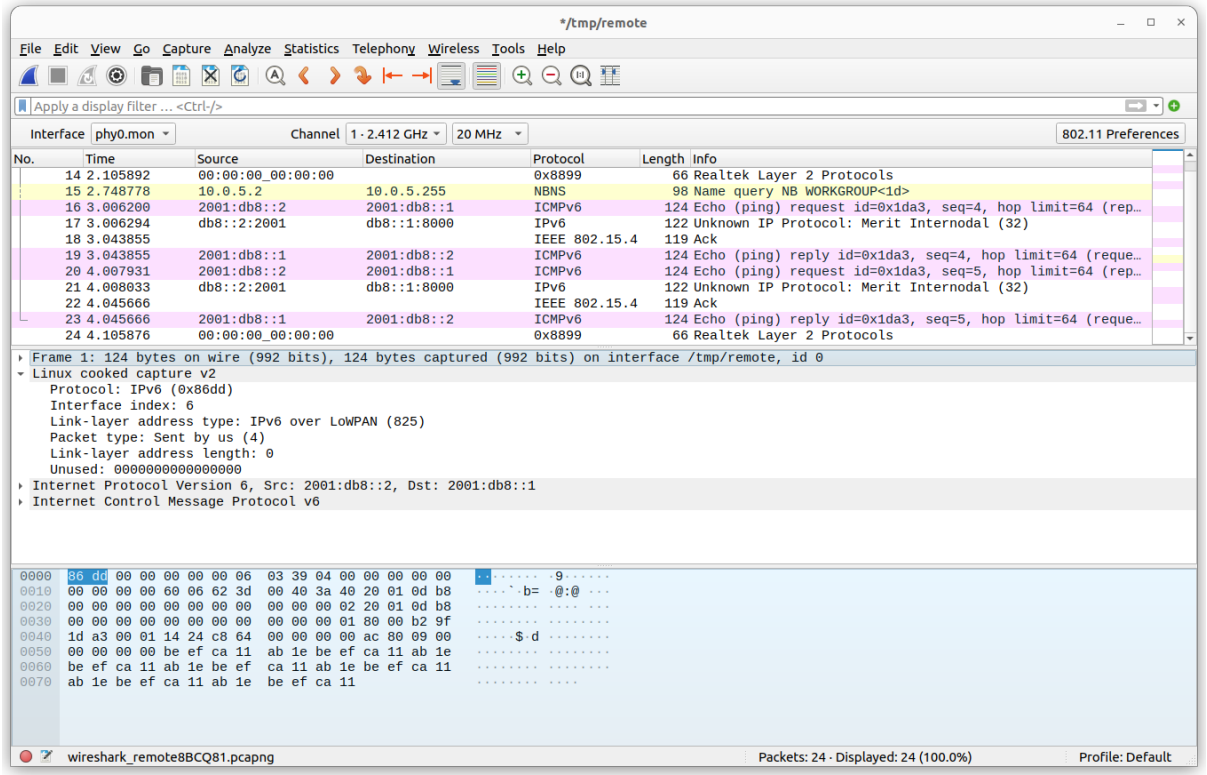


Fig. 15.102: Wireshark ping6 -I lowpan0 2001:db8::1 -c 5 -p ca11ab1ebeef

(continued from previous page)

```
/lib/modules/5.10.168-ti-arm64-r104/kernel/drivers/hwmon/tmp103.ko.xz
/lib/modules/5.10.168-ti-arm64-r104/kernel/drivers/hwmon/tmp421.ko.xz
/lib/modules/5.10.168-ti-arm64-r104/kernel/drivers/hwmon/tmp108.ko.xz
/lib/modules/5.10.168-ti-arm64-r104/kernel/drivers/hwmon/tmp513.ko.xz
/lib/modules/5.10.168-ti-arm64-r104/kernel/drivers/hwmon/tmp401.ko.xz
/lib/modules/5.10.168-ti-arm64-r104/kernel/drivers/hwmon/tmp102.ko.xz
```

Looks like the same list, but here we can see what type of driver it is, either *hwmon* or *iio*. hwmon is an older harware monitor. iio is the newer, and prefered, Industrial IO driver. Googling tmp006 and tmp007 shows that they are Infrared Thermopile Sensors, not the same at the *tmp114*. (Google it). Let's keep looking for a more compatible device.

Browse over to http://kernel.org to see if there are tmp114 drivers in the newer versions of the kernel. The first line in the table is **mainline**. Click on the **browse** link on the right. Here you will see the top level of the Linux sourse tree for the *mainline* version of the kernel. Click on **drivers** and then **iio**. Finally, since tmp114 is a temperture sensor, click on **temperature**. Here you see all the source code for the iio temperature drivers for the mainline version of the kernel. We've seen tmp006 and tmp007 as before, tmp117 is new. Maybe it will work. Click on **tmp117.c** to see the code. Looks like it also works for the tmp116 too. Let's try convering it to work with the tmp114.

A quick way to copy the code to the bone is to right-click on the **plain** link and select *Copy link address*. Then, on the bone enter **wget** and paste the link. Mine looks like the following, yours will be similar.

```
bone$ wget https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.
↪git/plain/drivers/iio/temperature/tmp117.c?h=v6.4-rc7
bone$ mv 'tmp117.c?h=v6.4-rc7' tmp117.c
bone$ cp tmp117.c tmp114.c
```

The **mv** command moves the downloaded file to a usable name and the **cp** copies to a new file with the new name.

Compiling the module Next we need to compile the driver. To do this we need to load the corresponding header files for the version of the kernel that's beening run.

```
bone$ uname -r
5.10.168-ti-arm64-r105
```

Here you see which version I'm running, yours will be similar. Now load the headers.

```
bone$ sudo apt install linux-headers-`uname -r`
```

Next create a *Makefile*. Put the following in a file called *Makefile*.

Listing 15.67: Makefile for compiling module (Makefile)

```
1  obj-m += tmp114.o
2
3  KDIR ?= /lib/modules/$(shell uname -r)/build
4  PWD := $(CURDIR)
5
6  all:
7          make -C $(KDIR) M=$(PWD) modules
8
9  clean:
10         make -C $(KDIR) M=$(PWD) cleanobj-m += tmp114.o
11
12 KDIR ?= /lib/modules/$(shell uname -r)/build
13 PWD := $(CURDIR)
14
15 all:
16         make -C $(KDIR) M=$(PWD) modules
17
18 clean:
19         make -C $(KDIR) M=$(PWD) clean
```

Makefile

Now you are ready to compile:

```
bone$ make
make -C /lib/modules/5.10.168-ti-arm64-r105/build M=/home/debian/play modules
make[1]: Entering directory '/usr/src/linux-headers-5.10.168-ti-arm64-r105'
CC [M]  /home/debian/play/tmp114.o
/home/debian/play/tmp114.c: In function ‘tmp117_identify’:
/home/debian/play/tmp114.c:150:7: error: implicit declaration of function␣
↪‘i2c_client_get_device_id’; did you mean ‘i2c_get_device_id’? [-
↪Werror=implicit-function-declaration]
150 |  id = i2c_client_get_device_id(client);
    |       ^~~~~~~~~~~~~~~~~~~~~~~
    |       i2c_get_device_id
/home/debian/play/tmp114.c:150:5: warning: assignment to ‘const struct i2c_
↪device_id *’ from ‘int’ makes pointer from integer without a cast [-Wint-
↪conversion]
150 |  id = i2c_client_get_device_id(client);
    |     ^
cc1: some warnings being treated as errors
make[2]: *** [scripts/Makefile.build:286: /home/debian/play/tmp114.o] Error 1
make[1]: *** [Makefile:1822: /home/debian/play] Error 2
make[1]: Leaving directory '/usr/src/linux-headers-5.10.168-ti-arm64-r105'
make: *** [Makefile:7: all] Error 2
```

Well, the good news is, it is compiling, that means it found the correct headers. But now the work begins converting to the tmp114.

Converting to the tmp114 You are mostly on your own for this part, but here are some suggestions:

- First get it to compile without errors. In this case, the function at line 150 isn't defined. Try commenting it out and recompiling.
- Once it's compiling without errors, try running it. First open another window and login to beagle. Then run:

```
bone$ dmesg -Hw
```

This will display the kernel messages. The **-H** put them in *human* readable form, and the **-w** waits for more messages.

- Next, "insert" it in the running kernel:

```
bone$ sudo insmod tmp114.ko
```

If all worked you shouldn't see any messages, either after the command or in the dmesg window. If you want to insert the module again, you will have to remove it first. Remove with:

```
bone$ sudo rmmod tmp114
```

Now we need to tell the kernel we have an I^{2}C device and which bus and which address.

Finding your I^{2}C device Each I^{2}C device appears at a certain address on a given bus. My device is on bus 3, so I run:

```
bone$ i2cdetect -y -r 3
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- 4d -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

This shows there is a device at address **0x4d**. If you don't know your bus number, just try a few until you find it.

The temperature is in register 0 for my device and it's 16 bits (one word), it is read with:

```
bone$  i2cget -y 3 0x4d 0 w
0xb510
```

The tmp114 swaps the two bytes, so the real temperature is **0x10b5**, or so. You need to look up the datawsheet to learn how to comvert it.

Registers and IDs Each I^{2}C device has a number of internal registers that interact with the device. The tmp114 uses different register numbers than the tmp117, so you need to change these values. To do this, Google for the data sheets for each and look them up. I found them at: https://www.ti.com/lit/gpn/tmp114 and https://www.ti.com/lit/gpn/tmp117.

Creating a new device Once you've converted the module for the tmp114 and inserted it, you can now create a new device.

```
bone$ cd /sys/class/i2c-adapter/i2c-3
bone$ sudo chgrp gpio *
bone$ sudo chmod g+w *
```

(continues on next page)

(continued from previous page)

```
bone$ ls -ls
total 0
0 --w--w---- 1 root gpio 4096 Jun 22 18:24 delete_device
0 lrwxrwxrwx 1 root root    0 Jan  1  1970 device -> ../../20030000.i2c
0 drwxrwxr-x 3 root gpio    0 Jun 22 18:20 i2c-dev
0 -r--rw-r-- 1 root gpio 4096 Jun 22 18:20 name
0 --w--w---- 1 root gpio 4096 Jun 22 18:20 new_device
0 lrwxrwxrwx 1 root root    0 Jan  1  1970 of_node -> ../../../../../
↪firmware/devicetree/base/bus@f0000/i2c@20030000
0 drwxrwxr-x 2 root gpio    0 Jun 22 18:20 power
0 lrwxrwxrwx 1 root root    0 Jan  1  1970 subsystem -> ../../../../../bus/
↪i2c
0 -rw-rw-r-- 1 root gpio 4096 Jun 22 18:20 uevent
```

The first line changes to the directory to where we can create the new device. The final **3** in the path is for bus **3**, your milage may vary. We then change the group to **gpio** and give it write permission. You only need to do this once.

Now make a new device.

```
bone$ echo tmp114 0x4d > new_device
```

Look in the demsg window and you should see:

```
[Jun22 19:24] tmp114 3-004d: tmp114_identify id (0x1114)
[  +0.000027] tmp114 3-004d: tmp114_probe id (0x1114)
[  +0.000502] i2c i2c-3: new_device: Instantiated device tmp114 at 0x4d
```

It's been found! Let's see what it knows about it.

```
bone$ iio_info
Library version: 0.24 (git tag: v0.24)
...
        iio:device1: tmp114
                1 channels found:
                        temp:  (input)
                        2 channel-specific attributes found:
                                attr  0: raw value: 4257
                                attr  1: scale value: 7.812500
                No trigger on this device
```

I've left out some of the lines, at the bottom you see the tmp114, and two values (**raw** and **scale**) that were read from it. Let's read them ourselves. Do an *ls* and you'll see a new directory, **3-004d**. This is address 0x4d on bus 3, just what we wanted.

```
bone$ cd 3-004d/iio:device1
bone$ ls
dev  in_temp_raw  in_temp_scale  name  power  subsystem  uevent
bone$ cat in_temp_raw
4275
```

You'll have to look in the datasheet to learn how to convert the temperature.

If you try to run i2cget again, you'll get an error:

```
bone$ i2cget -y 3 0x4d 0 w
Error: Could not set address to 0x4d: Device or resource busy
```

This is because the module is using it. Delete the device and you'll have access again.

```
bone$ echo 0x4d > /sys/class/i2c-adapter/i2c-3/delete_device
bone$ i2cget -y 3 0x4d 0 w
0x8e10
```

You should also see a message in dmesg.

Documenting with Sphinx

Problem You want to add or update the Beagle documentation.

Solution BeagleBoard.org uses the Sphinx Python Documentation Generator and the rst markup language.

Here's what you need to do to fork the repository and render a local copy of the documentation. Browse to https://docs.beagleboard.org/latest/ and click on the **Edit on GitLab** button on the upper-right of the page. Clone the repository.

```
bash$ git clone git@git.beagleboard.org:docs/docs.beagleboard.io.git
bash$ cd docs.beagleboard.io
```

Then run the following to load the **code** submodule

```
bash$ git submodule update --init
```

Now, sync changes with upstream:

```
bone$ git remote add upstream https://git.beagleboard.org/docs/docs.
↪beagleboard.io.git
bone$ git fetch upstream
bone$ git pull upstream main
```

Downloading Sphinx Run the following to download Sphinx. Note: This will take a while, it loads some 6G bytes.

```
bone$ sudo apt update
bone$ sudo apt upgrade
bone$ sudo apt install -y \
    make git wget \
    doxygen graphviz librsvg2-bin\
    texlive-latex-base texlive-latex-extra latexmk texlive-fonts-recommended␣
↪\
    python3 python3-pip \
    python3-sphinx python3-sphinx-rtd-theme python3-sphinxcontrib.
↪svg2pdfconverter \
    python3-pil \
    imagemagick-6.q16 librsvg2-bin webp \
    texlive-full texlive-latex-extra texlive-fonts-extra \
    fonts-freefont-otf fonts-dejavu fonts-dejavu-extra fonts-freefont-ttf
bone$ python3 -m pip install --upgrade pip
bone$ pip install -U sphinx_design
bone$ pip install -U sphinxcontrib-images
bone$ pip install -U sphinx-serve
```

These instructions came from lorforlinux on the Beagleboard Slack channel.

Now go to the cloned *docs.beagleboard.io* repository folder and do the following. To clean build directory:

```
bone$ cd docs.beagleboard.io
bone$ make clean
```

To generate HTML output of docs:

```
bone$ make html
```

To generate PDF output of docs:

```
bone$ make latexpdf
```

To preview docs on your local machine:

```
bone$ sphinx-serve
```

Then point your browser to localhost:8081.

Tip: You can keep the sphinx-serve running until you clean the build directory using make clean. Warnings will be hidden after first run of make html or make latexpdf, to see all the warnings again just run make clean before building HTML or PDF

Creating A New Book

- Create a new book folder here: https://git.beagleboard.org/docs/docs.beagleboard.io/-/tree/main/books
- Create rst files for all the chapters in there respective folders so that you can easily manage media for that chapter as shown here: https://git.beagleboard.org/docs/docs.beagleboard.io/-/tree/main/books/pru-cookbook
- Create an index.rst file in the book folder and add a table of content (toc) for all the chapters. For example see this file: https://git.beagleboard.org/docs/docs.beagleboard.io/-/raw/main/books/pru-cookbook/index.rst
- Add the bookname/index.rst reference in the main index file as well: https://git.beagleboard.org/docs/docs.beagleboard.io/-/raw/main/books/index.rst
- At last you have to update the two files below to render the book in HTML and PDF version of the docs respectively: https://git.beagleboard.org/docs/docs.beagleboard.io/-/raw/main/index.rst https://git.beagleboard.org/docs/docs.beagleboard.io/-/raw/main/index-tex.rst

Running Sparkfun's qwiic Python Examples

Many of the Sparkfun qwiic devices have Python examples showing how to use them. Unfortunately the examples assume I^2C bus 1 is used, but the qwiic bus on the Play is bus 5. Here is a quick hack to get the Sparkfun Python examples to use bus 5. I'll show it for the Joystick, but it should work for the others as well.

First, browse to Sparkfun's qwiic Joystick page, https://www.sparkfun.com/products/15168 and click on the **DOCUMENTS** tab and then on **Python Package**. Follow the pip instillation instructions (sudo pip install sparkfun-qwiic-joystick)

Next, uninstall the current qwiic I^2C package.

```
bone$ sudo pip uninstall sparkfun-qwiic-i2c
```

Then clone the Qwiic I^2C repo:

```
bone$ git clone git@github.com:sparkfun/Qwiic_I2C_Py.git
bone$ cd Qwiic_I2C_Py/qwiic_i2c
```

Edit **linux_i2c.py** and go to around line 62 and change it to:

```
iBus = 5
```

Next, cd up a level to the Qwiic_I2C_Py directory and reinstall

```
bone$ cd ..
bone$ sudo python setup.py install
```

Finally, run one of the Joystick examples. If it isn't using bus 5, try reinstalling setup.py again.

Qwiic Alphanumeric display Here's the repo I used for this display. https://github.com/thess/qwiic_alphanumeric_py

Controlling LEDs by Using SYSFS Entries

Problem You want to control the onboard LEDs from the command line.

Solution On Linux, everything is a file that is, you can access all the inputs and outputs, the LEDs, and so on by opening the right `file` and reading or writing to it. For example, try the following:

```
bone$ cd /sys/class/leds/
bone$ ls
beaglebone:green:usr0  beaglebone:green:usr2
beaglebone:green:usr1  beaglebone:green:usr3
```

What you are seeing are four directories, one for each onboard LED. Now try this:

```
bone$ cd beaglebone\:green\:usr0
bone$ ls
brightness  device  max_brightness  power  subsystem  trigger  uevent
bone$ cat trigger
none nand-disk mmc0 mmc1 timer oneshot [heartbeat]
    backlight gpio cpu0 default-on transient
```

The first command changes into the directory for LED *usr0*, which is the LED closest to the edge of the board. The *[heartbeat]* indicates that the default trigger (behavior) for the LED is to blink in the heartbeat pattern. Look at your LED. Is it blinking in a heartbeat pattern?

Then try the following:

```
bone$ echo none > trigger
bone$ cat trigger
[none] nand-disk mmc0 mmc1 timer oneshot heartbeat
    backlight gpio cpu0 default-on transient
```

This instructs the LED to use *none* for a trigger. Look again. It should be no longer blinking.

Now, try turning it on and off:

```
bone$ echo 1 > brightness
bone$ echo 0 > brightness
```

The LED should be turning on and off with the commands.

Controlling GPIOs by Using SYSFS Entries

Problem You want to control a GPIO pin from the command line.

Solution *Controlling LEDs by Using SYSFS Entries* introduces the *sysfs*. This recipe shows how to read and write a GPIO pin.

Reading a GPIO Pin via sysfs

Suppose that you want to read the state of the *P9_42* GPIO pin. (*Reading the Status of a Pushbutton or Magnetic Switch (Passive On/Off Sensor)* shows how to wire a switch to *P9_42*.) First, you need to map the *P9* header location to GPIO number using *Mapping P9_42 header position to GPIO 7*, which shows that *P9_42* maps to GPIO 7.

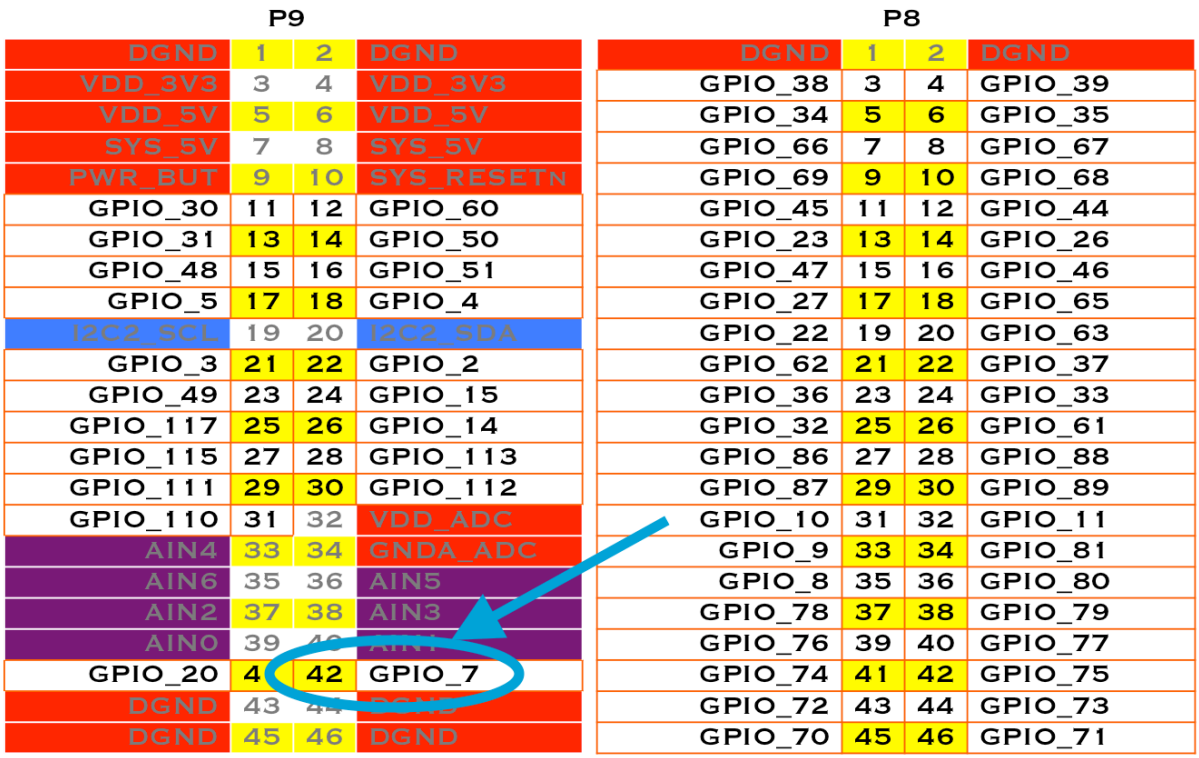


Fig. 15.103: Mapping P9_42 header position to GPIO 7

Next, change to the GPIO *sysfs* directory:

```
bone$ cd /sys/class/gpio/
bone$ ls
export  gpiochip0  gpiochip32  gpiochip64  gpiochip96  unexport
```

The *ls* command shows all the GPIO pins that have be exported. In this case, none have, so you see only the four GPIO controllers. Export using the *export* command:

```
bone$ echo 7 > export
bone$ ls
export  gpio7  gpiochip0  gpiochip32  gpiochip64  gpiochip96  unexport
```

Now you can see the `gpio7` directory. Change into the `gpio7` directory and look around:

```
bone$ cd gpio7
bone$ ls
active_low  direction  edge  power  subsystem  uevent  value
bone$ cat direction
in
bone$ cat value
0
```

Notice that the pin is already configured to be an input pin. (If it wasn't already configured that way, use *echo in > direction* to configure it.) You can also see that its current value is *0*—that is, it isn't pressed. Try pressing and holding it and running again:

```
bone$ cat value
1
```

The *1* informs you that the switch is pressed. When you are done with GPIO 7, you can always *unexport* it:

```
bone$ cd ..
bone$ echo 7 > unexport
bone$ ls
export  gpiochip0  gpiochip32  gpiochip64  gpiochip96  unexport
```

Writing a GPIO Pin via sysfs

Now, suppose that you want to control an external LED. *Toggling an External LED* shows how to wire an LED to *P9_14*. *Mapping P9_42 header position to GPIO 7* shows *P9_14* is GPIO 50. Following the approach in *Controlling GPIOs by Using SYSFS Entries*, enable GPIO 50 and make it an output:

```
bone$ cd /sys/class/gpio/
bone$ echo 50 > export
bone$ ls
gpio50  gpiochip0  gpiochip32  gpiochip64  gpiochip96
bone$ cd gpio50
bone$ ls
active_low  direction  edge  power  subsystem  uevent  value
bone$ cat direction
in
```

By default, *P9_14* is set as an input. Switch it to an output and turn it on:

```
bone$ echo out > direction
bone$ echo 1 > value
bone$ echo 0 > value
```

The LED turns on when a *1* is written to *value* and turns off when a *0* is written.

The Play's Boot Sequence

The BeagleBoard Play is based on the Texas Instrument's AM625 Sitara processor which supports many boot modes.

Note: bootlin (https://bootlin.com/) has many great Linux training materials for free on their site. Their embedded Linux workshop (https://bootlin.com/training/embedded-linux/) gives a detailed presentation of the Play's boot sequence (https://bootlin.com/doc/training/embedded-linux-beagleplay/embedded-linux-beagleplay-labs.pdf, starting at page 9). Check it out for details on building the boot sequence from scratch.

Here we'll take a high-level look at booting from both the user's view and the developer's view.

Booting for the User The most common way for the Play to boot is the power up the board, if the micro SD card is present, it will boot from it, if it isn't present it will boot from the built in eMMC.

You can override the boot sequence by using the **USR** button (located near the micro SD cage). If the **USR** button is pressed the Play will boot from the micro SD card.

Note: If the eMMC fails to boot, it will attempt to boot from the UART. If the SD card fails to boot, it will try booting via the USB.

Booting for the Developer

Tip: These diagrams might help: https://github.com/u-boot/u-boot/blob/6e8fa0611f19824e200fe4725f18bce7e2000071/doc/board/ti/k3.rst

If you are developing firmware for the Play you may need to have access to the processor early in the booting sequence. Much can happen before the Linux kernel starts its boot process. Here are some notes on what the BeagleBoard Play does when it boots up. Many of the booting details come from Chapter 5 (Initialization) of the AM62x Technical Reference Manual (TRM) (https://www.ti.com/product/AM625, https://www.ti.com/lit/pdf/spruiv7). The following figure, taken from page 2456, shows the Initialization Process.



Figure 5-1. Initialization Process

Fig. 15.104: Initialization Process

We are interested in what happens in the **ROM code**. Page 2457, of the TRM, shows the different ROM Code Boot Modes.

Table 5-1. ROM Code Boot Modes

Boot Mode	Boot Media/Host	SoC Peripheral	Can be a Backup Mode?(1)	Notes
No-boot/Dev-boot	No media or host	None	N	No boot or development boot – debug modes
OSPI	OSPI flash	FSS0_OSPI0	N	On OSPI port
QSPI	QSPI flash	FSS0_OSPI0	N	On OSPI port
SPI	SPI flash	FSS0_OSPI0	Y	On OSPI port
Ethernet	External host	CPSW0	Y	In BOOTP mode. RGMII or RMII PHY
I2C	I^2C EEPROM	I2C0	Y	I2C target boot is not supported
UART	External host	UART0	Y	XMODEM protocol
MMCSD	eMMC flash or SD card	MMCSD0 (8bit) or MMCSD1 (4bit)	Y	Boot from User Data Area (UDA) in raw or file system mode
eMMC	eMMC flash	MMCSD0 (8bit)	N	Boot from boot partition
USB - target	USB boot from external host	USB0	Y	USB device mode boot using DFU (device firmware upgrade), Boot is running on USB2.0 speeds.
USB - host	USB mass storage	USB0	Y	USB2.0 host mode, boot from FAT32 filesystem
Serial Flash	Serial Flash	FSS0_OSPI0	N	On OSPI port
xSPI	xSPI flash	FSS0_OSPI0	N	On OSPI port
GPMC	NOR flash, NAND flash	GPMC0	N	CSn0 connected, 8 bit NAND flash only, 16-bit non-mux NOR flash only

(1) The peripheral can be selected also as a backup boot mode. A backup mode is tried if primary boot mode fails.

Fig. 15.105: ROM Code Boot Modes

These are selected at boot time based on the state of the BOOTMODE pins. The table on page 2465 shows the BOOTMODE pins.

Page 14 of of the Play's schematic (https://git.beagleboard.org/beagleplay/beagleplay/-/blob/main/BeaglePlay_sch.pdf) shows how the BOOTMODE pins are set during boot.

Therefore the following modes are selected if **Button Not-pressed**

Table 5-2. BOOTMODE Pin Mapping

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Reserved	Backup Boot Mode Config	Backup Boot Mode			Primary Boot Mode Config			Primary Boot Mode				PLL Config		

Table 5-3 describes the BOOTMODE pins that need to be set according to the system clock provided to the device.

Fig. 15.106: BOOTMODE Pin Mapping

Bootstrap

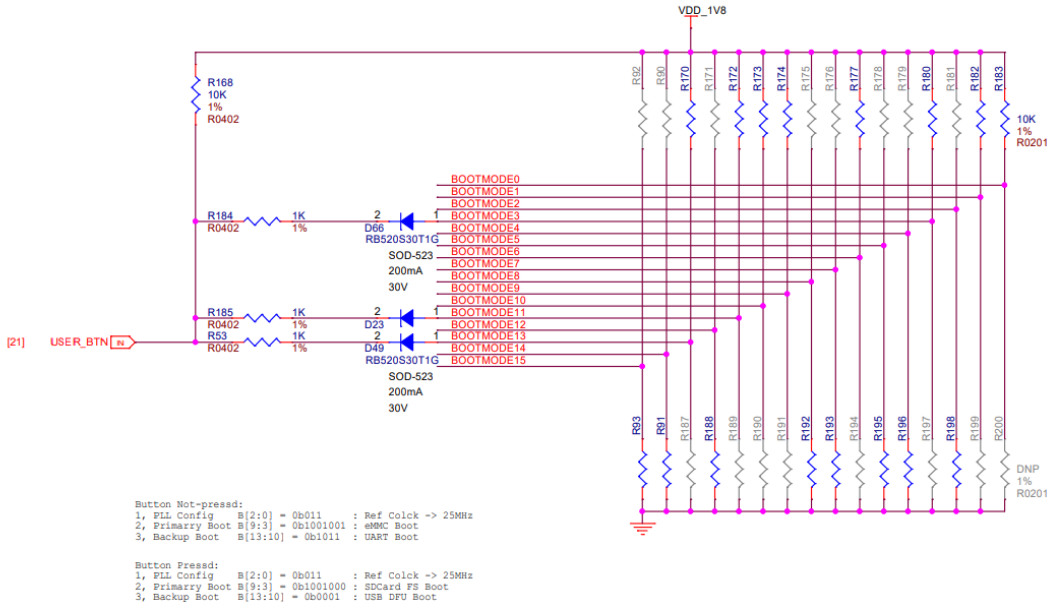


Fig. 15.107: Bootstrap

1,	PLL Config	B[2:0] = 0b011	: Ref Clcok -> 25MHz
2,	Primary Boot	B[9:3] = 0b1001001	: eMMC Boot
3,	Backup Boot	B[13:10] = 0b1011	: UART Boot

That is, you boot off the eMMC and if that fails you boot off the UART.

If **Button is Pressed**

1,	PLL Config	B[2:0] = 0b011	: Ref Clcok -> 25MHz
2,	Primary Boot	B[9:3] = 0b1001000	: SD Card FS Boot
3,	Backup Boot	B[13:10] = 0b0001	: USB DFU Boot

Here you are booting off the SD card (in filesystem mode), or the USB if that fails.

Boot Flow There are many steps that occur after the BOOTMODE is selected and before the Linux Kernel boots. Boot Flow shows those steps for the **R5** processor and the arm (**A53**) processor. The key parts are **tiboot3.bin** and **tispl.bin** runnng on the *R5* and **u-boot.img** running on the *A53*. These binary files are found on the Play in `/boot/firmware`.

Note: The files on the SD card and the eMMC are in `ext4` format. The files used for booting must be in `vfat` format. Therefore `/boot/firmware` is mounted in `vfat` as seen in `/etc/fstab`.

```
# /etc/fstab: static file system information.
#
/dev/mmcblk0p2  /  ext4  noatime,errors=remount-ro  0  1
/dev/mmcblk0p1  /boot/firmware vfat defaults 0 0
debugfs  /sys/kernel/debug  debugfs  mode=755,uid=root,gid=gpio,defaults  0 ␣
↪0
```

Source Code The source code and examples of how to compile the source is found in: https://git.beagleboard.org/beagleboard/repos-arm64/-/blob/main/bb-u-boot-beagleplay/suite/bookworm/debian/rules#L29

Home Assistant

1. **Get an image here:** https://www.beagleboard.org/distros/beagleplay-home-assistant-webinar-demo-image
   I chose the boot from SD image.
2. Boot the Play from the SD card
3. Log into the Play
4. **Find the Play's IP address by running**

   ```
   bone$ ip a show eth0
   2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state␣
   ↪UP group default qlen 1000
       link/ether 34:08:e1:85:1b:a6 brd ff:ff:ff:ff:ff:ff
       inet 10.0.5.24/24 brd 10.0.5.255 scope global dynamic␣
   ↪noprefixroute eth0
   ```

   The address is after `inet`, in my case it's 10.0.5.24.
5. Wait 5 or 10 minutes and then open a browser at `10.0.5.24:8123` using your IP address.

Tip: If you get a "This site can't be reached" error message, try running `journalctl -f` to see the log messages.

---

1. **Open another browser and follow the instructions at:** https://www.home-assistant.io/getting-started/onboarding

mqtt Here are Jason's addons. https://git.beagleboard.org/jkridner/home-assistant-addons

15.2 PRU Cookbook

Contributors

- Author: Mark A. Yoder
- Book revision: v2.0 beta

Outline

A cookbook for programming the PRUs in C using remoteproc and compiling on the Beagle

15.2.1 Case Studies - Introduction

It's an exciting time to be making projects that use embedded processors. Make:'s Makers' Guide to Boards shows many of the options that are available and groups them into different types. *Single board computers* (SBCs) generally run Linux on some sort of ARM processor. Examples are the BeagleBoard and the Raspberry Pi. Another type is the *microcontroller*, of which the Arduino is popular.

The SBCs are used because they have an operating system to manage files, I/O, and schedule when things are run, all while possibly talking to the Internet. Microcontrollers shine when things being interfaced require careful timing and can't afford to have an OS preempt an operation.

But what if you have a project that needs the flexibility of an OS and the timing of a microcontroller? This is where the BeagleBoard excels since it has both an ARM procssor running Linux and two[1] **P**rogrammable **R**eal-Time **U**nits (PRUs). The PRUs have 32-bit cores which run independently of the ARM processor, therefore they can be programmed to respond quickly to inputs and produce very precisely timed outputs.

There are many *Projects* that use the PRU. They are able to do things that can't be done with just a SBC or just a microcontroller. Here we present some case studies that give a high-level view of using the PRUs. In later chapters you will see the details of how they work.

Here we present:

- Robotics Control Library
- BeagleLogic
- NeoPixels – 5050 RGB LEDs with Integrated Drivers (Falcon Christmas)
- RGB LED Matrix (Falcon Christmas)
- simpPRU – A python-like language for programming the PRUs
- MachineKit
- BeaglePilot
- BeagleScope

The following are resources used in this chapter.

Resources

- PocketBeagle System Reference Manual

[1] Four if you are on the BeagleBone AI

- **BeagleBone Black P8 Header Table**
  - P8 Header Table from exploringBB
- **BeagleBone Black P9 Header Table**
  - P9 Header Table from exploringBB
- BeagleBone AI System Reference Manual

Robotics Control Library

Robotics is an embedded application that often requires both an SBC to control the high-level tasks (such as path planning, line following, communicating with the user) *and* a microcontroller to handle the low-level tasks (such as telling motors how fast to turn, or how to balance in response to an IMU input). The EduMIP balancing robot demonstrates that by using the PRU, the Blue can handle both the high and low -level tasks without an additional microcontroller. The EduMIP is shown in *Blue balancing*.

The Robotics Control Library is a package that is already installed on the Beagle that contains a C library and example/testing programs. It uses the PRU to extend the real-time hardware of the Bone by adding eight additional servo channels and one addition real-time encoder input.

The following examples show how easy it is to use the PRU for robotics.

Controlling Eight Servos

Problem You need to control eight servos, but the Bone doesn't have enough pulse width modulation (PWM) channels and you don't want to add hardware.

Solution The Robotics Control Library provides eight additional PWM channels via the PRU that can be used out of the box.

Note: The I/O pins on the Beagles have a mutliplexer that lets you select what I/O appears on a given pin. The Blue has the mux already configured to run these examples. Follow the instructions in *Configuring Pins for Controlling Servos* to configure the pins for the Black and the Pocket.

Just run:

```
bone$ sudo rc_test_servos -f 10 -p 1.5
```

The `-f 10` says to use a frequency of 10 Hz and the `-p 1.5` says to set the position to `1.5`. The range of positions is `-1.5` to `1.5`. Run `rc_test_servos -h` to see all the options.

```
bone$ rc_test_servos -h

Options
-c {channel}   Specify one channel from 1-8.
               Otherwise all channels will be driven equally
-f {hz}        Specify pulse frequency, otherwise 50hz is used
-p {position}  Drive servo to a position between -1.5 & 1.5
-w {width_us}  Send pulse width in microseconds (us)
-s {limit}     Sweep servo back/forth between +- limit
               Limit can be between 0 & 1.5
-r {ch}        Use DSM radio channel {ch} to control servo
-h             Print this help message

sample use to center servo channel 1:
   rc_test_servo -c 1 -p 0.0
```

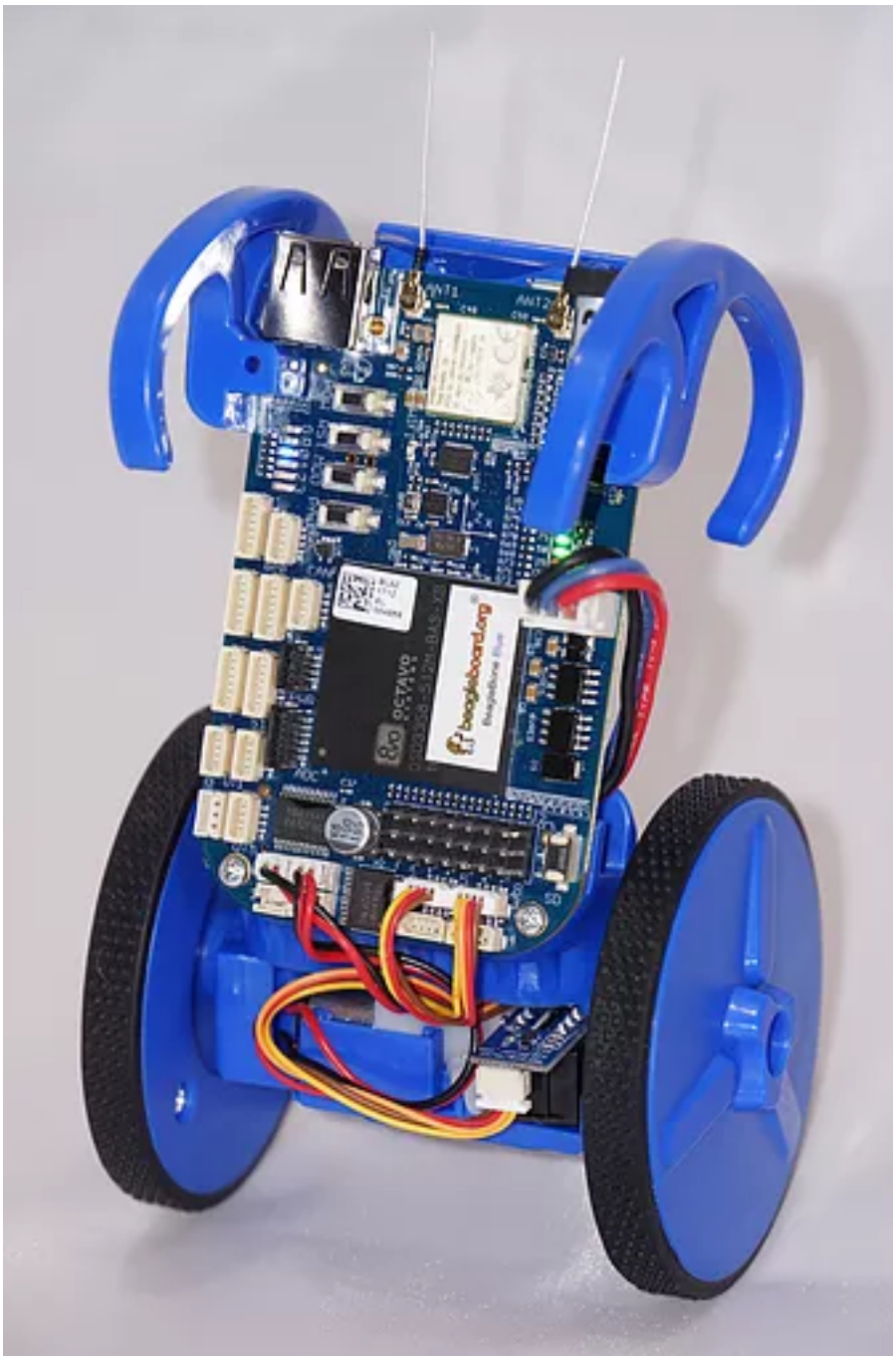


Fig. 15.108: Blue balancing

Discussion The BeagleBone Blue sends these eight outputs to its servo channels. The others use the pins shown in the *PRU register to pin table*.

PRU register to pin table

PRU pin	Blue pin	Black pin	Pocket pin	AI pin
pru1_r30_8	1	P8_27	P2.35	
pru1_r30_10	2	P8_28	P1.35	P9_42
pru1_r30_9	3	P8_29	P1.02	P8_14
pru1_r30_11	4	P8_30	P1.04	P9_27
pru1_r30_6	5	P8_39		P8_19
pru1_r30_7	6	P8_40		P8_13
pru1_r30_4	7	P8_41		
pru1_r30_5	8	P8_42		P8_18

You can find these details in the

- PocketBeagle pinout
- BeagleBone AI PRU pins

Be default the PRUs are already loaded with the code needed to run the servos. All you have to do is run the command.

Controlling Individual Servos

Problem `rc_test_servos` is nice, but I need to control the servos individually.

Solution You can modify `rc_test_servos.c`. You'll find it on the bone online at https://git.beagleboard.org/beagleboard/librobotcontrol/-/blob/master/examples/src/rc_test_servos.c

Just past line 250 you'll find a `while` loop that has calls to `rc_servo_send_pulse_normalized(ch, servo_pos)` and `rc_servo_send_pulse_us(ch, width_us)`. The first call sets the pulse width relative to the pulse period; the other sets the width to an absolute time. Use whichever works for you.

Controlling More Than Eight Channels

Problem I need more than eight PWM channels, or I need less jitter on the off time.

Solution This is a more advanced problem and required reprograming the PRUs. See *PWM Generator* for an example.

Reading Hardware Encoders

Problem I want to use four encoders to measure four motors, but I only see hardware for three.

Solution The forth encoder can be implemented on the PRU. If you run `rc_test_encoders_eqep` on the Blue, you will see the output of encoders E1-E3 which are connected to the eEQP hardware.

```
bone$ rc_test_encoders_eqep

Raw encoder positions
      E1    |       E2    |       E3    |
         0  |          0  |          0  |^C
```

You can also access these hardware encoders on the Black and Pocket using the pins shown in *eQEP to pin mapping*.

eQEP to pin mapping

eQEP	Blue pin	Black pin A	Black pin B	AI pin A	AI pin B	Pocket pin A	Pocket pin B
0	E1	P9_42B	P9_27			P1.31	P2.24
1	E2	P8_35	P8_33	P8_35	P8_33	P2.10	
2	E3	P8_12	P8_11	P8_12	P8_11	P2.24	P2.33
2		P8_41	P8_42	P9_19	P9_41		
	E4	P8_16	P8_15			P2.09	P2.18
3				P8_25	P8_24		
3				P9_42	P9_27		

Note: The I/O pins on the Beagles have a mutliplexer that lets you select what I/O appears on a given pin. The Blue has the mux already configured to run these examples. Follow the instructions in *Configuring Pins for Controlling Encoders* to configure the pins for the Black and the Pocket.

Reading PRU Encoder

Problem I want to access the PRU encoder.

Solution The forth encoder is implemented on the PRU and accessed with *sudo rc_test_encoders_pru*

Note: This command needs root permission, so the *sudo* is needed. The default password is *temppwd*.

Here's what you will see

```
bone$ sudo rc_test_encoders_pru
[sudo] password for debian:

Raw encoder position
      E4   |
         0 |^C
```

Note: If you aren't running the Blue you will have to configure the pins as shown in the note above.

BeagleLogic – a 14-channel Logic Analyzer

Problem I need a 100Msps, 14-channel logic analyzer

Solution BeagleLogic documentation is a 100Msps, 14-channel logic analyzer that runs on the Beagle.

information

BeagleLogic turns your BeagleBone [Black] into a 14-channel, 100Msps Logic Analyzer. Once loaded, it presents itself as a character device node */dev/beaglelogic*. The core of the logic analyzer is the 'beaglelogic' kernel module that reserves memory for and drives the two Programmable Real-Time Units (PRU) via the remoteproc interface wherein the PRU directly writes logic samples to the System Memory (DDR RAM) at the configured sample rate one-shot or continuously without intervention from the ARM core.

https://github.com/abhishek-kakkar/BeagleLogic/wiki

The quickest solution is to get the no-setup-required image. It points to an older image (beaglelogic-stretch-2017-07-13-4gb.img.xz) but should still work.

If you want to be running a newer image, there are instructions on the site for installing BeagleLogic, but I had to do the additional steps in *Installing BeagleLogic*.

Listing 15.68: Installing BeagleLogic

```
bone$ git clone https://github.com/abhishek-kakkar/BeagleLogic
bone$ cd BeagleLogic/kernel
bone$ mv beaglelogic-00A0.dts beaglelogic-00A0.dts.orig
bone$ wget https://gist.githubusercontent.com/abhishek-kakkar/
 ↪0761ef7b10822cff4b3efd194837f49c/raw/
 ↪eb2cf6cfb59ff5ccb1710dcd7d4a40cc01cfc050/beaglelogic-00A0.dts
bone$ make overlay
bone$ sudo cp beaglelogic-00A0.dtbo /lib/firmware/
bone$ sudo update-initramfs -u -k \`uname -r`
bone$ sudo reboot
```

Once the Bone has rebooted, browse to 192.168.7.2:4000 where you'll see *BeagleLogic Data Capture*. Here you can easily select the sample rate, number of samples, and which pins to sample. Then click *Begin Capture* to capture your data, at up to 100 MHz!

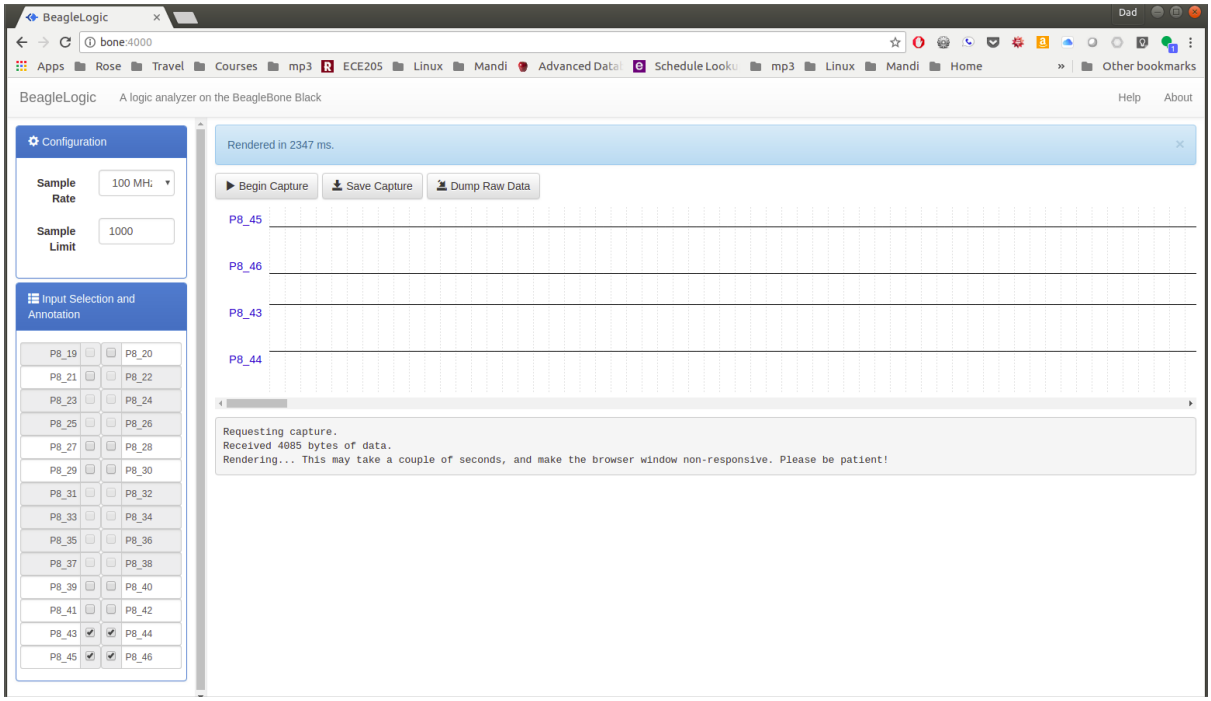


Fig. 15.109: BeagleLogic Data Capture

Discussion BeagleLogic is a complete system that includes firmware for the PRUs, a kernel module and a web interface that create a powerful 100 MHz logic analyzer on the Bone with no additional hardware needed.

Tip: If you need buffered inputs, consider BeagleLogic Standalone, a turnkey Logic Analyzer built on top of BeagleLogic.

The kernel interface makes it easy to control the PRUs through the command line. For example

```
bone$ dd if=/dev/beaglelogic of=mydump bs=1M count=1
```

will capture a binary dump from the PRUs. The sample rate and number of bits per sample can be controlled through `/sys/`.

```
bone$ cd /sys/devices/virtual/misc/beaglelogic
bone$ ls
buffers      filltestpattern  power       state         uevent
bufunitsize  lasterror        samplerate  subsystem
dev          memalloc         sampleunit  triggerflags
bone$ *cat samplerate*
1000
bone$ *cat sampleunit*
8bit
```

You can set the sample rate by simply writing to `samplerate`.

```
bone$ echo 100000000 > samplerate
```

sysfs attributes Reference has more details on configuring via sysfs.

If you run `dmesg -Hw` in another window you can see when a capture is started and stopped.

```
bone$ dmesg -Hw
[Jul25 08:46] misc beaglelogic: capture started with sample rate=100000000␣
 ↪Hz, sampleunit=1, triggerflags=0
[  +0.086261] misc beaglelogic: capture session ended
```

BeagleLogic uses the two PRUs to sample at 100Msps. Getting a PRU running at 200Hz to sample at 100Msps is a slick trick. The Embedded Kitchen has a nice article explaining how the PRUs get this type of performance.

RGB LED Matrix - No Integrated Drivers (Falcon Christmas)

Problem You want to use a RGB LED Matrix display that doesn't have integrated drivers such as the 64x32 RGB LED Matrix by Adafuit shown in *Adafruit LED Matrix*.

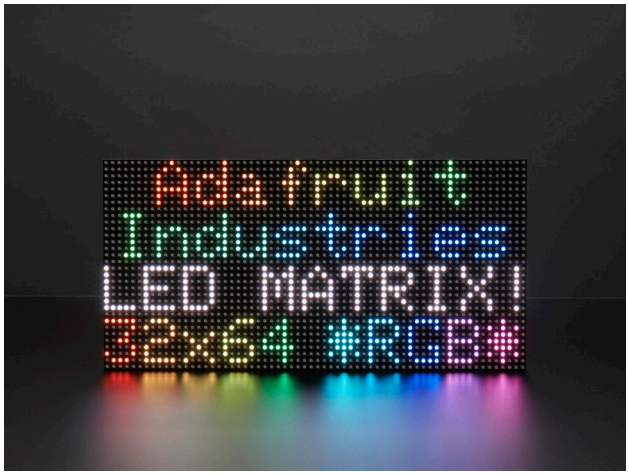


Fig. 15.110: Adafruit LED Matrix

Solution Falcon Christmas makes a software package called Falcon Player (FPP) which can drive such displays.

information:

The Falcon Player (FPP) is a lightweight, optimized, feature-rich sequence player designed to run on low-cost SBC's (Single Board Computers). FPP is a software solution that you download and install on hardware which can be purchased from numerous sources around the internet. FPP aims to be controller agnostic, it can talk E1.31, DMX, Pixelnet, and Renard to hardware from multiple hardware vendors, including controller hardware from Falcon Christmas available via COOPs or in the store on FalconChristmas.com.

http://www.falconchristmas.com/wiki/FPP:FAQ#What_is_FPP.3F

Hardware The Beagle hardware can be either a BeagleBone Black with the Octoscroller Cape, or a PocketBeagle with the PocketScroller LED Panel Cape. (See to purchase.) Building and Octoscroller Matrix Display gives details for using the BeagleBone Black.

PocketBeagle Driving a P5 RGB LED Matrix via the PocketScroller Cape shows how to attach the PocketBeagle to the P5 LED matrix and where to attach the 5V power. If you are going to turn on all the LEDs to full white at the same time you will need at least a 4A supply.

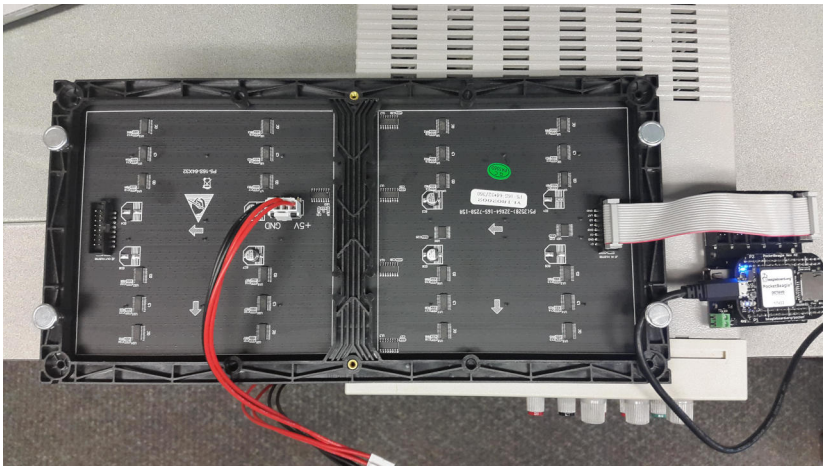


Fig. 15.111: PocketBeagle Driving a P5 RGB LED Matrix via the PocketScroller Cape

Software The FPP software is most easily installed by downloading the current FPP release, flashing an SD card and booting from it.

Tip: The really brave can install it on a already running image. See details at https://github.com/FalconChristmas/fpp/blob/master/SD/FPP_Install.sh

---

Assuming the PocketBeagle is attached via the USB cable, on your host computer browse to http://192.168.7.2/ and you will see *Falcon Play Program Control*.

You can test the display by first setting up the Channel Outputs and then going to *Display Testing*. *Selecting Channel Outputs* shows where to select Channel Outputs and *Channel Outputs Settings* shows which settings to use.

Click on the **LED Panels** tab and then the only changes I made was to select the **Single Panel Size** to be *64x32* and to check the **Enable LED Panel Output**.

Next we need to test the display. Select *Display Testing* shown in *Selecting Display Testing*.

Set the **End Channel** to **6144**. (6144 is 3*64*32) Click **Enable Test Mode** and your matrix should light up. Try the different testing patterns shown in *Display Testing Options*.

xLights - Creating Content for the Display Once you are sure your LED Matrix is working correctly you can program it with a sequence.

---

information:

xLights is a free and open source program that enables you to design, create and play amazing lighting displays through the use of DMX controllers, E1.31 Ethernet controllers and more.

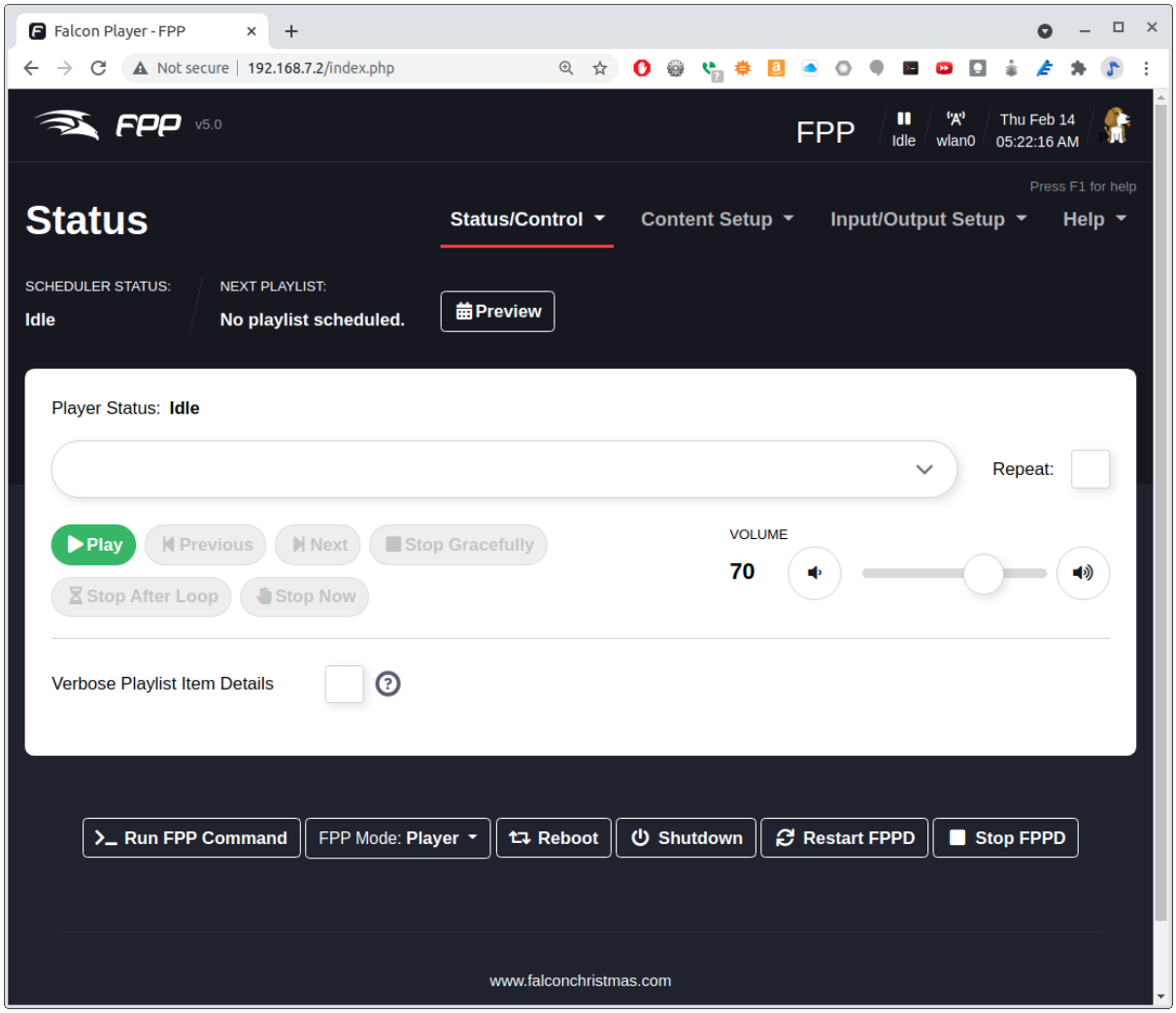


Fig. 15.112: Falcon Play Program Control

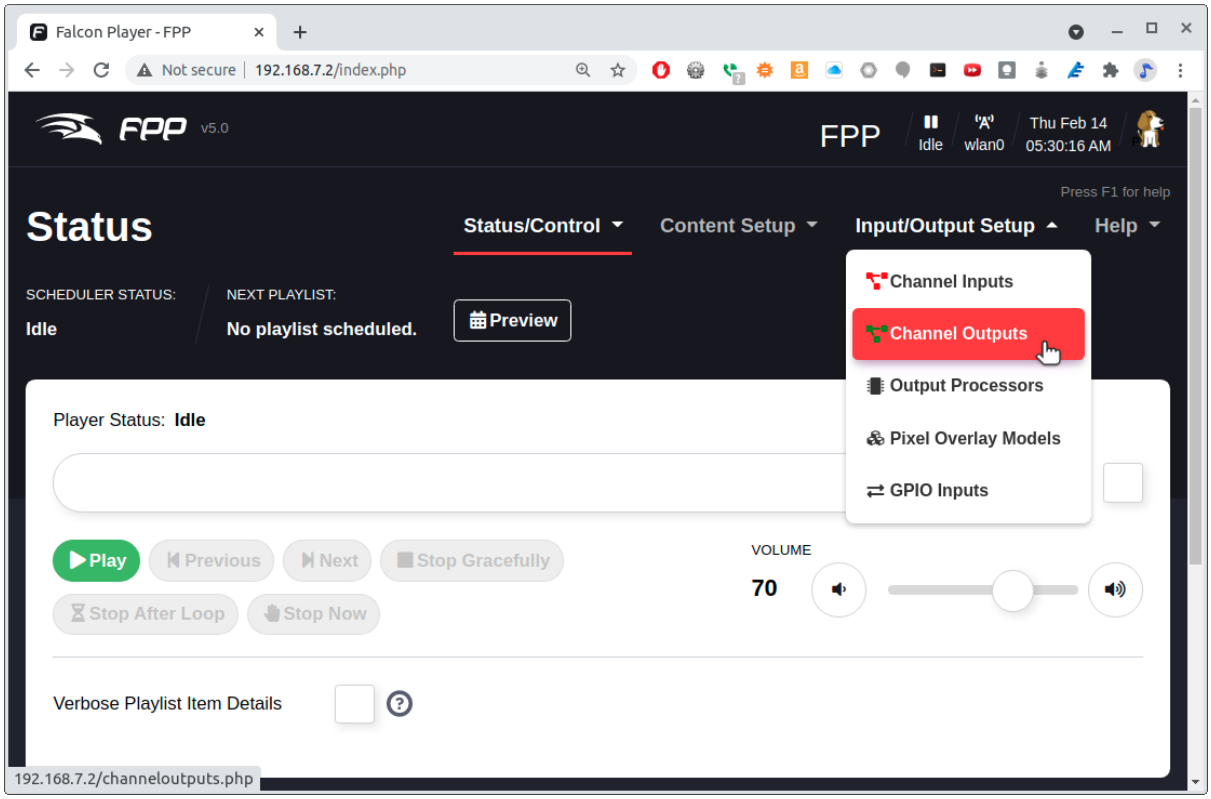


Fig. 15.113: Selecting Channel Outputs

With it you can layout your display visually then assign effects to the various items throughout your sequence. This can be in time to music (with beat-tracking built into xLights) or just however you like. xLights runs on Windows, OSX and Linux

https://xlights.org/

---

xLights can be installed on your host computer (not the Beagle) by following instructions at https://xlights.org/releases/.

Run xLights and you'll see *xLights Setup*.

```
host$ chmod +x xLights-2021.18-x86_64.AppImage
host$ ./xLights-2021.18-x86_64.AppImage
```

We'll walk you through a simple setup to get an animation to display on the RGB Matrix. xLights can use a protocol called E1.31 to send information to the display. Setup xLights by clicking on *Add Ethernet* and entering the values shown in *Setting Up E1.31*.

The **IP Address** is the Bone's address as seen from the host computer. Each LED is one channel, so one RGB LED is three channels. The P5 board has 3*64*32 or 6144 channels. These are grouped into universes of 512 channels each. This gives 6144/512 = 12 universes. See the E.13 documentation for more details.

Your setup should look like *xLights setup for P5 display*. Click the *Save Setup* button to save.

Next click on the **Layout** tab. Click on the *Matrix* button as shown in *Setting up the Matrix Layout*, then click on the black area where you want your matrix to appear.

Layout details for P5 matrix shows the setting to use for the P5 matrix.

All I changed was **# Strings**, **Nodes/String**, **Starting Location** and most importantly, expand **String Properties** and select at **String Type** of **RGB Nodes**. Above the setting you should see that **Start Chan** is 1 and the **End Chan** is 6144, which is the total number of individual LEDs (3*63*32). xLights now knows we are working with a P5 matrix, now on to the sequencer.

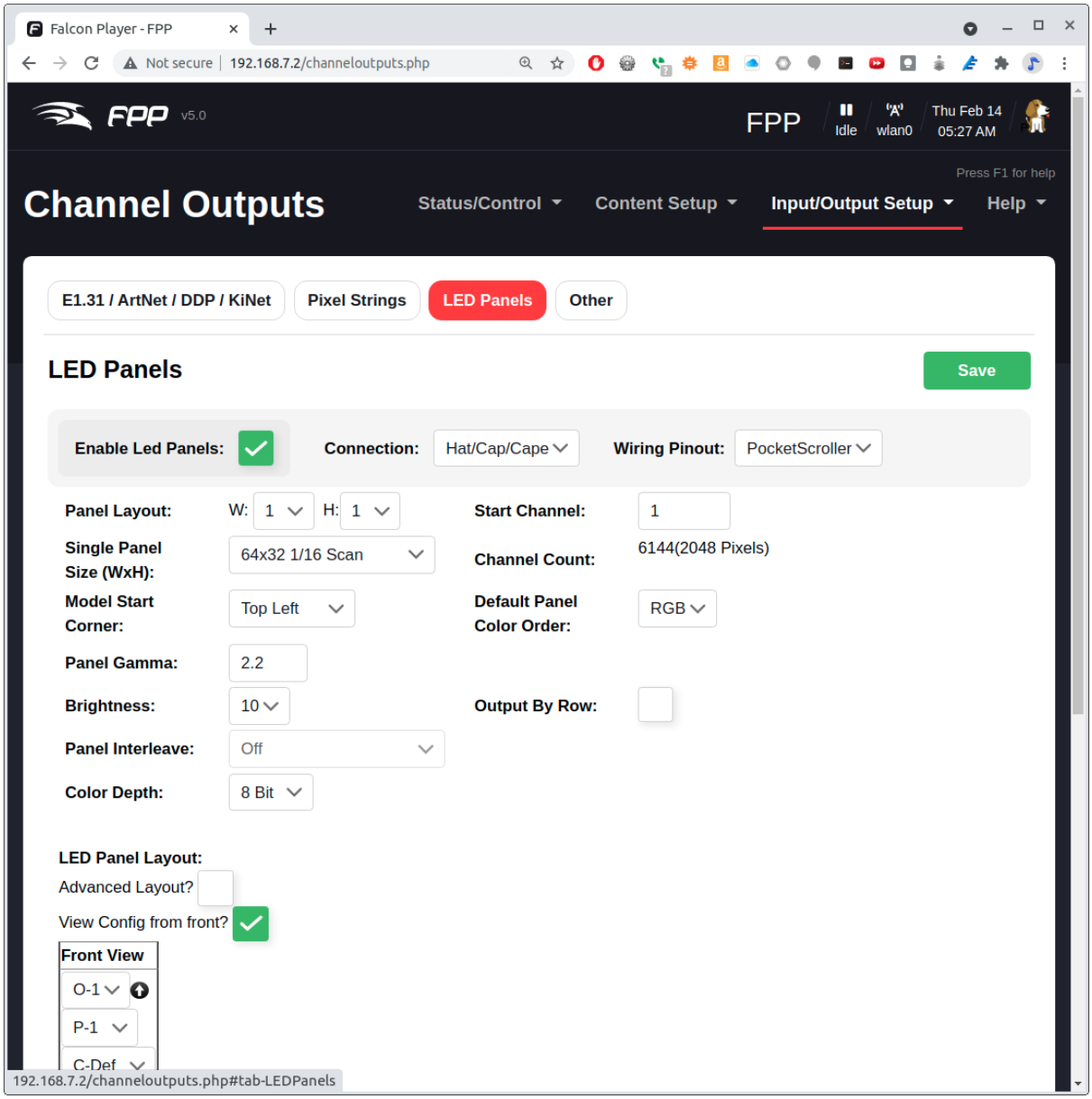


Fig. 15.114: Channel Outputs Settings

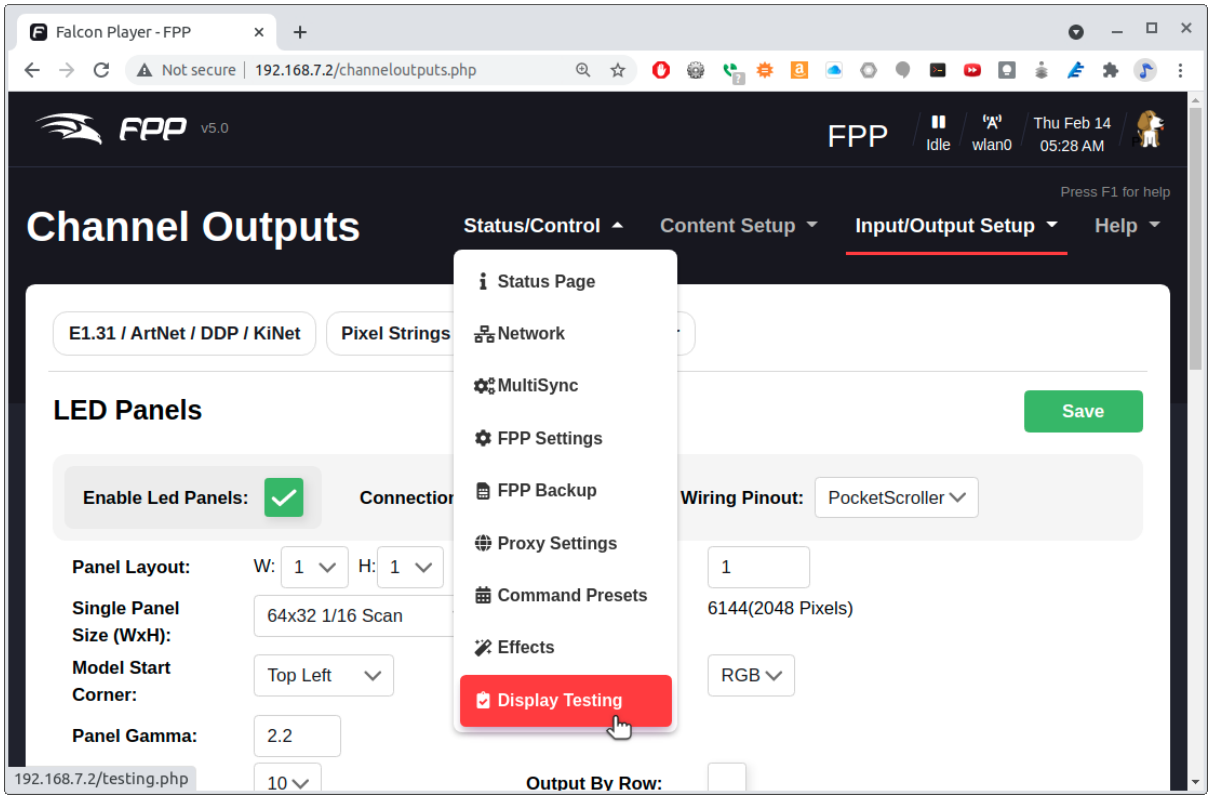


Fig. 15.115: Selecting Display Testing

Now click on the *Sequencer* tab and then click on the **New Sequence** button (*Starting a new sequence*).

Then click on **Animation**, **20fps (50ms)**, and **Quick Start**. Learning how to do sequences is beyond the scope of this cookbook, however I'll shown you how do simple sequence just to be sure xLights is talking to the Bone.

Setting Up E1.31 on the Bone First we need to setup FPP to take input from xLights. Do this by going to the *Input/Output Setup* menu and selecting *Channel Inputs*. Then enter *12* for *Universe Count* and click *set* and you will see *E1.31 Inputs*.

Click on the **Save** button above the table.

Then go to the **Status/Control** menu and select **Status Page**.

Testing the xLights Connection The Bone is now listening for commands from xLights via the E1.31 protocol. A quick way to verify everything is t o return to xLights and go to the *Tools* menu and select **Test** (*xLights test page*).

Click the box under **Select channels...**, click **Output to lights** and select **Twinkle 50%**. You matrix should have a colorful twinkle pattern (*xLights Twinkle test pattern*).

A Simple xLights Sequence Now that the xLights to FPP link is tested you can generate a sequence to play. Close the Test window and click on the **Sequencer** tab. Then drag an effect from the **Effects** box to the timeline that below it. Drop it to the right of the **Matrix** label (*Drag an effect to the timeline*). The click *Output To Lights* which is the yellow lightbulb to the right on the top toolbar. Your matrix should now be displaying your effect.

The setup requires the host computer to send the animation data to the Bone. The next section shows how to save the sequence and play it on the Bone standalone.

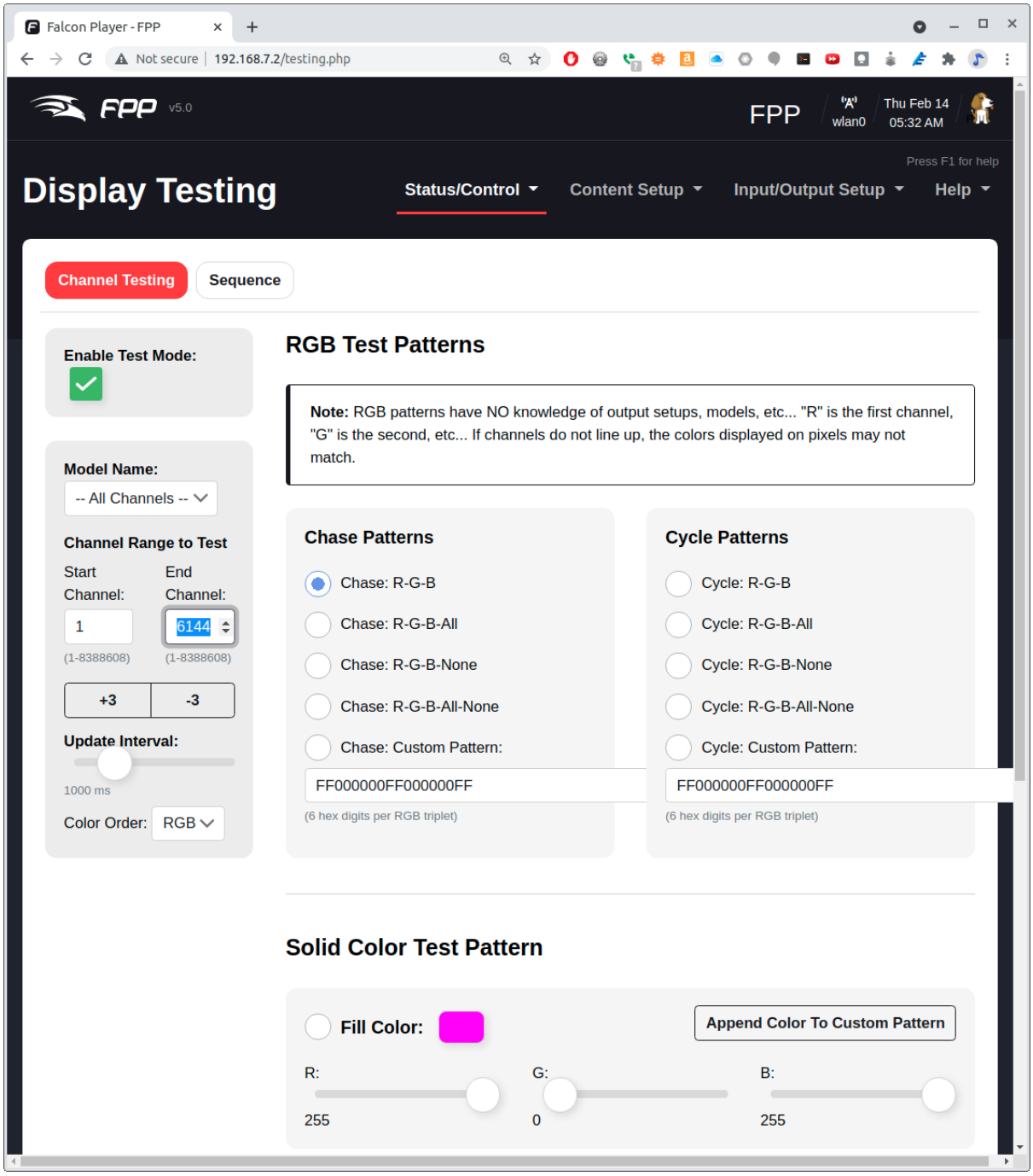


Fig. 15.116: Display Testing Options

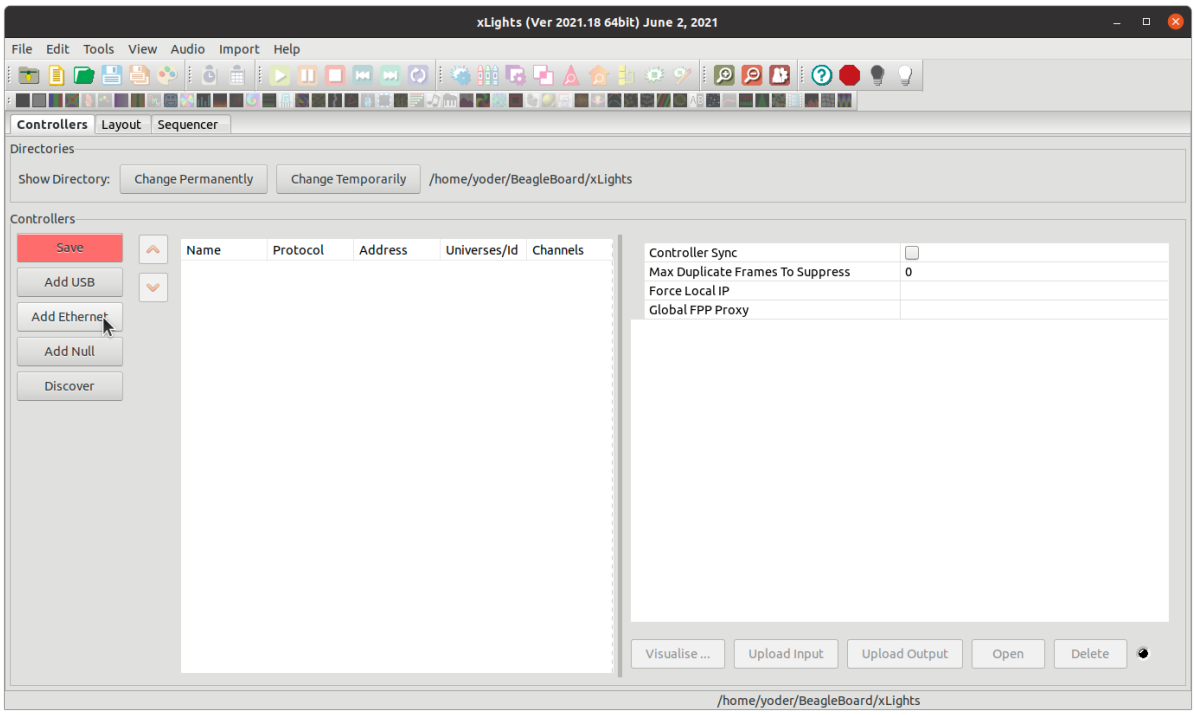


Fig. 15.117: xLights Setup

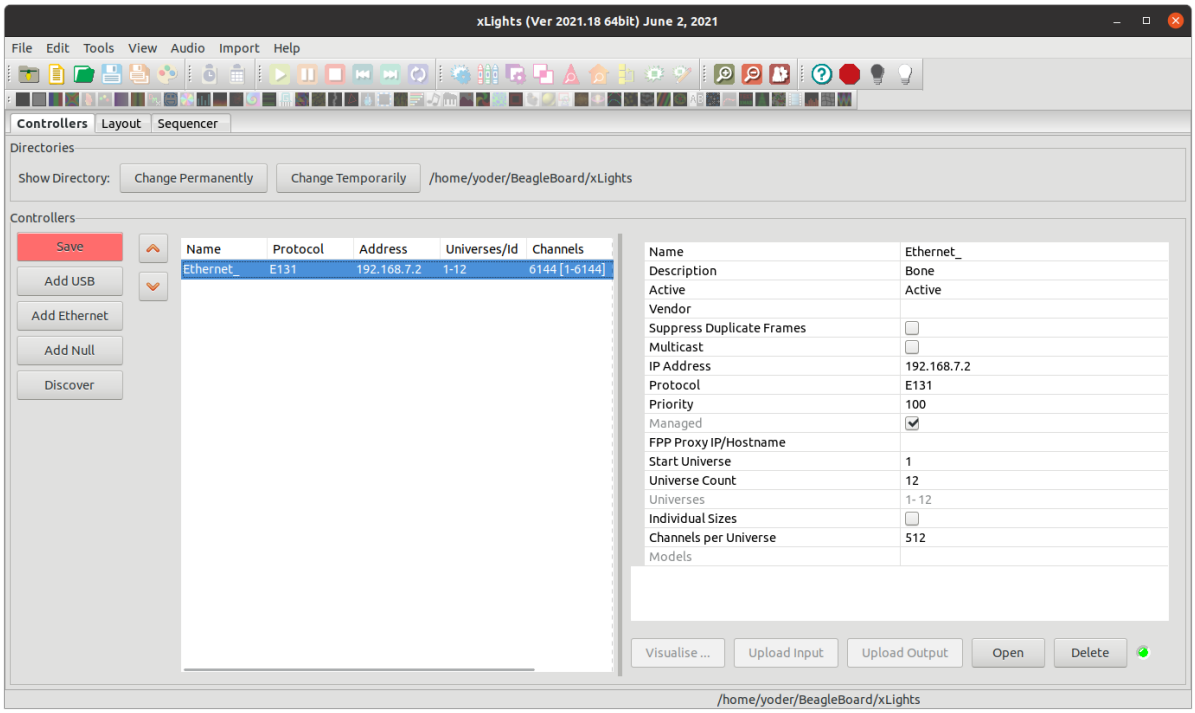


Fig. 15.118: Setting Up E1.31

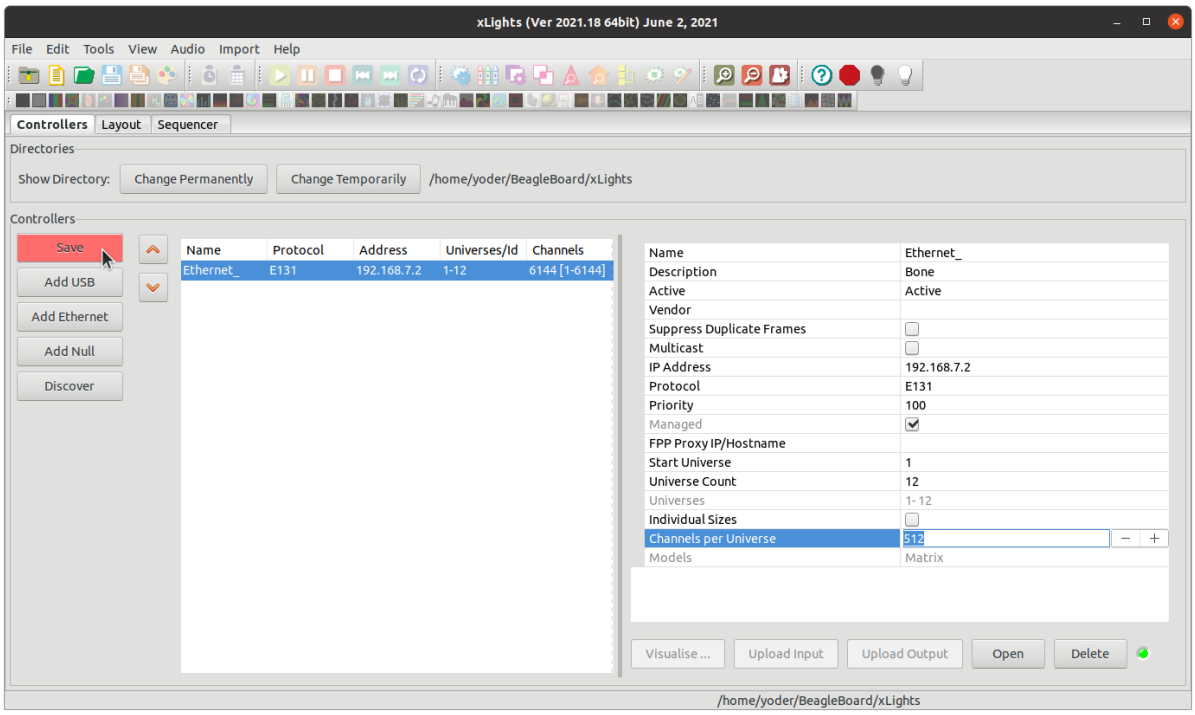


Fig. 15.119: xLights setup for P5 display

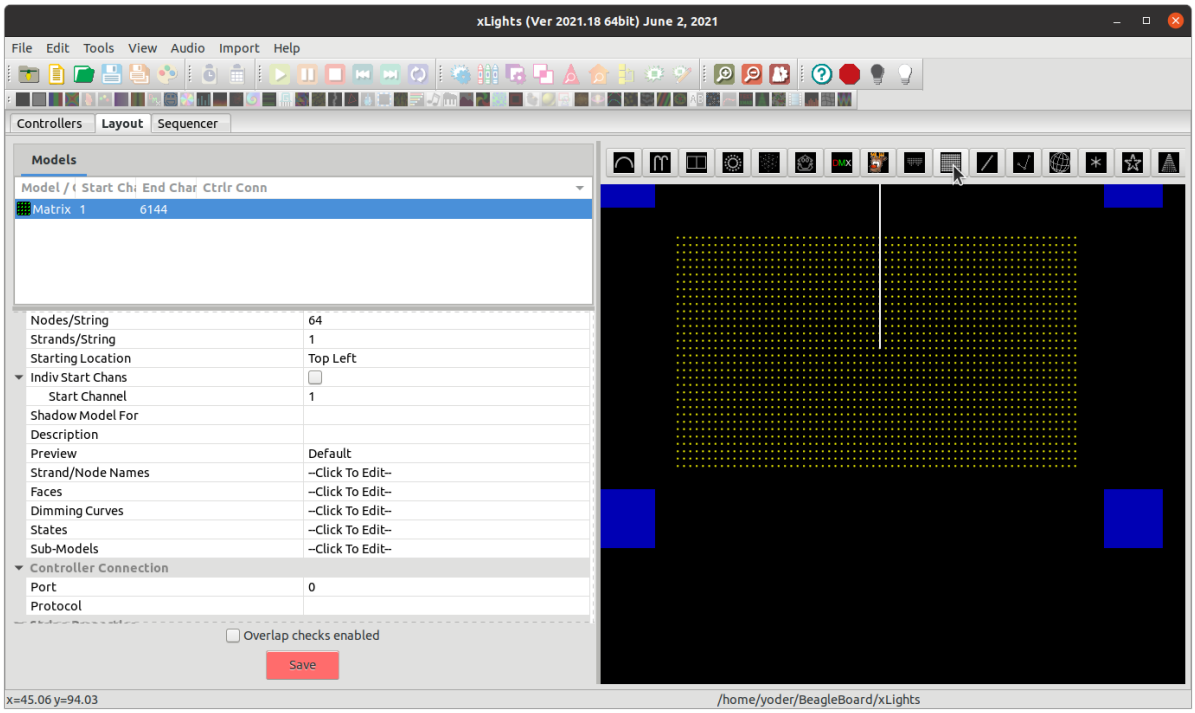


Fig. 15.120: Setting up the Matrix Layout

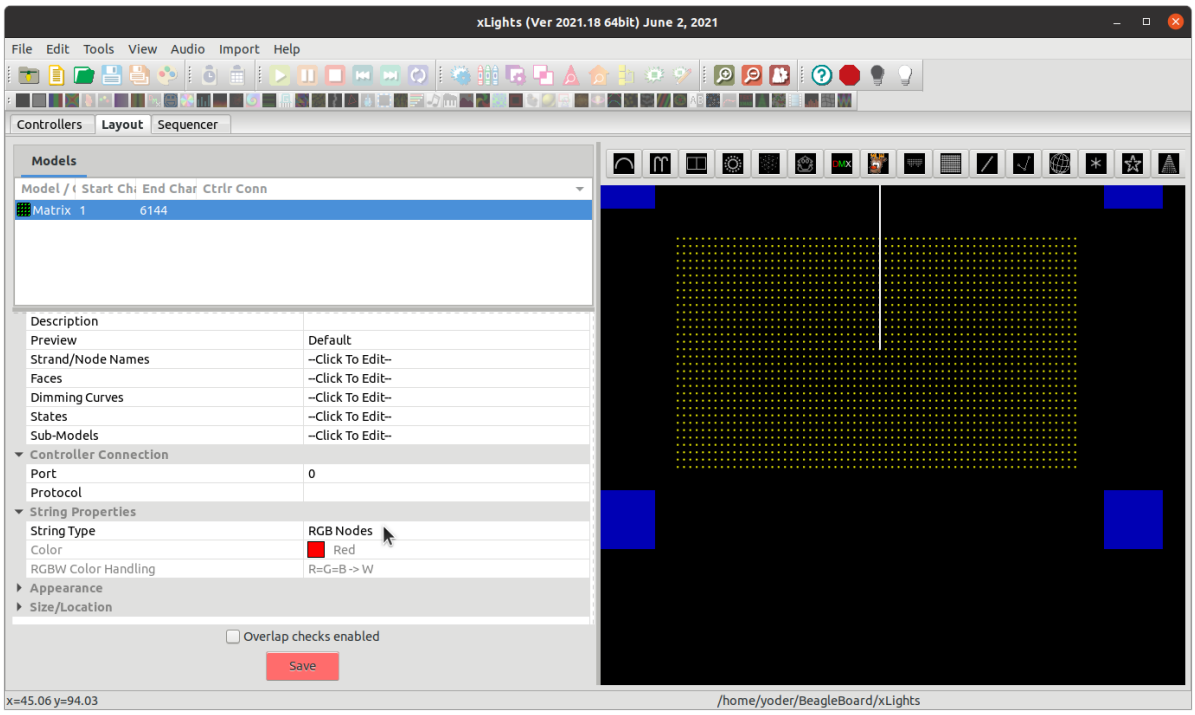


Fig. 15.121: Layout details for P5 matrix

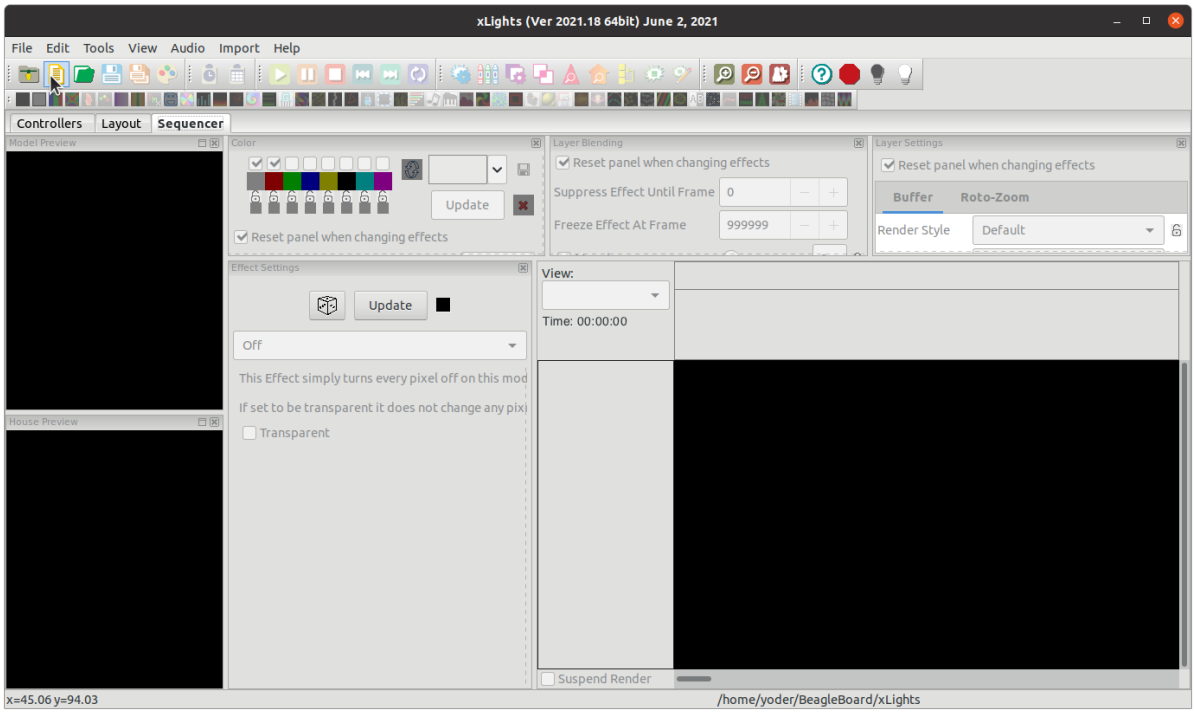


Fig. 15.122: Starting a new sequence

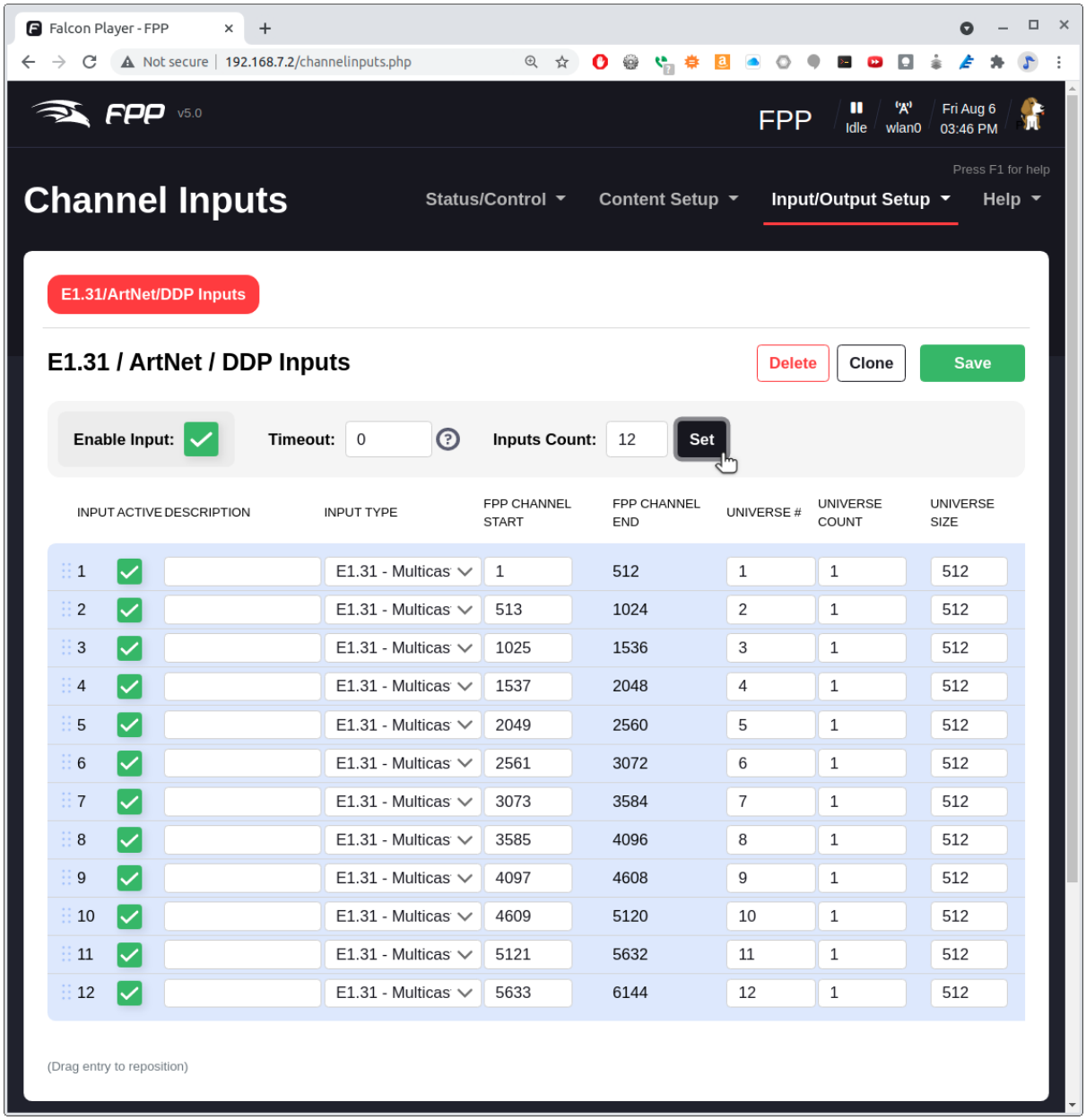


Fig. 15.123: E1.31 Inputs

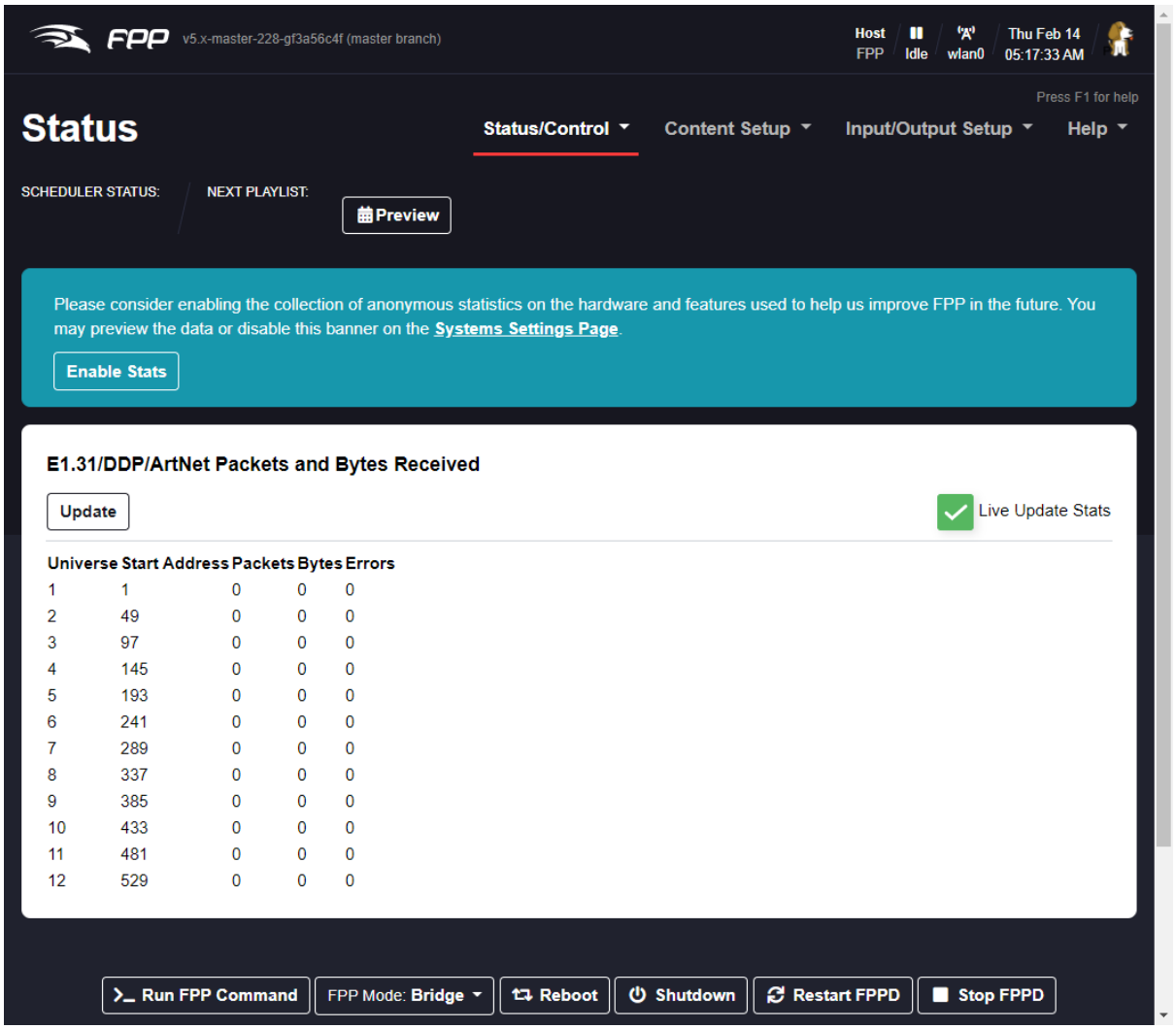


Fig. 15.124: Bridge Mode

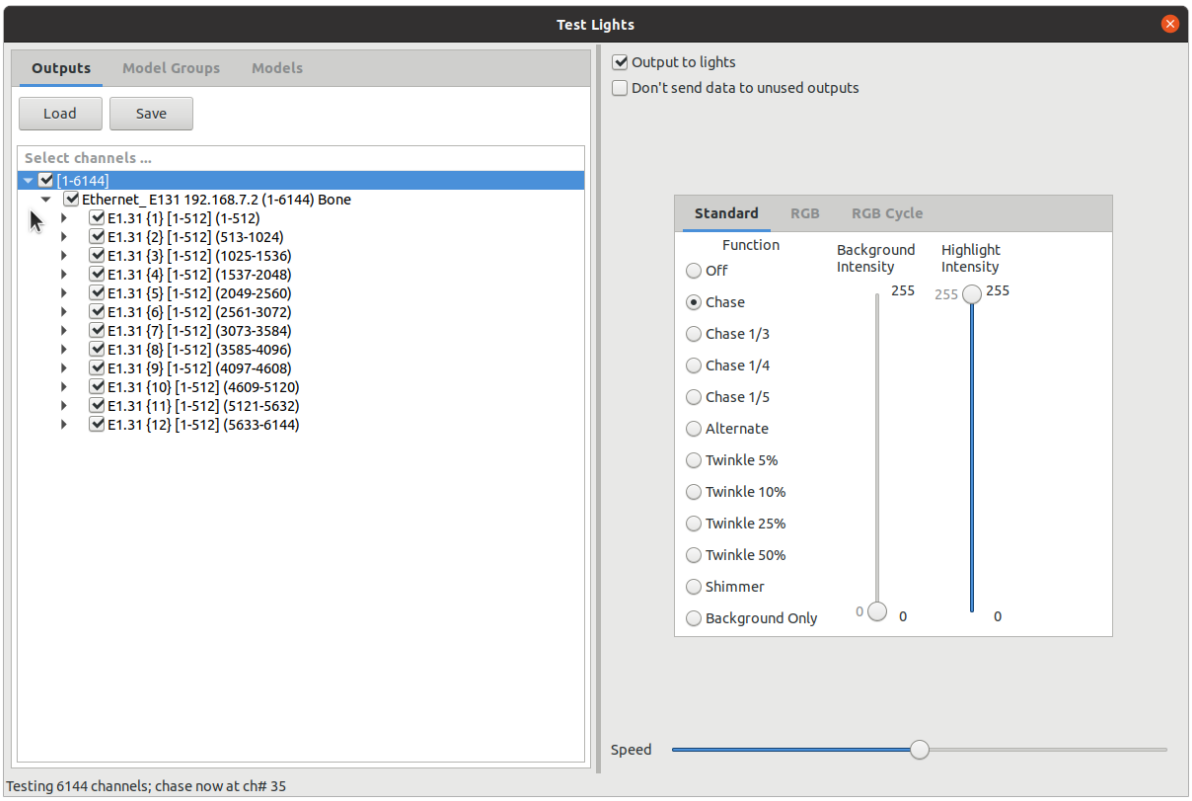


Fig. 15.125: xLights test page

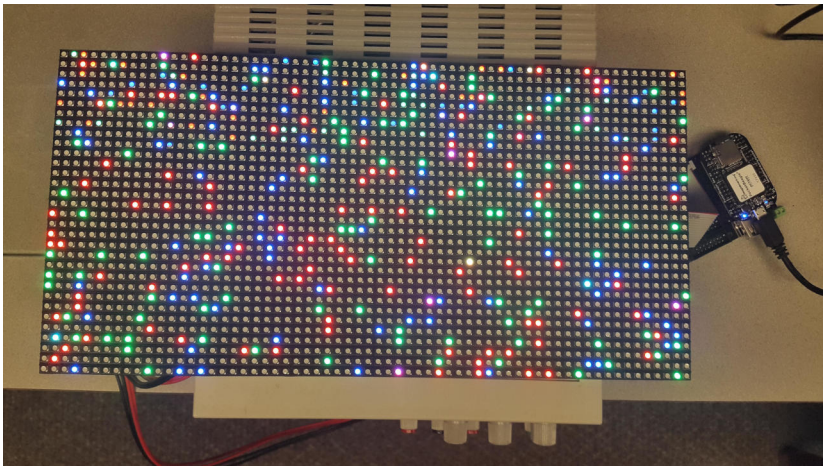


Fig. 15.126: xLights Twinkle test pattern

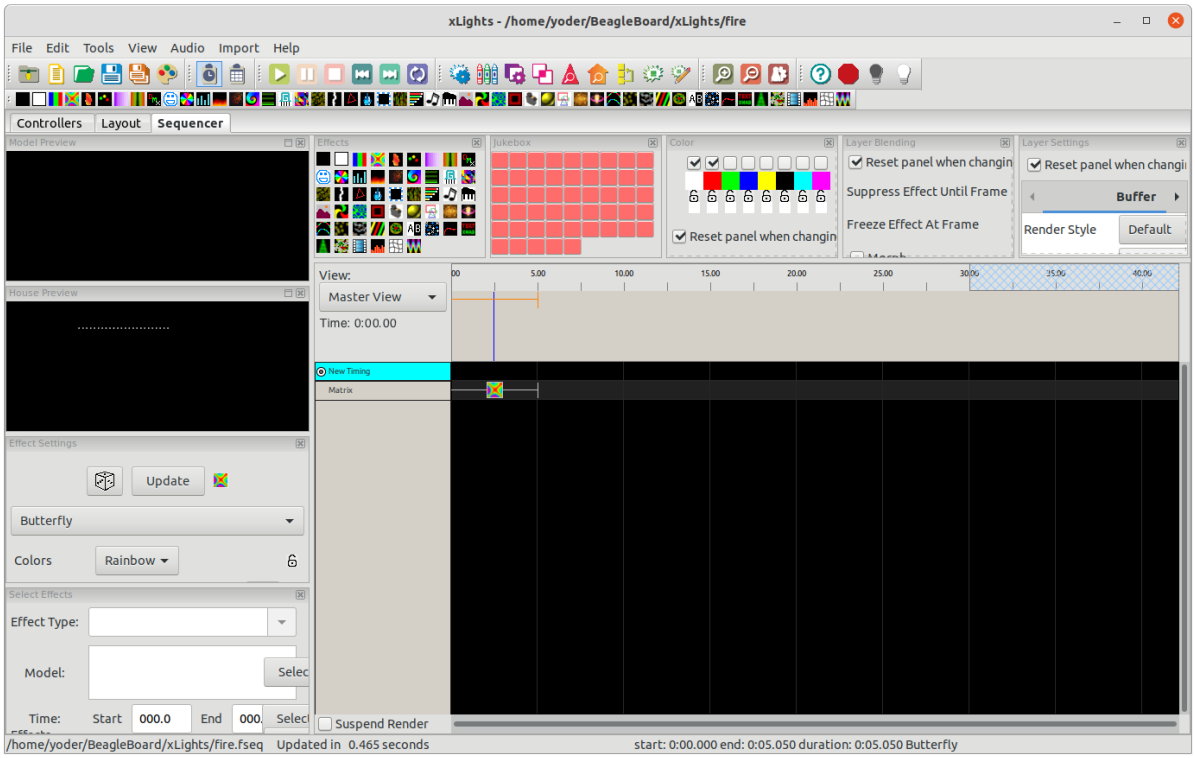


Fig. 15.127: Drag an effect to the timeline

Saving a Sequence and Playing it Standalone In xLights save your sequence by hitting Ctrl-S and giving it a name. I called mine *fire* since I used a fire effect. Now, switch back to FPP and select the *Content Setup* menu and select *File Manager*. Click the black *Select Files* button and select your sequence file that ends in *.fseq* (*FPP file manager*).

Once your sequence is uploaded, got to **Content Setup** and select **Playlists**. Enter you playlist name (I used **fire**) and click **Add**. Then click **Add a Sequence/Entry** and select **Sequence Only** (*Adding a new playlist to FPP*), then click **Add**.

Be sure to click **Save Playlist** on the right. Now return to **Status/Control** and **Status Page** and make sure **FPPD Mode:** is set to **Standalone**. You should see your playlist. Click the **Play** button and your sequence will play.

The beauty of the PRU is that the Beagle can play a detailed sequence at 20 frames per second and the ARM processor is only 15% used. The PRUs are doing all the work.

simpPRU – A python-like language for programming the PRUs simpPRU is a simple, python-like programming language designed to make programming the PRUs easy. It has detailed documentation and many examples.

---

information

simpPRU is a procedural programming language that is statically typed. Variables and functions must be assigned data types during compilation. It is type-safe, and data types of variables are decided during compilation. simPRU codes have a +.sim+ extension. simpPRU provides a console app to use Remoteproc functionality.

https://simppru.readthedocs.io/en/latest/

You can build simpPRU from source, more easily just install it. On the Beagle run:

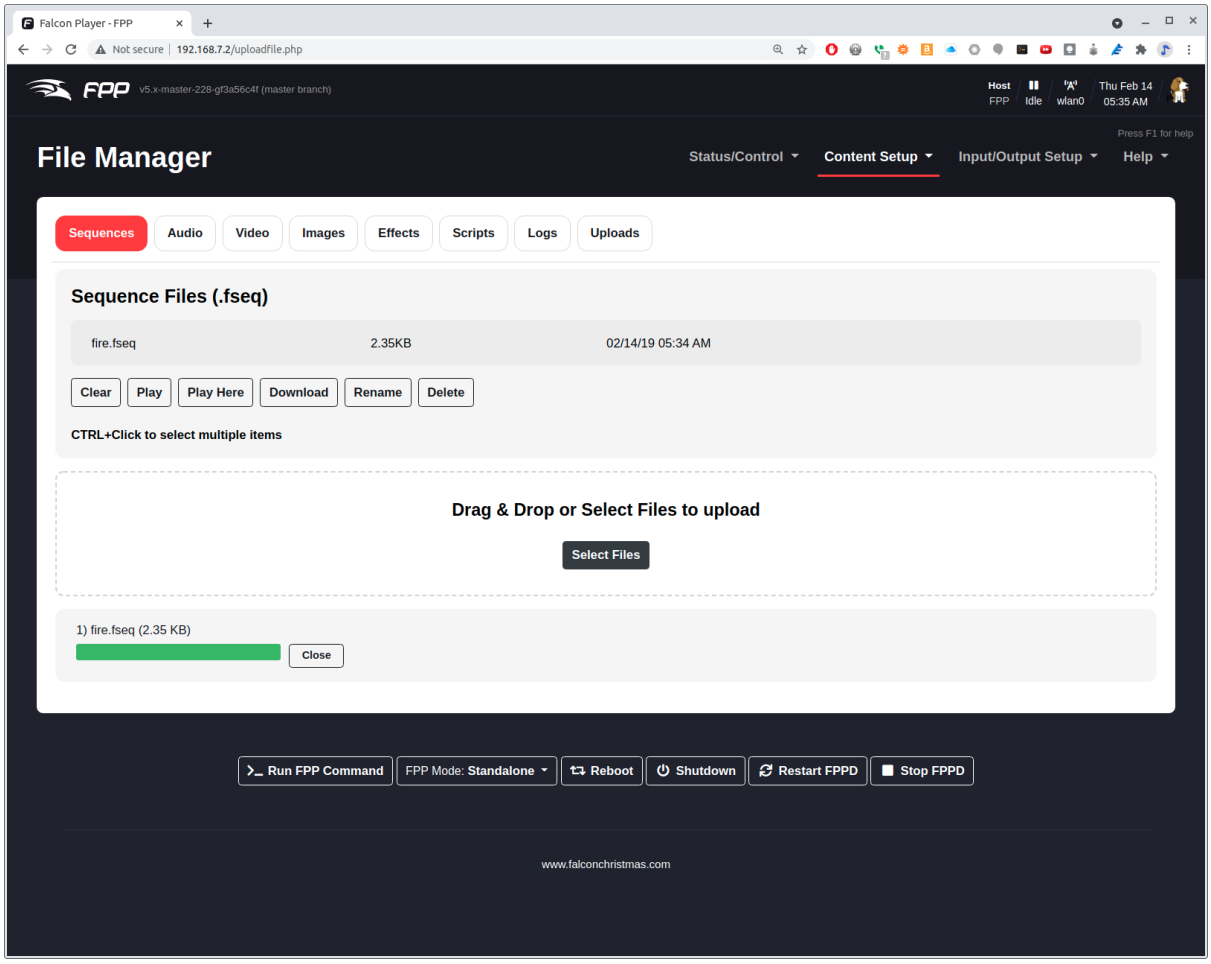


Fig. 15.128: FPP file manager

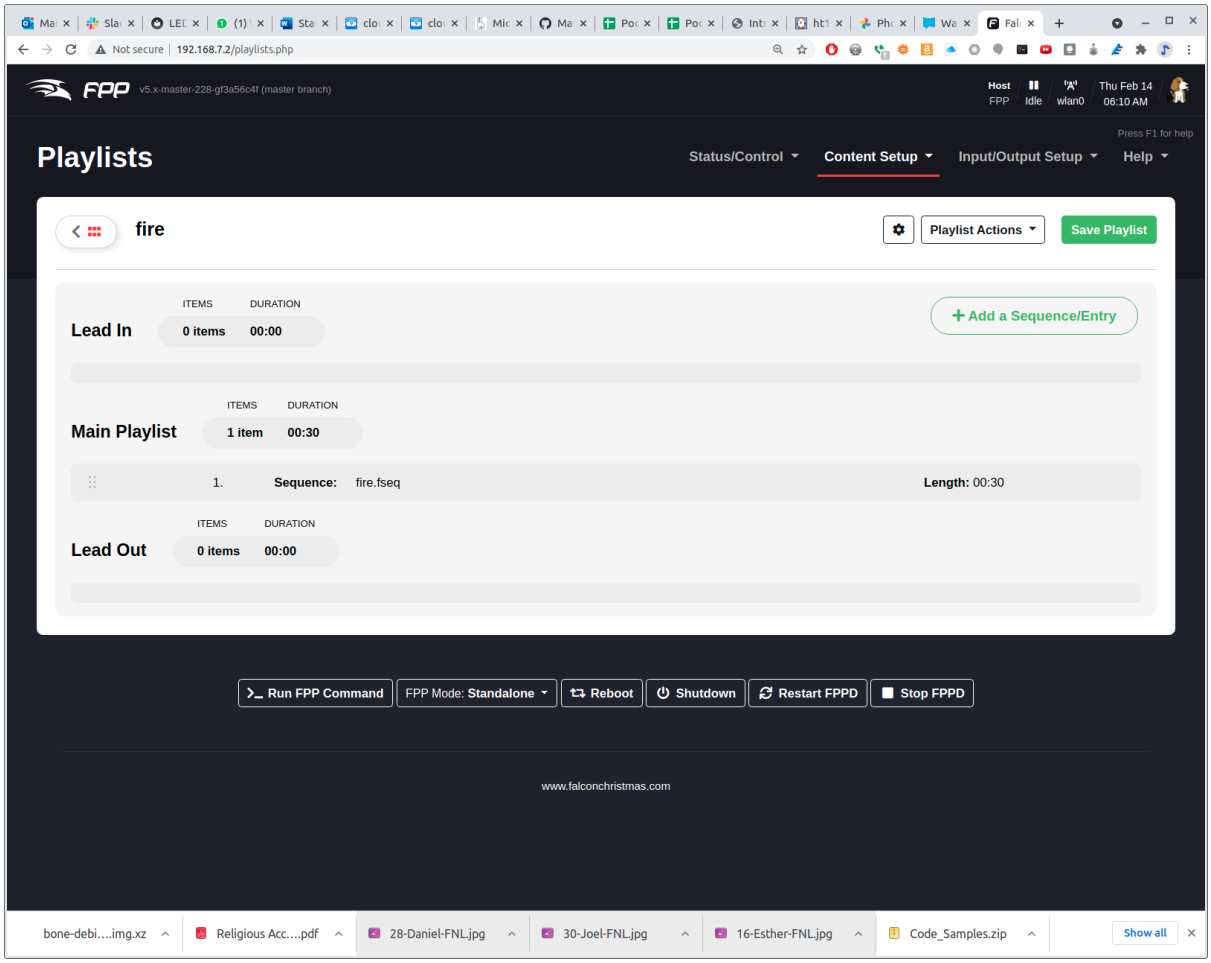


Fig. 15.129: Adding a new playlist to FPP

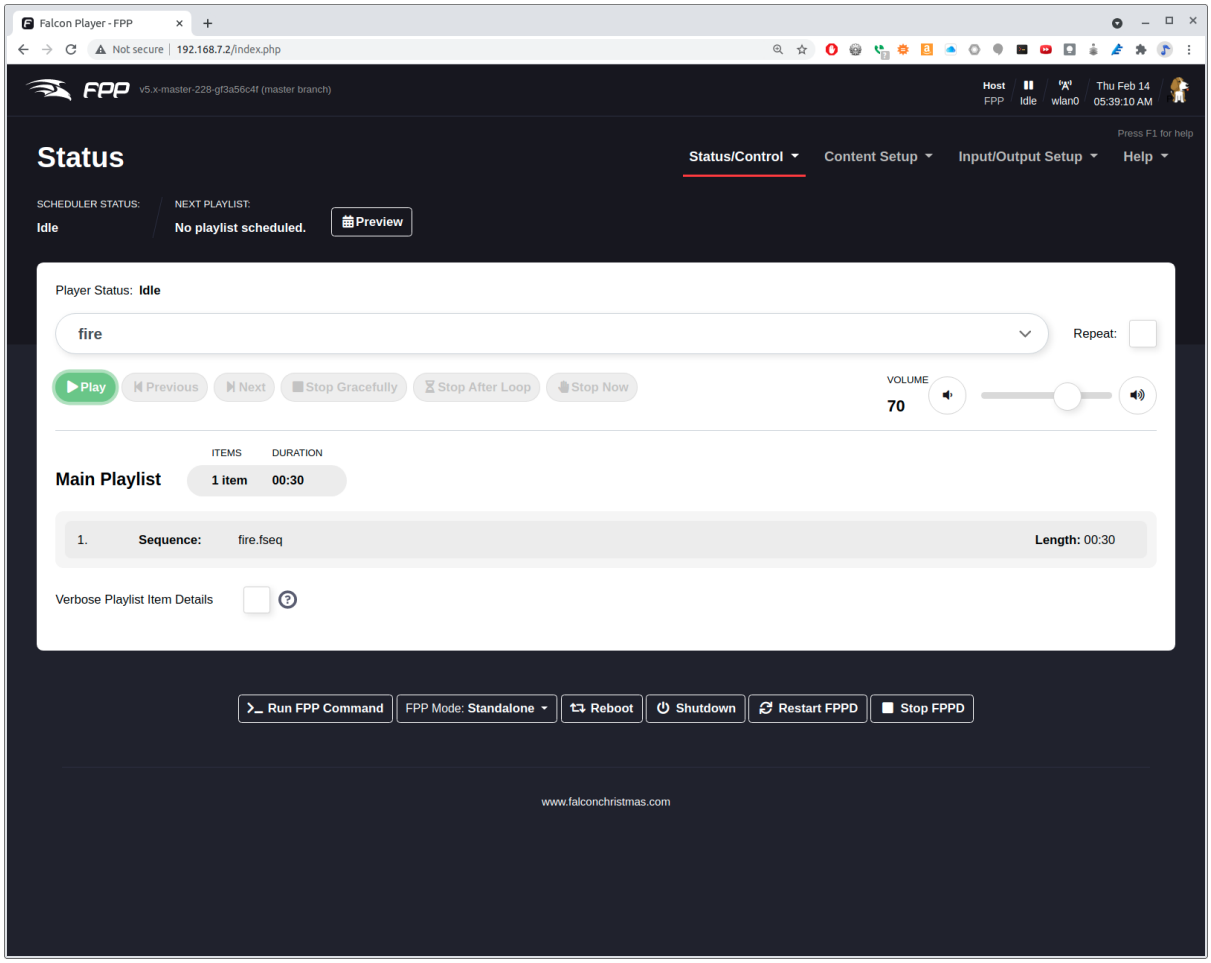


Fig. 15.130: Adding a new playlist to FPP

```
bone$ wget https://github.com/VedantParanjape/simpPRU/releases/download/1.4/
↪simppru-1.4-armhf.deb
bone$ sudo dpkg -i simppru-1.4-armhf.deb
bone$ sudo apt update
bone$ sudo apt install gcc-pru
```

Now, suppose you wanted to run the LED blink example which is reproduced here.

Listing 15.69: LED Blink (blink.sim)

```
/* From: https://simppru.readthedocs.io/en/latest/examples/led_blink/ */
while : 1 == 1 {
    digital_write(P1_31, true);
    delay(250);     /* Delay 250 ms */
    digital_write(P1_31, false);
    delay(250);
}
```

`blink.sim`

Just run simppru

```
bone$ simppru blink.sim --load
Detected TI AM335x PocketBeagle
inside while
[4] : setting P1_31 as output

Current mode for P1_31 is:     pruout
```

Detected TI AM335x PocketBeagle The +–load+ flag caused the compiled code to be copied to +/lib/firmware+. To start just do:

```
bone$ cd /dev/remoteproc/pruss-core0/
bone$ ls
device  firmware  name  power  state  subsystem  uevent
bone$ echo start > state
bone$ cat state
running
```

Your LED should now be blinking.

Check out the many examples (https://simppru.readthedocs.io/en/latest/examples/led_blink/).

MachineKit MachineKit is a platform for machine control applications. It can control machine tools, robots, or other automated devices. It can control servo motors, stepper motors, relays, and other devices related to machine tools.

---

information

Machinekit is portable across a wide range of hardware platforms and real-time environments, and delivers excellent performance at low cost. It is based on the HAL component architecture, an intuitive and easy to use circuit model that includes over 150 building blocks for digital logic, motion, control loops, signal processing, and hardware drivers. Machinekit supports local and networked UI options, including ubiquitous platforms like phones or tablets.

http://www.machinekit.io/about/

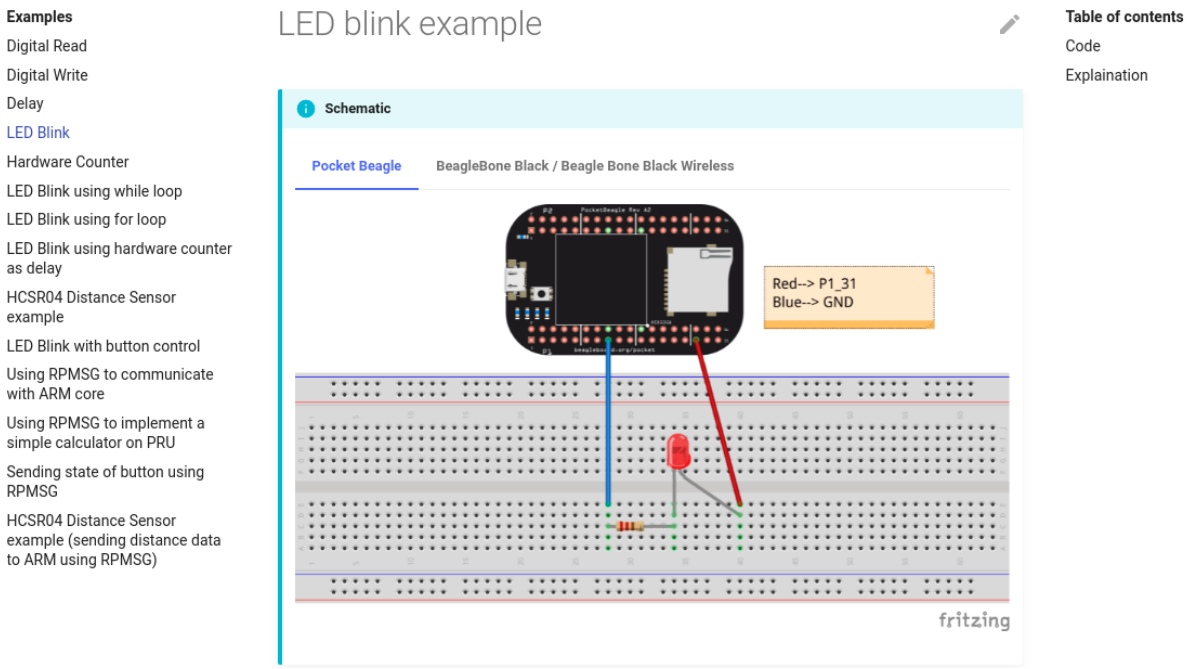


Fig. 15.131: simpPRU Examples

ArduPilot ArduPilot is a open source autopilot system supporting multi-copters, traditional helicopters, fixed wing aircraft and rovers. ArduPilot runs on a many hardware platforms including the BeagleBone Black and the BeagleBone Blue.

---

information

Ardupilot is the most advanced, full-featured and reliable open source autopilot software available. It has been developed over 5+ years by a team of diverse professional engineers and computer scientists. It is the only autopilot software capable of controlling any vehicle system imaginable, from conventional airplanes, multirotors, and helicopters, to boats and even submarines. And now being expanded to feature support for new emerging vehicle types such as quad-planes and compound helicopters.

Installed in over 1,000,000 vehicles world-wide, and with its advanced data-logging, analysis and simulation tools, Ardupilot is the most tested and proven autopilot software. The open-source code base means that it is rapidly evolving, always at the cutting edge of technology development. With many peripheral suppliers creating interfaces, users benefit from a broad ecosystem of sensors, companion computers and communication systems. Finally, since the source code is open, it can be audited to ensure compliance with security and secrecy requirements.

The software suite is installed in aircraft from many OEM UAV companies, such as 3DR, jDrones, PrecisionHawk, AgEagle and Kespry. It is also used for testing and development by several large institutions and corporations such as NASA, Intel and Insitu/Boeing, as well as countless colleges and universities around the world.

http://www.machinekit.io/about/

15.2.2 Getting Started

We assume you have some experience with the Beagle and are here to learn about the PRU. This chapter discusses what Beagles are out there, how to load the latest software image on your Beagle, how to run the Visual Studio Code IDE and how to blink an LED. ======= latest software image on your Beagle, how to run the Visual Studio Code (VS Code) IDE and how to blink an LED.

If you already have your Beagle and know your way around it, you can find the code at https://git.beagleboard.

org/beagleboard/pru-cookbook-code and book contents at https://git.beagleboard.org/docs/docs.beagleboard.io under the books/pru-cookbook directory.

Selecting a Beagle

Problem Which Beagle should you use?

Solution http://beagleboard.org/boards lists the many Beagles from which to choose. Here we'll give examples for the venerable BeagleBone Black, the robotics BeagleBone Blue, tiny PockeBeagle and the powerful AI. All the examples should also run on the other Beagles too.

Discussion

BeagleBone Black If you aren't sure which Beagle to use, it's hard to go wrong with the BeagleBone Black. It's the most popular member of the open hardware Beagle family.

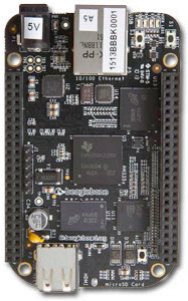


Fig. 15.132: BeagleBone Black

The Black has:

- AM335x 1GHz ARM® Cortex-A8 processor
- 512MB DDR3 RAM
- 4GB 8-bit eMMC on-board flash storage
- 3D graphics accelerator
- NEON floating-point accelerator
- 2x PRU 32-bit microcontrollers
- USB client for power & communications
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers

See http://beagleboard.org/black for more details.

BeagleBone Blue The Blue is a good choice if you are doing robotics.

The Blue has everything the Black has except it has no Ethernet and no HDMI. But it also has:

- Wireless: 802.11bgn, Bluetooth 4.1 and BLE
- Battery support: 2-cell LiPo with balancing, LED state-of-charge monitor



Fig. 15.133: BeagleBone Blue

- Charger input: 9-18V
- Motor control: 8 6V @ 4A servo out, 4 bidirectional DC motor out, 4 quadrature encoder in
- Sensors: 9 axis IMU (accels, gyros, magnetometer), barometer, thermometer
- User interface: 11 user programmable LEDs, 2 user programmable buttons

In addition you can mount the Blue on the EduMIP kit as shown in *BeagleBone Blue EduMIP Kit* to get a balancing robot.

https://www.hackster.io/53815/controlling-edumip-with-ni-labview-2005f8 shows how to assemble the robot and control it from LabVIEW.

PocketBeagle The PocketBeagle is the smallest member of the Beagle family. It is an ultra-tiny-yet-complete Beagle that is software compatible with the other Beagles.

The Pocket is based on the same processor as the Black and Blue and has:

- 8 analog inputs
- 44 digital I/Os and
- numerous digital interface peripherals

See http://beagleboard.org/pocket for more details.

BeagleBone AI If you want to do deep learning, try the BeagleBone AI.

The AI has:

- Dual Arm® Cortex®-A15 microprocessor subsystem
- 2 C66x floating-point VLIW DSPs
- 2.5MB of on-chip L3 RAM
- 2x dual Arm® Cortex®-M4 co-processors
- 4x Embedded Vision Engines (EVEs)
- 2x dual-core Programmable Real-Time Unit and Industrial Communication SubSystem (PRU-ICSS)
- 2D-graphics accelerator (BB2D) subsystem
- Dual-core PowerVR® SGX544™ 3D GPU
- IVA-HD subsystem (4K @ 15fps encode and decode support for H.264, 1080p60 for others)



Fig. 15.134: BeagleBone Blue EduMIP Kit

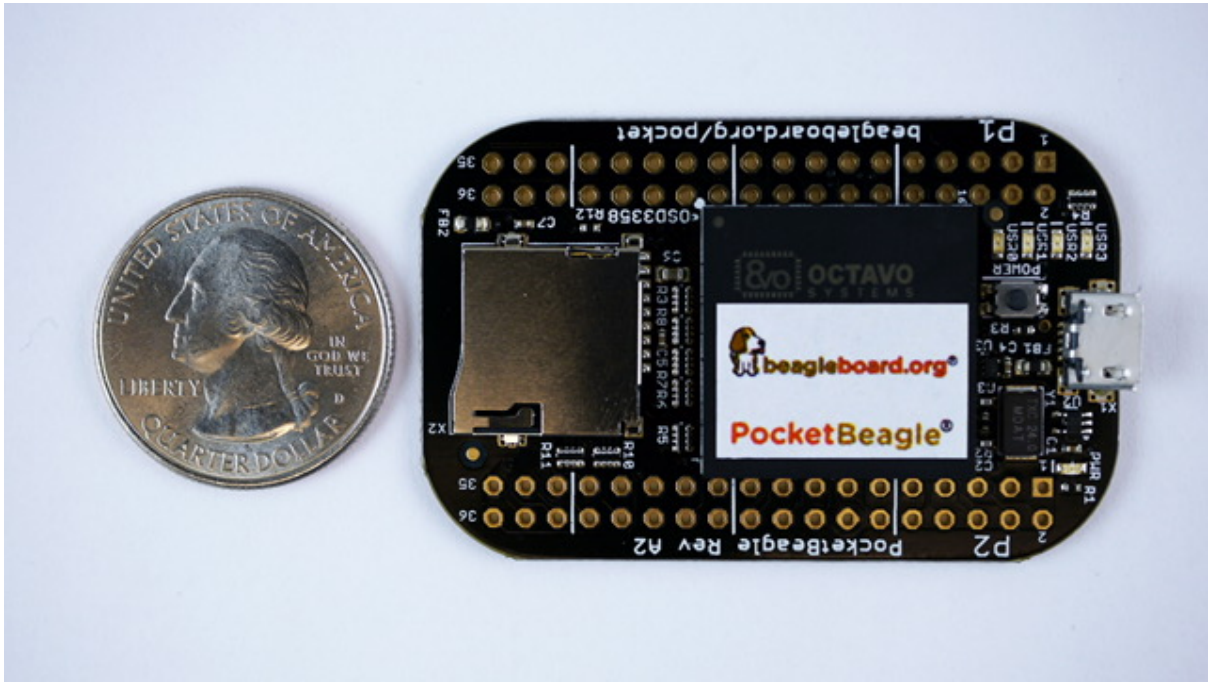


Fig. 15.135: PocketBeagle

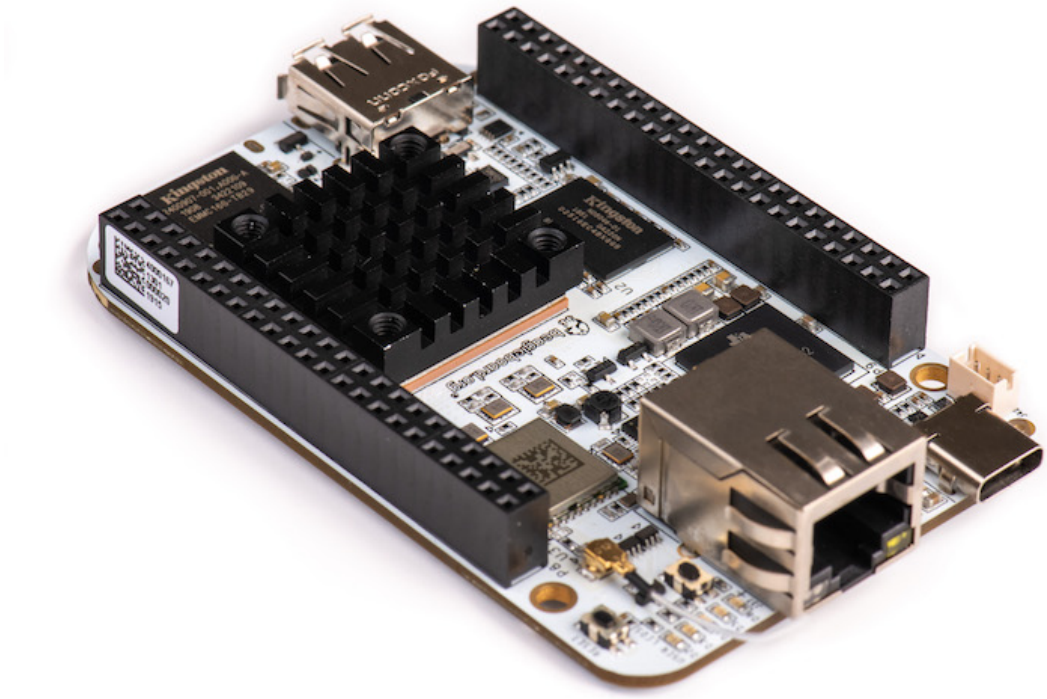


Fig. 15.136: BeagleBone AI

- BeagleBone Black mechanical and header compatibility
- 1GB RAM and 16GB on-board eMMC flash with high-speed interface
- USB type-C for power and superspeed dual-role controller; and USB type-A host
- Gigabit Ethernet, 2.4/5GHz WiFi, and Bluetooth
- microHDMI
- Zero-download out-of-box software experience with Debian GNU/Linux

Installing the Latest OS on Your Bone

Problem You want to find the latest version of Debian that is available for your Bone.

Solution On your host computer open a browser and go to http://www.beagleboard.org/distros.

This shows you two current choices of recent Debian images, one for the BeagleBone AI (AM5729 Debian 10.3 2020-04-06 8GB SD IoT TIDL) and one for all the other Beagles (AM3358 Debian 10.3 2020-04-06 4GB SD IoT). Download the one for your Beagle.

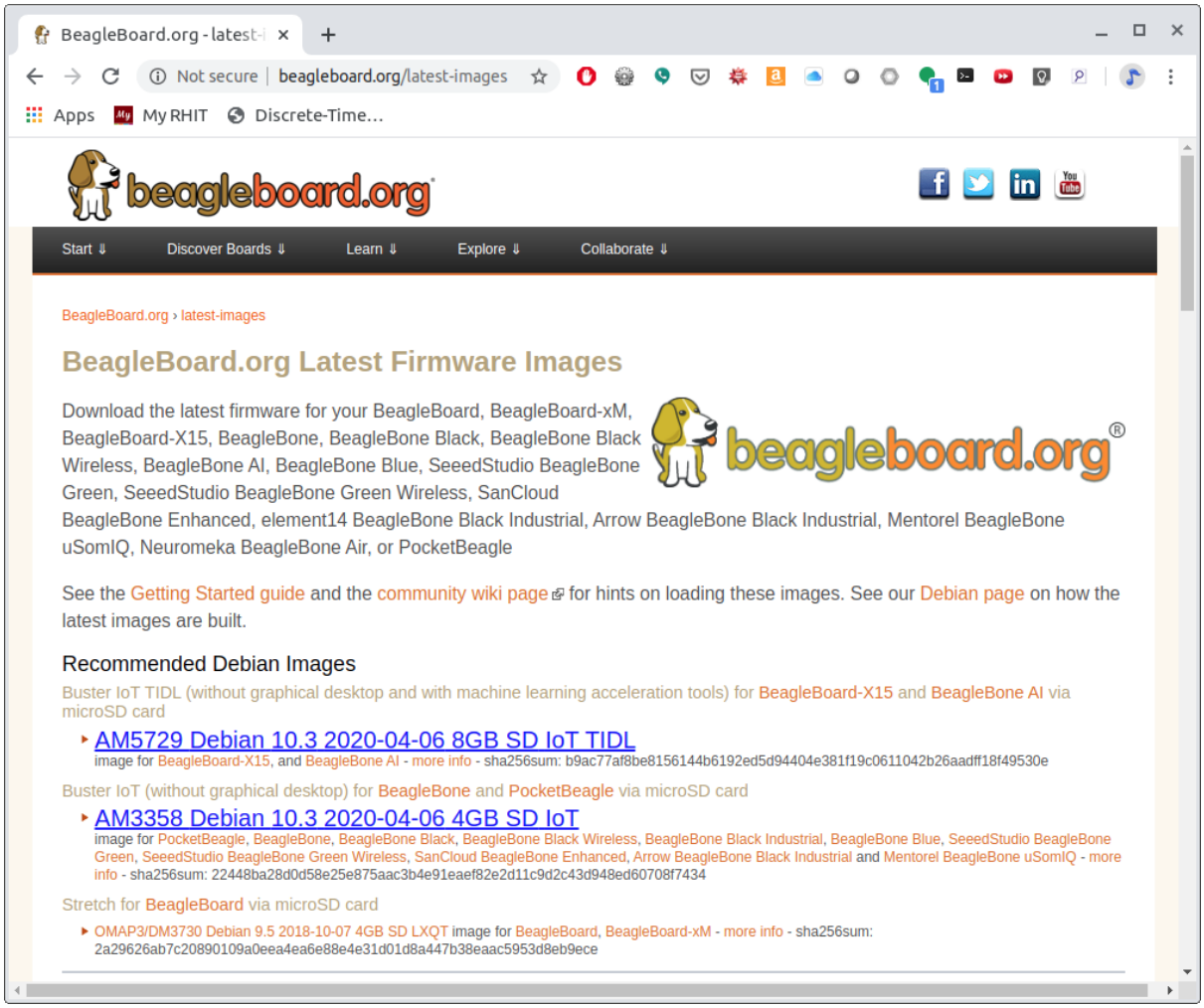


Fig. 15.137: Latest Debian images

It contains all the packages we'll need.

Flashing a Micro SD Card

Problem I've downloaded the image and need to flash my micro SD card.

Solution Get a micro SD card that has at least 4GB and preferably 8GB.

There are many ways to flash the card, but the best seems to be Etcher by https://www.balena.io/. Go to https://www.balena.io/etcher/ and download the version for your host computer. Fire up Etcher, select the image you just downloaded (no need to uncompress it, Etcher does it for you), select the SD card and hit the *Flash* button and wait for it to finish.

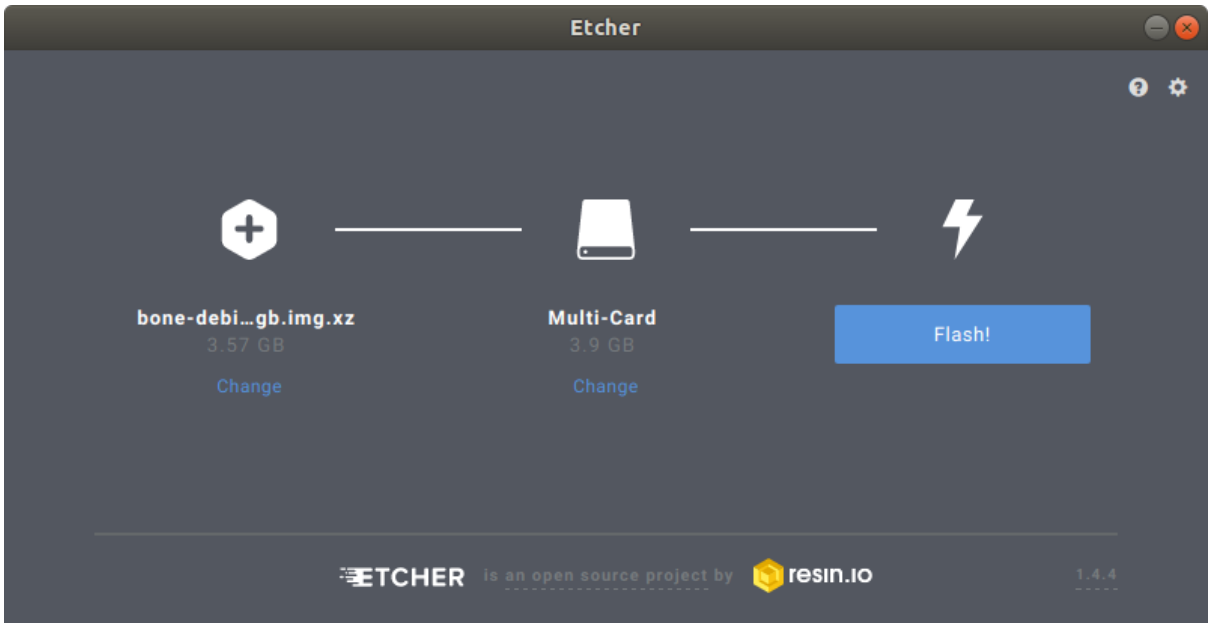


Fig. 15.138: Etcher

Once the SD is flashed, insert it in the Beagle and power it up.

Visual Studio Code IDE

Problem How do I manage and edit my files?

Solution The image you downloaded includes Visual Studio Code, a web-based integrated development environment (IDE) as shown in *Visual Studio Code IDE*.

Just point the browser on your host computer to http://192.168.7.2:3000 and start exploring. You may also want to upgrade bb-code-server to pull in the latest updates. Another route to take is to apply this command to boot the service called bb-code-server.

sudo systemctl start bb-code-server.service

If you want the files in your home directory to appear in the tree structure click the settings gear and select *Show Home in Favorites* as shown in *Visual Studio Code Showing Home files*.

Just point the browser on your host computer to http://192.168.7.2:3000 and start exploring.

If you want to edit files beyond your home directory you can link to the root file system by:

```
bone:~$ cd
bone:~$ ln -s / root
bone:~$ cd root
```

(continues on next page)

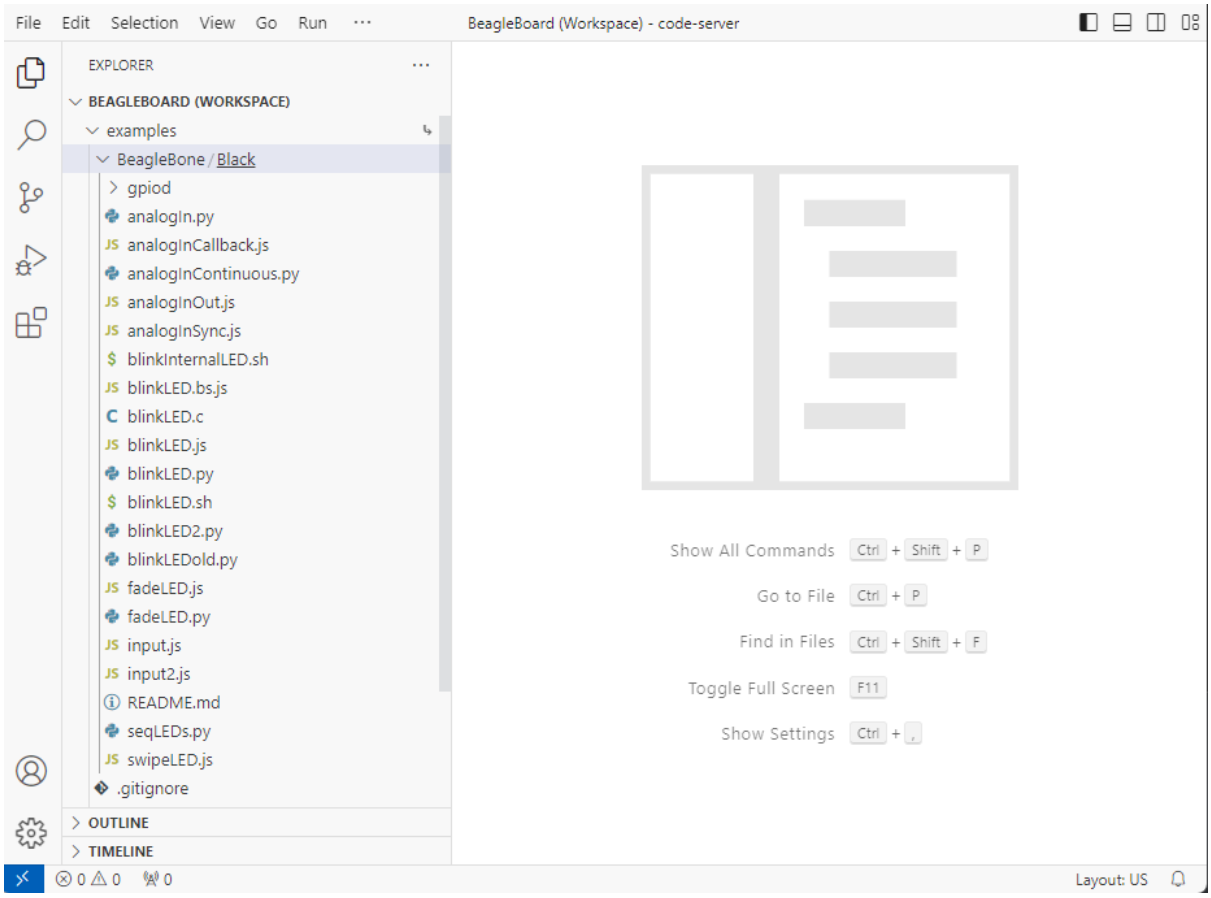


Fig. 15.139: Visual Studio Code IDE

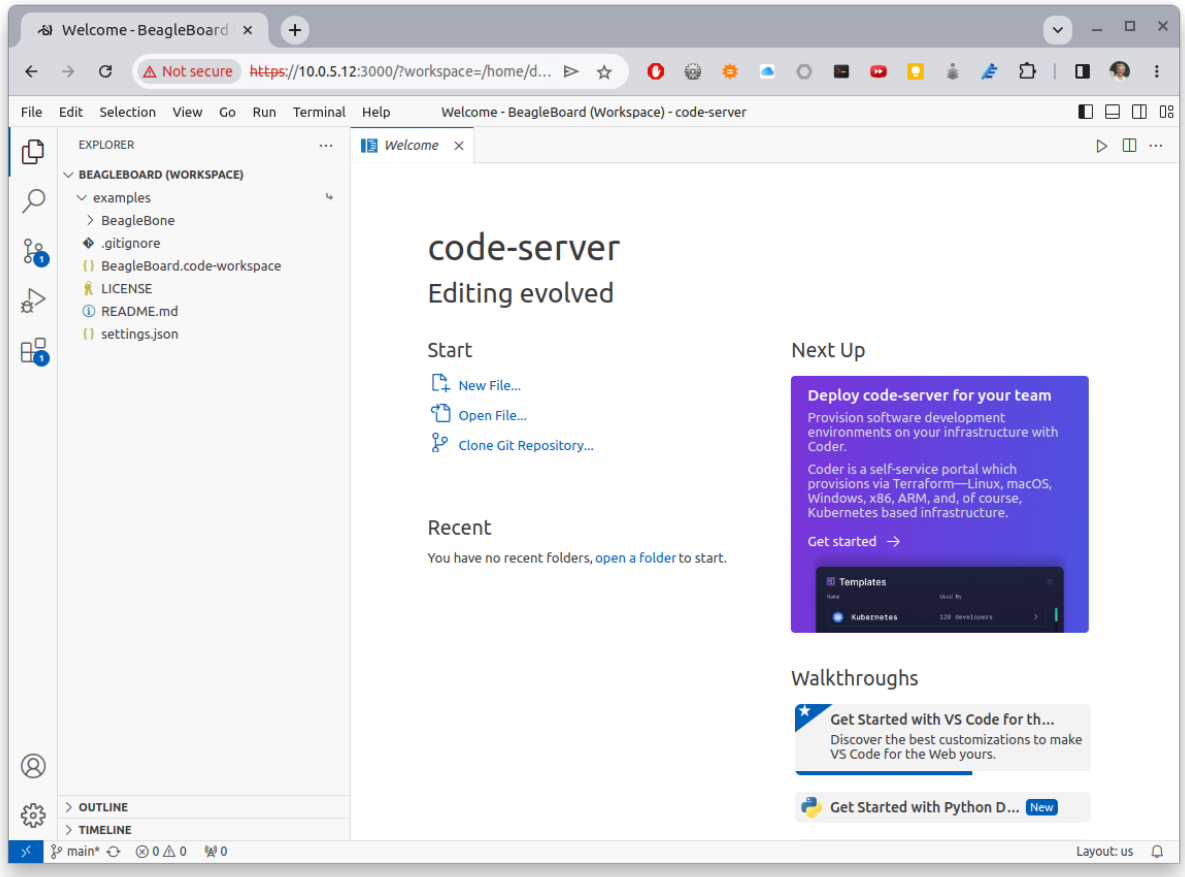


Fig. 15.140: Visual Studio Code Showing Home files

(continued from previous page)

```
bone:~$ ls
bbb-uEnv.txt  boot  etc   ID.txt  lost+found  mnt           opt   root  sbin␣
↪ sys  usr
bin           dev   home  lib     media       nfs-uEnv.txt  proc  run   srv ␣
↪ tmp  var
```

Now you can reach all the files from VS Code.

Getting Example Code

Problem You are ready to start playing with the examples and need to find the code.

Solution You can find the code on the PRU Cookbook Code project on git.beagleboard.org: https://git.beagleboard.org/beagleboard/pru-cookbook-code. Just clone it on your Beagle.

```
bone:~$ cd /opt/source
bone:~$ git clone https://git.beagleboard.org/beagleboard/pru-cookbook-code
bone:~$ cd pru-cookbook-code
bone:~$ sudo ./install.sh
bone:~$ ls -F
01case/   03details/  05blocks/  07more/  README.md
02start/  04details/  06io/      08ai/
```

Each chapter has its own directory that has all of the code.

```
bone:~$ cd 02start/
bone:~$ ls
hello.pru0.c  hello.pru1_1.c  Makefile  setup.sh
ai.notes          hello2.pru1_1.c  hello2.pru2_1.c  Makefile
hello2.pru0.c     hello2.pru1.c    hello.pru0.c     setup2.sh*
hello2.pru1_0.c   hello2.pru2_0.c  hello.pru1_1.c   setup.sh*
```

Go and explore.

Blinking an LED

Problem You want to make sure everything is set up by blinking an LED.

Solution The 'hello, world' of the embedded world is to flash an LED. *hello.pru0.c* is some code that blinks the USR3 LED ten times using the PRU.

Listing 15.70: hello.pru0.c

```
1  #include <stdint.h>
2  #include <pru_cfg.h>
3  #include "resource_table_empty.h"
4  #include "prugpio.h"
5
6  volatile register unsigned int __R30;
7  volatile register unsigned int __R31;
8
9  void main(void) {
10         int i;
11
12         uint32_t *gpio1 = (uint32_t *)GPIO1;
13
14         /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
```

(continues on next page)

(continued from previous page)

```
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        for(i=0; i<10; i++) {
                gpio1[GPIO_SETDATAOUT]   = USR3;        // The the USR3 LED
 ↪on

                __delay_cycles(500000000/5);            // Wait 1/2 second

                gpio1[GPIO_CLEARDATAOUT] = USR3;

                __delay_cycles(500000000/5);

        }
        __halt();
}

// Turns off triggers
#pragma DATA_SECTION(init_pins, ".init_pins")
#pragma RETAIN(init_pins)
const char init_pins[] =
        "/sys/class/leds/beaglebone:green:usr3/trigger\0none\0" \
        "\0\0";
```

hello.pru0.c

Later chapters will go into details of how this code works, but if you want to run it right now do the following.

```
bone:~$ cd /opt/source
bone:~$ git clone https://git.beagleboard.org/beagleboard/pru-cookbook-code
bone:~$ cd pru-cookbook-code/02start
bone:~$ sudo ../install.sh
```

Tip: If the following doesn't work see *Compiling with clpru and lnkpru* for installation instructions.

Running Code on the Black or Pocket

```
bone:~$ make TARGET=hello.pru0
/opt/source/pru-cookbook-code/common/Makefile:27: MODEL=TI_AM335x_BeagleBone_
 ↪Green_Wireless,TARGET=hello.pru0,COMMON=/opt/source/pru-cookbook-code/
 ↪common
-    Stopping PRU 0
CC  hello.pru0.c
"/opt/source/pru-cookbook-code/common/prugpio.h", line 53: warning #1181-D:
 ↪#warning directive: "Found else"
LD  /tmp/vsx-examples/hello.pru0.o
-   copying firmware file /tmp/vsx-examples/hello.pru0.out to /lib/firmware/
 ↪am335x-pru0-fw
-    Starting PRU 0
write_init_pins.sh
writing "none" to "/sys/class/leds/beaglebone:green:usr3/trigger"
MODEL   = TI_AM335x_BeagleBone_Green_Wireless
PROC    = pru
PRUN    = 0
PRU_DIR = /sys/class/remoteproc/remoteproc1
```

Tip: If you get the following error:

```
cp: cannot create regular file '/lib/firmware/am335x-pru0-fw': Permission
 ↪denied
```

Run the following command to set the permissions.

```
bone:~$ sudo chown debian:debian /lib/firmware/am335x-pru*
```

Running Code on the AI

```
bone$ make TARGET=hello.pru1_1
/var/lib/code-server/common/Makefile:28: MODEL=BeagleBoard.org_BeagleBone_AI,
 ↪TARGET=hello.pru1_1
-    Stopping PRU 1_1
CC  hello.pru1_1.c
"/var/lib/code-server/common/prugpio.h", line 4: warning #1181-D: #warning␣
 ↪directive: "Found AI"
LD  /tmp/code-server-examples/hello.pru1_1.o
-   copying firmware file /tmp/code-server-examples/hello.pru1_1.out to /lib/
 ↪firmware/am57xx-pru1_1-fw
write_init_pins.sh
writing "none" to "/sys/class/leds/beaglebone:green:usr3/trigger"
-    Starting PRU 1_1
MODEL   = BeagleBoard.org_BeagleBone_AI
PROC    = pru
PRUN    = 1_1
PRU_DIR = /dev/remoteproc/pruss1-core1
rm /tmp/code-server-examples/hello.pru1_1.o
```

Look quickly and you will see the USR3 LED blinking.

Later sections give more details on how all this works.

15.2.3 Running a Program; Configuring Pins

There are a lot of details in compiling and running PRU code. Fortunately those details are captured in a common *Makefile* that is used throughout this book. This chapter shows how to use the *Makefile* to compile code and also start and stop the PRUs.

Note: The following are resources used in this chapter:

- PRU Code Generation Tools - Compiler
- PRU Software Support Package
- PRU Optimizing C/C++ Compiler
- PRU Assembly Language Tools
- AM572x Technical Reference Manual (AI)
- AM335x Technical Reference Manual (All others)

Getting Example Code

Problem I want to get the files used in this book.

Solution It's all on a GitHub repository.

```
bone$ cd /opt/source
bone$ git clone https://git.beagleboard.org/beagleboard/pru-cookbook-code
bone$ cd pru-cookbook-code
bone$ sudo ./install.sh
```

Note: #TODO#: The version of code used needs to be noted in the documentation.

Note: #TODO#: Why is this documented in multiple places?

Compiling with clpru and lnkpru

Problem You need details on the c compiler, linker and other tools for the PRU.

Solution The PRU compiler and linker are already installed on many images. They are called `clpru` and `lnkpru`. Do the following to see if `clpru` is installed.

```
bone$ which clpru
/usr/bin/clpru
```

Tip: If `clpru` isn't installed, follow the instructions at https://elinux.org/Beagleboard:BeagleBoneBlack_Debian#TI_PRU_Code_Generation_Tools to install it.

```
bone$ sudo apt update
bone$ sudo apt install ti-pru-cgt-installer
```

Details on each can be found here:

- PRU Optimizing C/C++ Compiler
- PRU Assembly Language Tools

In fact there are PRU versions of many of the standard code generation tools.

code tools

```
bone$ ls /usr/bin/*pru
/usr/bin/abspru    /usr/bin/clistpru  /usr/bin/hexpru      /usr/bin/ofdpru
/usr/bin/acpiapru  /usr/bin/clpru     /usr/bin/ilkpru      /usr/bin/optpru
/usr/bin/arpru     /usr/bin/dempru    /usr/bin/libinfopru  /usr/bin/rc_test_
↪encoders_pru
/usr/bin/asmpru    /usr/bin/dispru    /usr/bin/lnkpru      /usr/bin/strippru
/usr/bin/cgpru     /usr/bin/embedpru  /usr/bin/nmpru       /usr/bin/xrefpru
```

See the PRU Assembly Language Tools for more details.

Making sure the PRUs are configured

Problem When running the Makefile for the PRU you get and error about `/dev/remoteproc` is missing.

Solution Edit `/boot/uEnv.txt` and enble pru_rproc by doing the following.

```
bone$ sudo vi /boot/uEnv.txt
```

Around line 40 you will see:

```
###pru_rproc (4.19.x-ti kernel)
uboot_overlay_pru=AM335X-PRU-RPROC-4-19-TI-00A0.dtbo
```

Uncomment the `uboot_overlay` line as shown and then reboot. `/dev/remoteproc` should now be there.

```
bone$ sudo reboot
bone$ ls -ls /dev/remoteproc/
total 0
0 lrwxrwxrwx 1 root root 33 Jul 29 16:12 pruss-core0 -> /sys/class/
↪remoteproc/remoteproc1
0 lrwxrwxrwx 1 root root 33 Jul 29 16:12 pruss-core1 -> /sys/class/
↪remoteproc/remoteproc2
```

Compiling and Running

Problem I want to compile and run an example.

Solution Change to the directory of the code you want to run.

```
bone$ cd pru-cookbook-code/06io
bone$ ls
gpio.pru0.c  Makefile  setup.sh
```

Source the setup file.

```
bone$ source setup.sh
TARGET=gpio.pru0
PocketBeagle Found
P2_05
Current mode for P2_05 is:     gpio
Current mode for P2_05 is:     gpio
```

Now you are ready to compile and run. This is automated for you in the Makefile

```
bone$ make
/opt/source/pru-cookbook-code/common/Makefile:27: MODEL=TI_AM335x_BeagleBone_
↪Green_Wireless,TARGET=gpio.pru0,COMMON=/opt/source/pru-cookbook-code/common
-    Stopping PRU 0
CC  gpio.pru0.c
"/opt/source/pru-cookbook-code/common/prugpio.h", line 53: warning #1181-D:
↪#warning directive: "Found else"
LD  /tmp/vsx-examples/gpio.pru0.o
-   copying firmware file /tmp/vsx-examples/gpio.pru0.out to /lib/firmware/
↪am335x-pru0-fw
-    Starting PRU 0
write_init_pins.sh
MODEL   = TI_AM335x_BeagleBone_Green_Wireless
PROC    = pru
PRUN    = 0
PRU_DIR = /sys/class/remoteproc/remoteproc1
rm /tmp/vsx-examples/gpio.pru0.o
```

Congratulations, your are now running a PRU. If you have an LED attached to `P9_11` on the Black, or `P2_05` on the Pocket, it should be blinking.

Discussion The `setup.sh` file sets the `TARGET` to the file you want to compile. Set it to the filename, without the `.c` extension (`gpio.pru0`). The file extension `.pru0` specifies the number of the PRU you are using (either `1_0`, `1_1`, `2_0`, `2_1` on the AI or `0` or `1` on the others)

You can override the `TARGET` on the command line.

```
bone$ cp gpio.pru0.c gpio.pru1.c
bone$ export TARGET=gpio.pru1
```

Notice the `TARGET` doesn't have the `.c` on the end.

You can also specify them when running `make`.

```
bone$ cp gpio.pru0.c gpio.pru1.c
bone$ make TARGET=gpio.pru1
```

The setup file also contains instructions to figure out which Beagle you are running and then configure the pins accordingly.

Listing 15.71: setup.sh

```
1  #!/bin/bash
2
3  export TARGET=gpio.pru0
4  echo TARGET=$TARGET
5
6  # Configure the PRU pins based on which Beagle is running
7  machine=$(awk '{print $NF}' /proc/device-tree/model)
8  echo -n $machine
9  if [ $machine = "Black" ]; then
10     echo " Found"
11     pins="P9_11"
12 elif [ $machine = "Blue" ]; then
13     echo " Found"
14     pins=""
15 elif [ $machine = "PocketBeagle" ]; then
16     echo " Found"
17     pins="P2_05"
18 else
19     echo " Not Found"
20     pins=""
21 fi
22
23 for pin in $pins
24 do
25     echo $pin
26     config-pin $pin gpio
27     config-pin -q $pin
28 done
```

`setup.sh`

Line	Explanation
2-5	Set which PRU to use and which file to compile.
7	Figure out which type of Beagle we have.
9-21	Based on the type, set the *pins*.
23-28	Configure (set the pin mux) for each of the pins.

Tip: The BeagleBone AI has it's pins preconfigured at boot time, so there's no need to use `config-pin`.

The `Makefile` stops the PRU, compiles the file and moves it where it will be loaded, and then restarts the PRU.

Stopping and Starting the PRU

Problem I want to stop and start the PRU.

Solution It's easy, if you already have `TARGET` set up:

```
bone$ make stop
-    Stopping PRU 0
stop
bone$ make start
-    Starting PRU 0
start
```

See *dmesg Hw* to see how to tell if the PRU is stopped.

This assumes `TARGET` is set to the PRU you are using. If you want to control the other PRU use:

```
bone$ cp gpio.pru0.c gpio.pru1.c
bone$ make TARGET=gpio.pru1
bone$ make TARGET=gpio.pru1 stop
bone$ make TARGET=gpio.pru1 start
```

The Standard Makefile

Problem There are all sorts of options that need to be set when compiling a program. How can I be sure to get them all right?

Solution The surest way to make sure everything is right is to use our standard `Makefile`.

Discussion It's assumed you already know how Makefiles work. If not, there are many resources online that can bring you up to speed. Here is the local `Makefile` used throughout this book.

Listing 15.72: Local Makefile

```
1 include /opt/source/pru-cookbook-code/common/Makefile
```

`Makefile`

Each of the local Makefiles refer to the same standard Makefile. The details of how the Makefile works is beyond the scope of this cookbook.

Fortunately you shouldn't have to modify the *Makefile*.

The Linker Command File - am335x_pru.cmd

Problem The linker needs to be told where in memory to place the code and variables.

Solution `am335x_pru.cmd` is the standard linker command file that tells the linker where to put what for the BeagleBone Black and Blue, and the Pocket. The `am57xx_pru.cmd` does the same for the AI. «««< HEAD Both files can be found in `/var/lib/code-server/common`. ======= Both files can be found in `/opt/source/pru-cookbook-code/common`. »»»> bf423e10a7d607eb485449d3f53e7823264dfebb

Listing 15.73: am335x_pru.cmd

```
/*****************************************************************************/
/*  AM335x_PRU.cmd                                                           */
/*  Copyright (c) 2015  Texas Instruments Incorporated                       */
/*                                                                           */
/*    Description: This file is a linker command file that can be used for   */
/*                 linking PRU programs built with the C compiler and        */
/*                 the resulting .out file on an AM335x device.              */
/*****************************************************************************/

-cr                                                         /* Link using C conventions */

/* Specify the System Memory Map */
MEMORY
{
      PAGE 0:
        PRU_IMEM                : org = 0x00000000 len = 0x00002000  /* 8kB PRU0 Instruction RAM */

      PAGE 1:

        /* RAM */

        PRU_DMEM_0_1        : org = 0x00000000 len = 0x00002000 CREGISTER=24 /* 8kB PRU Data RAM 0_1 */
        PRU_DMEM_1_0        : org = 0x00002000 len = 0x00002000        CREGISTER=25 /* 8kB PRU Data RAM 1_0 */

          PAGE 2:
        PRU_SHAREDMEM        : org = 0x00010000 len = 0x00003000 CREGISTER=28 /* 12kB Shared RAM */

        DDR                             : org = 0x80000000 len = 0x00000100        CREGISTER=31
        L3OCMC                          : org = 0x40000000 len = 0x00010000        CREGISTER=30


        /* Peripherals */

        PRU_CFG                         : org = 0x00026000 len = 0x00000044        CREGISTER=4
        PRU_ECAP                : org = 0x00030000 len = 0x00000060        CREGISTER=3
        PRU_IEP                         : org = 0x0002E000 len = 0x0000031C        CREGISTER=26
        PRU_INTC                : org = 0x00020000 len = 0x00001504        CREGISTER=0
        PRU_UART                : org = 0x00028000 len = 0x00000038        CREGISTER=7

```

(continues on next page)

(continued from previous page)

```
        DCAN0                           : org = 0x481CC000 len =␣
↪0x000001E8         CREGISTER=14
        DCAN1                           : org = 0x481D0000 len =␣
↪0x000001E8         CREGISTER=15
        DMTIMER2                 : org = 0x48040000 len =␣
↪0x0000005C         CREGISTER=1
        PWMSS0                           : org = 0x48300000 len =␣
↪0x000002C4         CREGISTER=18
        PWMSS1                           : org = 0x48302000 len =␣
↪0x000002C4         CREGISTER=19
        PWMSS2                           : org = 0x48304000 len =␣
↪0x000002C4         CREGISTER=20
        GEMAC                           : org = 0x4A100000 len =␣
↪0x0000128C         CREGISTER=9
        I2C1                           : org = 0x4802A000 len =␣
↪0x000000D8         CREGISTER=2
        I2C2                           : org = 0x4819C000 len =␣
↪0x000000D8         CREGISTER=17
        MBX0                           : org = 0x480C8000 len =␣
↪0x00000140         CREGISTER=22
        MCASP0_DMA                : org = 0x46000000 len =␣
↪0x00000100         CREGISTER=8
        MCSPI0                           : org = 0x48030000 len =␣
↪0x000001A4         CREGISTER=6
        MCSPI1                           : org = 0x481A0000 len =␣
↪0x000001A4         CREGISTER=16
        MMCHS0                           : org = 0x48060000 len =␣
↪0x00000300         CREGISTER=5
        SPINLOCK                 : org = 0x480CA000 len =␣
↪0x00000880         CREGISTER=23
        TPCC                           : org = 0x49000000 len =␣
↪0x00001098         CREGISTER=29
        UART1                           : org = 0x48022000 len =␣
↪0x00000088         CREGISTER=11
        UART2                           : org = 0x48024000 len =␣
↪0x00000088         CREGISTER=12

        RSVD10                           : org = 0x48318000 len =␣
↪0x00000100         CREGISTER=10
        RSVD13                           : org = 0x48310000 len =␣
↪0x00000100         CREGISTER=13
        RSVD21                           : org = 0x00032400 len =␣
↪0x00000100         CREGISTER=21
        RSVD27                           : org = 0x00032000 len =␣
↪0x00000100         CREGISTER=27

}

/* Specify the sections allocation into memory */
SECTIONS {
        /* Forces _c_int00 to the start of PRU IRAM. Not necessary when␣
↪loading
           an ELF file, but useful when loading a binary */
        .text:_c_int00*        >  0x0, PAGE 0

        .text                >  PRU_IMEM, PAGE 0
        .stack                 >  PRU_DMEM_0_1, PAGE 1
        .bss                 >  PRU_DMEM_0_1, PAGE 1
        .cio                 >  PRU_DMEM_0_1, PAGE 1
        .data                 >  PRU_DMEM_0_1, PAGE 1
        .switch                >  PRU_DMEM_0_1, PAGE 1
```

(continues on next page)

(continued from previous page)

```
78          .sysmem                 >  PRU_DMEM_0_1, PAGE 1
79          .cinit                  >  PRU_DMEM_0_1, PAGE 1
80          .rodata                 >  PRU_DMEM_0_1, PAGE 1
81          .rofardata          >  PRU_DMEM_0_1, PAGE 1
82          .farbss                 >  PRU_DMEM_0_1, PAGE 1
83          .fardata          >  PRU_DMEM_0_1, PAGE 1
84
85          .resource_table > PRU_DMEM_0_1, PAGE 1
86          .init_pins > PRU_DMEM_0_1, PAGE 1
87  }
```

`am335x_pru.cmd`

The cmd file for the AI is about the same, with appropriate addresses for the AI.

Discussion The important things to notice in the file are given in the following table.

AM335x_PRU.cmd important things

Line	Explanation
16	This is where the instructions are stored. See page 206 of the AM335x Technical Reference Manual rev. P Or see page 417 of AM572x Technical Reference Manual for the AI.
22	This is where PRU 0's DMEM 0 is mapped. It's also where PRU 1's DMEM 1 is mapped.
23	The reverse to above. PRU 0's DMEM 1 appears here and PRU 1's DMEM 0 is here.
26	The shared memory for both PRU's appears here.
72	The *.text* section is where the code goes. It's mapped to *IMEM*
73	The ((stack)) is then mapped to DMEM 0. Notice that DMEM 0 is one bank
	of memory for PRU 0 and another for PRU1, so they both get their own stacks.
74	The *.bss* section is where the **heap** goes.

Why is it important to understand this file? If you are going to store things in DMEM, you need to be sure to start at address 0x0200 since the **stack** and the **heap** are in the locations below 0x0200.

Loading Firmware

Problem I have my PRU code all compiled and need to load it on the PRU.

Solution It's a simple three step process.

- Stop the PRU
- Write the `.out` file to the right place in `/lib/firmware`
- Start the PRU.

This is all handled in the *The Standard Makefile*.

Discussion The PRUs appear in the Linux file space at `/dev/remoteproc/`.

Finding the PRUs

```
bone$ cd /dev/remoteproc/
bone$ ls
pruss-core0  pruss-core1
```

Or if you are on the AI:

```
bone$ cd /dev/remoteproc/
bone$ ls
dsp1  dsp2  ipu1  ipu2  pruss1-core0  pruss1-core1  pruss2-core0  pruss2-
↪core1
```

You see there that the AI has two pairs of PRUs, plus a couple of DSPs and other goodies.

Here we see PRU 0 and PRU 1 in the path. Let's follow PRU 0.

```
bone$ cd pruss-core0
bone$ ls
device  firmware  name  power  state  subsystem  uevent
```

Here we see the files that control PRU 0. `firmware` tells where in `/lib/firmware` to look for the code to run on the PRU.

```
bone$ cat firmware
am335x-pru0-fw
```

Therefore you copy your `.out` file to `/lib/firmware/am335x-pru0-fw`.

Configuring Pins for Controlling Servos

Problem You want to **configure** the pins so the PRU outputs are accessible.

Solution It depends on which Beagle you are running on. If you are on the AI or Blue, everything is already configured for you. If you are on the Black or Pocket you'll need to run the following script.

Listing 15.74: servos_setup.sh

```
#!/bin/bash
# Configure the PRU pins based on which Beagle is running
machine=$(awk '{print $NF}' /proc/device-tree/model)
echo -n $machine
if [ $machine = "Black" ]; then
    echo " Found"
    pins="P8_27 P8_28 P8_29 P8_30 P8_39 P8_40 P8_41 P8_42"
elif [ $machine = "Blue" ]; then
    echo " Found"
    pins=""
elif [ $machine = "PocketBeagle" ]; then
    echo " Found"
    pins="P2_35 P1_35 P1_02 P1_04"
else
    echo " Not Found"
    pins=""
fi

for pin in $pins
do
    echo $pin
    config-pin $pin pruout
    config-pin -q $pin
done
```

`servos_setup.sh`

Discussion The first part of the code looks in `/proc/device-tree/model` to see which Beagle is running. Based on that it assigns `pins` a list of pins to configure. Then the last part of the script loops through each of the pins and configures it.

Configuring Pins for Controlling Encoders

Problem You want to **configure** the pins so the PRU inputs are accessible.

Solution It depends on which Beagle you are running on. If you are on the AI or Blue, everything is already configured for you. If you are on the Black or Pocket you'll need to run the following script.

Listing 15.75: encoder_setup.sh

```
#!/bin/bash
# Configure the pins based on which Beagle is running
machine=$(awk '{print $NF}' /proc/device-tree/model)
echo -n $machine

# Configure eQEP pins
if [ $machine = "Black" ]; then
    echo " Found"
    pins="P9_92 P9_27 P8_35 P8_33 P8_12 P8_11 P8_41 P8_42"
elif [ $machine = "Blue" ]; then
    echo " Found"
    pins=""
elif [ $machine = "PocketBeagle" ]; then
    echo " Found"
    pins="P1_31 P2_34 P2_10 P2_24 P2_33"
else
    echo " Not Found"
    pins=""
fi

for pin in $pins
do
    echo $pin
    config-pin $pin qep
    config-pin -q $pin
done

########################################
# Configure PRU pins
if [ $machine = "Black" ]; then
    echo " Found"
    pins="P8_16 P8_15"
elif [ $machine = "Blue" ]; then
    echo " Found"
    pins=""
elif [ $machine = "PocketBeagle" ]; then
    echo " Found"
    pins="P2_09 P2_18"
else
    echo " Not Found"
    pins=""
fi

for pin in $pins
do
    echo $pin
    config-pin $pin pruin
    config-pin -q $pin
done
```

encoder_setup.sh

Discussion This works like the servo setup except some of the pins are configured as to the hardware eQEPs and other to the PRU inputs.

15.2.4 Debugging and Benchmarking

One of the challenges is getting debug information out of the PRUs since they don't have a traditional `printf()`. In this chapter four different methods are presented that I've found useful in debugging. The first is simply attaching an LED. The second is using `dmesg` to watch the kernel messages. `prudebug`, a simple debugger that allows you to inspect registers and memory of the PRUs, is then presented. Finally, using one of the UARTS to send debugging information out a serial port is shown.

Debugging via an LED

Problem I need a simple way to see if my program is running without slowing the real-time execution.

Solution One of the simplest ways to do this is to attach an LED to the output pin and watch it flash. *LED used for debugging P9_29* shows an LED attached to pin P9_29 of the BeagleBone Black.

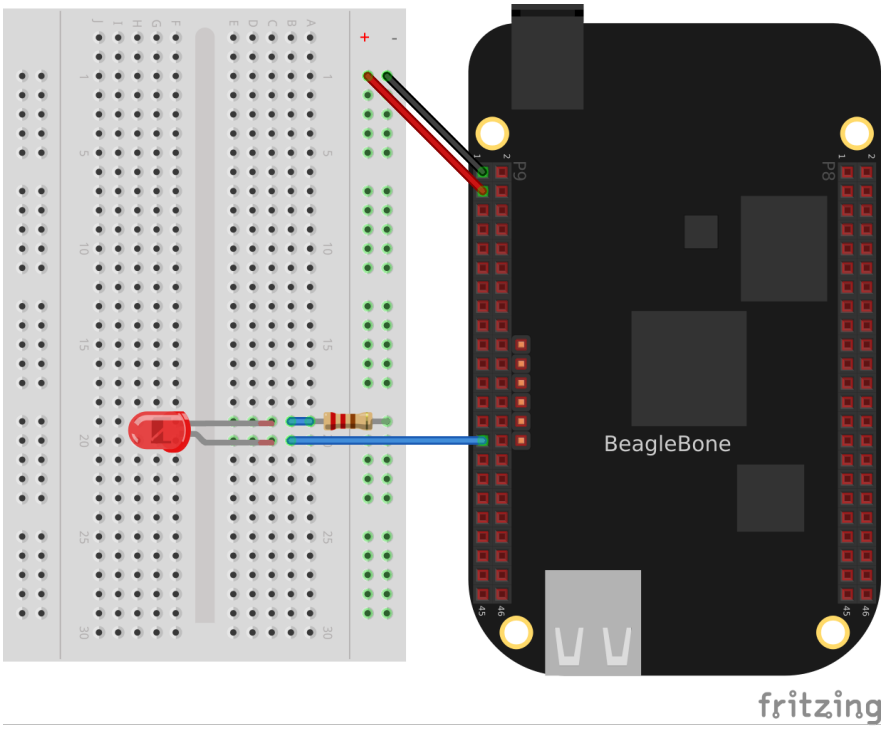


Fig. 15.141: LED used for debugging P9_29

Make sure you have the LED in the correct way, or it won't work.

Discussion If your output is changing more than a few times a second, the LED will be blinking too fast and you'll need an oscilloscope or a logic analyzer to see what's happening.

Another useful tool that let's you see the contents of the registers and RAM is discussed in *prudebug - A Simple Debugger for the PRU*.

dmesg Hw

Problem I'm getting an error message (`/sys/devices/platform/ocp/4a326000.pruss-soc-bus/4a300000.pruss/4a334000.pru0/remoteproc/remoteproc1/state: Invalid argument`) when I load my code, but don't know what's causing it.

Solution The command `dmesg` outputs useful information when dealing with the kernel. Simply running `dmesg -Hw` can tell you a lot. The `-H` flag puts the dates in the human readable form, the `-w` tells it to wait for more information. Often I'll have a window open running `dmesg -Hw`.

Here's what `dmesg` said for the example above.

dmesg -Hw

```
[  +0.000018] remoteproc remoteproc1: header-less resource table
[  +0.011879] remoteproc remoteproc1: Failed to find resource table
[  +0.008770] remoteproc remoteproc1: Boot failed: -22
```

It quickly told me I needed to add the line `#include "resource_table_empty.h"` to my code.

prudebug - A Simple Debugger for the PRU

Problem You need to examine registers and memory on the PRUs.

Solution `prudebug` is a simple debugger for the PRUs that lets you start and stop the PRUs and examine the registers and memory. It can be found on GitHub https://github.com/RRvW/prudebug-rl. I have a version I updated to use byte addressing rather than word addressing. This makes it easier to work with the assembler output. You can find it in my GitHub BeagleBoard repo https://github.com/MarkAYoder/BeagleBoard-exercises/tree/master/pru/prudebug.

Just download the files and type `make`.

Discussion Once `prudebug` is installed is rather easy to use.

Note: `prudebug` has now been ported to the AI.

```
bone$ *sudo prudebug*
PRU Debugger v0.25
(C) Copyright 2011, 2013 by Arctica Technologies.  All rights reserved.
Written by Steven Anderson

Using /dev/mem device.
Processor type                AM335x
PRUSS memory address  0x4a300000
PRUSS memory length   0x00080000

        offsets below are in 32-bit byte addresses (not ARM byte addresses)
        PRU            Instruction    Data         Ctrl
        0              0x00034000     0x00000000   0x00022000
        1              0x00038000     0x00002000   0x00024000
```

You get help by entering `help`. You cal also enter `hb` to get a brief help.

```
PRU0> *hb*
Command help

    BR [breakpoint_number [address]] - View or set an instruction breakpoint
    D memory_location_ba [length] - Raw dump of PRU data memory (32-bit byte␣
↪offset from beginning of full PRU memory block - all PRUs)
    DD memory_location_ba [length] - Dump data memory (32-bit byte offset␣
↪from beginning of PRU data memory)
    DI memory_location_ba [length] - Dump instruction memory (32-bit byte␣
↪offset from beginning of PRU instruction memory)
    DIS memory_location_ba [length] - Disassemble instruction memory (32-bit␣
↪byte offset from beginning of PRU instruction memory)
    G - Start processor execution of instructions (at current IP)
    GSS - Start processor execution using automatic single stepping - this␣
↪allows running a program with breakpoints
    HALT - Halt the processor
    L memory_location_iwa file_name - Load program file into instruction␣
↪memory
    PRU pru_number - Set the active PRU where pru_number ranges from 0 to 1
    Q - Quit the debugger and return to shell prompt.
    R - Display the current PRU registers.
    RESET - Reset the current PRU
    SS - Single step the current instruction.
    WA [watch_num [address [value]]] - Clear or set a watch point
    WR memory_location_ba value1 [value2 [value3 ...]] - Write a 32-bit␣
↪value to a raw (offset from beginning of full PRU memory block)
    WRD memory_location_ba value1 [value2 [value3 ...]] - Write a 32-bit␣
↪value to PRU data memory for current PRU
    WRI memory_location_ba value1 [value2 [value3 ...]] - Write a 32-bit␣
↪value to PRU instruction memory for current PRU
```

Initially you are talking to PRU 0. You can enter `pru 1` to talk to PRU 1. The commands I find most useful are, `r`, to see the registers.

```
PRU0> *r*
Register info for PRU0
    Control register: 0x00008003
      Reset PC:0x0000  RUNNING, FREE_RUN, COUNTER_DISABLED, NOT_SLEEPING,␣
↪PROC_ENABLED

    Program counter: 0x0030
      Current instruction: ADD R0.b0, R0.b0, R0.b0

    Rxx registers not available since PRU is RUNNING.
```

Notice the PRU has to be stopped to see the register contents.

```
PRU0> *h*
PRU0 Halted.
PRU0> *r*
Register info for PRU0
    Control register: 0x00000001
      Reset PC:0x0000  STOPPED, FREE_RUN, COUNTER_DISABLED, NOT_SLEEPING,␣
↪PROC_DISABLED

    Program counter: 0x0028
      Current instruction: LBBO R15, R15, 4, 4

    R00: 0x00000000    R08: 0x00000000    R16: 0x00000001    R24: 0x00000002
    R01: 0x00000000    R09: 0xaf40dcf2    R17: 0x00000000    R25: 0x00000003
    R02: 0x000000dc    R10: 0xd8255b1b    R18: 0x00000003    R26: 0x00000003
    R03: 0x000f0000    R11: 0xc50cbefd    R19: 0x00000100    R27: 0x00000002
```

(continues on next page)

(continued from previous page)

```
    R04: 0x00000000    R12: 0xb037c0d7    R20: 0x00000100    R28: 0x8ca9d976
    R05: 0x00000009    R13: 0xf48bbe23    R21: 0x441fb678    R29: 0x00000002
    R06: 0x00000000    R14: 0x00000134    R22: 0xc8cc0752    R30: 0x00000000
    R07: 0x00000009    R15: 0x00000200    R23: 0xe346fee9    R31: 0x00000000
```

You can resume using `g` which starts right where you left off, or use `reset` to restart back at the beginning.

The `dd` command dumps the memory. Keep in mind the following.

Table 15.12: Important memory locations

Address	Contents
0x00000	Start of the **stack** for PRU 0. The file **AM335x_PRU.cmd** specifies where the stack is.
0x00100	Start of the **heap** for PRU 0.
0x00200	Start of DRAM that your programs can use. The **Makefile** specifies
	the size of the **stack** and the **heap**.
0x10000	Start of the memory shared between the PRUs.

Using `dd` with no address prints the next section of memory.

```
PRU0> *dd*
dd
Absolute addr = 0x0000, offset = 0x0000, Len = 16
[0x0000] 0x00000000 0x00000000 0x00000000 0x00000000
[0x0010] 0x00000000 0x00000000 0x00000000 0x00000000
[0x0020] 0x00000000 0x00000000 0x00000000 0x00000000
[0x0030] 0x00000000 0x00000000 0x00000000 0x00000000
```

The stack grows from higher memory to lower memory, so you often won't see much around address `0x0000`.

```
PRU0> *dd 0x100*
dd 0x100
Absolute addr = 0x0100, offset = 0x0000, Len = 16
[0x0100] 0x00000001 0x00000002 0x00000003 0x00000004
[0x0110] 0x00000004 0x00000003 0x00000002 0x00000001
[0x0120] 0x00000001 0x00000000 0x00000000 0x00000000
[0x0130] 0x00000000 0x00000200 0x862e5c18 0xfeb21aca
```

Here we see some values on the heap.

```
PRU0> *dd 0x200*
dd 0x200
Absolute addr = 0x0200, offset = 0x0000, Len = 16
[0x0200] 0x00000001 0x00000004 0x00000002 0x00000003
[0x0210] 0x00000003 0x00000011 0x00000004 0x00000010
[0x0220] 0x0a4fe833 0xb222ebda 0xe5575236 0xc50cbefd
[0x0230] 0xb037c0d7 0xf48bbe23 0x88c460f0 0x011550d4
```

Data written explicitly to `0x0200` of the DRAM.

```
PRU0> *dd 0x10000*
dd 0x10000
Absolute addr = 0x10000, offset = 0x0000, Len = 16
[0x10000] 0x8ca9d976 0xebcb119e 0x3aebce31 0x68c44d8b
[0x10010] 0xc370ba7e 0x2fea993b 0x15c67fa5 0xfbf68557
[0x10020] 0x5ad81b4f 0x4a55071a 0x48576eb7 0x1004786b
[0x10030] 0x2265ebc6 0xa27b32a0 0x340d34dc 0xbfa02d4b
```

Here's the shared memory.

You can also use `prudebug` to set breakpoints and single step, but I haven't used that feature much.

Memory Allocation gives examples of how you can control where your variables are stored in memory.

UART

Problem I'd like to use something like `printf()` to debug my code.

Solution One simple, yet effective approach to 'printing' from the PRU is an idea taken from the Adruino playbook; use the UART (serial port) to output debug information. The PRU has it's own UART that can send characters to a serial port.

You'll need a 3.3V FTDI cable to go between your Beagle and the USB port on your host computer as shown in *FTDI cable*.[1] you can get such a cable from places such as Sparkfun or Adafruit.



Fig. 15.142: FTDI cable

Discussion The Beagle side of the FTDI cable has a small triangle on it as shown in *FTDI connector* which marks the ground pin, pin 1.

The *Wring for FTDI cable to Beagle* table shows which pins connect where and *FTDI to BB Black* is a wiring diagram for the BeagleBone Black.

Table 15.13: Wring for FTDI cable to Beagle

FTDI pin	Color	Black pin	AI 1 pin	AI 2 pin	Pocket	Function
0	black	P9_1	P8_1	P8_1	P1_16	ground
4	orange	P9_24	P8_43	P8_33a	P1_12	rx
5	yellow	P9_26	P8_44	P8_31a	P1_06	tx

[1] FTDI images are from the BeagleBone Cookbook

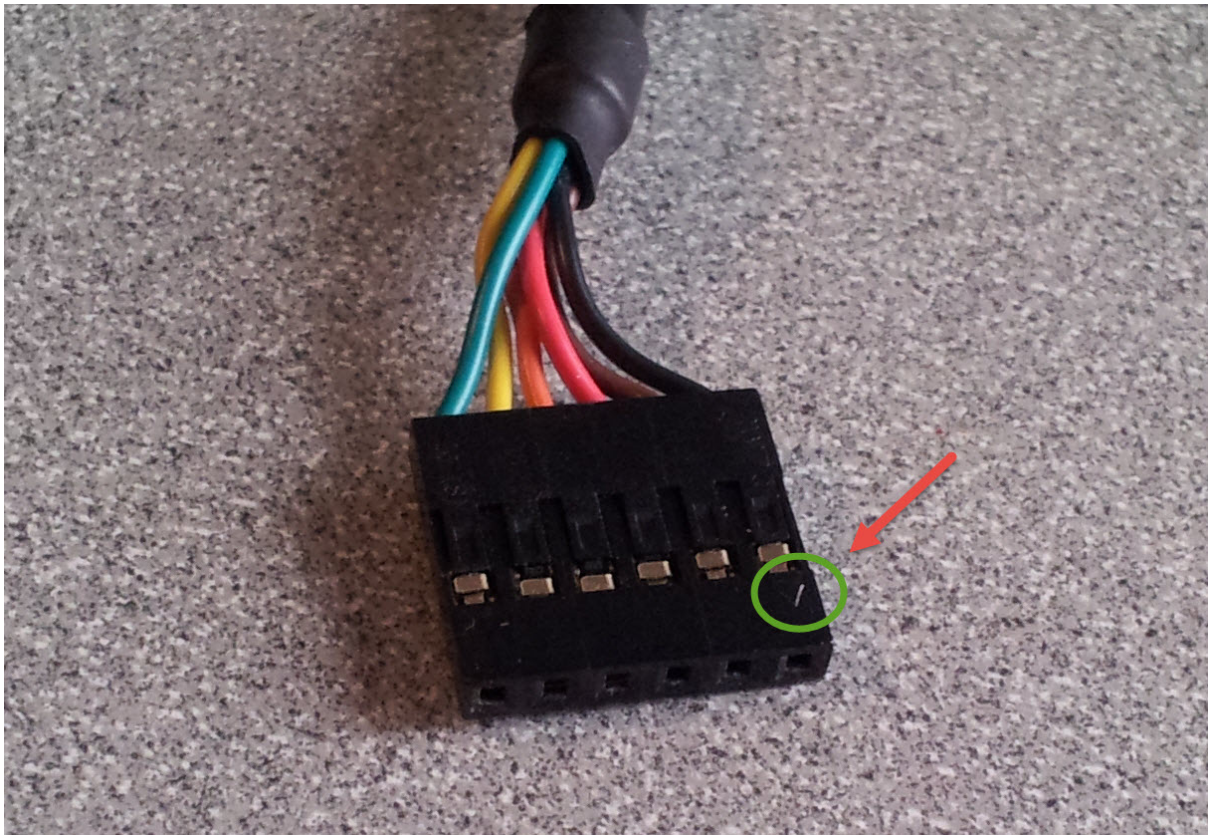


Fig. 15.143: FTDI connector

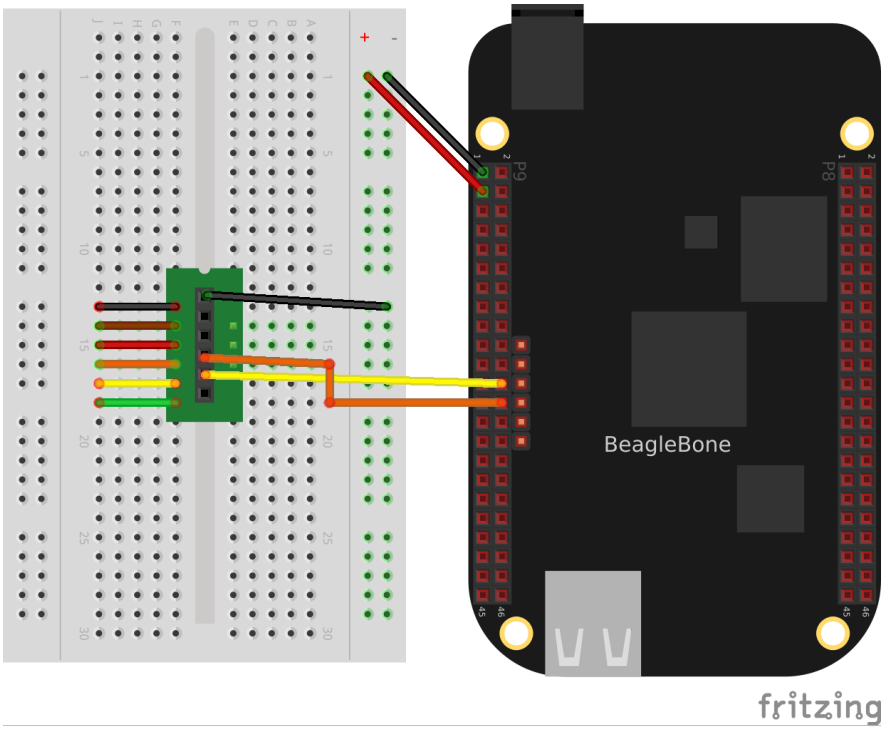


Fig. 15.144: FTDI to BB Black

Details Two examples of using the UART are presented here. The first (*uart1.pru1_0.c*) sends a character out the serial port then waits for a character to come in. Once the new character arrives another character is output.

The second example (*uart2.pru1_0.c*) prints out a string and then waits for characters to arrive. Once an ENTER appears the string is sent back.

Tip: On the Black, either PRU0 and PRU1 can run this code. Both have access to the same UART.

You need to set the pin muxes.

config-pin

```
# Configure tx Black
bone$ *config-pin P9_24 pru_uart*
# Configure rx Black
bone$ *config-pin P9_26 pru_uart*

# Configure tx Pocket
bone$ *config-pin P1_06 pru_uart*
# Configure rx Pocket
bone$ *config-pin P1_12 pru_uart*
```

Note: See *Configuring pins on the AI via device trees* for configuring pins on the AI. Make sure your *rx* pins are configured as input pins in the device tree.

For example

```
DRA7XX_CORE_IOPAD(0x3610, *PIN_INPUT* | MUX_MODE10) // C6: P8.33a:
```

Listing 15.76: uart1.pru1_0.c

```
// From: http://git.ti.com/pru-software-support-package/pru-software-support-package/trees/master/examples/am335x/PRU_Hardware_UART
// This example was converted to the am5729 by changing the names in pru_uart.h
// for the am335x to the more descriptive names for the am5729.
// For example  DLL convertes to DIVISOR_REGISTER_LSB_
#include <stdint.h>
#include <pru_uart.h>
#include "resource_table_empty.h"

/* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily)
 * only going to send 8 at a time */
#define FIFO_SIZE        16
#define MAX_CHARS        8

void main(void)
{
        uint8_t tx;
        uint8_t rx;
        uint8_t cnt;

        /*  hostBuffer points to the string to be printed */
        char* hostBuffer;

        /*** INITIALIZATION ***/

        /* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x oversample
```

(continues on next page)

(continued from previous page)

```
26          * 192MHz / 104 / 16 = ~115200 */
27         CT_UART.DIVISOR_REGISTER_LSB_ = 104;
28         CT_UART.DIVISOR_REGISTER_MSB_ = 0;
29         CT_UART.MODE_DEFINITION_REGISTER = 0x0;
30
31         /* Enable Interrupts in UART module. This allows the main thread to␣
   ↪poll for
32          * Receive Data Available and Transmit Holding Register Empty */
33         CT_UART.INTERRUPT_ENABLE_REGISTER = 0x7;
34
35         /* If FIFOs are to be used, select desired trigger level and enable
36          * FIFOs by writing to FCR. FIFOEN bit in FCR must be set first␣
   ↪before
37          * other bits are configured */
38         /* Enable FIFOs for now at 1-byte, and flush them */
39         CT_UART.INTERRUPT_IDENTIFICATION_REGISTER_FIFO_CONTROL_REGISTER =␣
   ↪(0x8) | (0x4) | (0x2) | (0x1);
40         //CT_UART.FCR = (0x80) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO␣
   ↪trigger
41
42         /* Choose desired protocol settings by writing to LCR */
43         /* 8-bit word, 1 stop bit, no parity, no break control and no␣
   ↪divisor latch */
44         CT_UART.LINE_CONTROL_REGISTER = 3;
45
46         /* Enable loopback for test */
47         CT_UART.MODEM_CONTROL_REGISTER = 0x00;
48
49         /* Choose desired response to emulation suspend events by configuring
50          * FREE bit and enable UART by setting UTRST and URRST in PWREMU_
   ↪MGMT */
51         /* Allow UART to run free, enable UART TX/RX */
52         CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER = 0x6001;
53
54         /*** END INITIALIZATION ***/
55
56         /* Priming the 'hostbuffer' with a message */
57         hostBuffer = "Hello!  This is a long string\r\n";
58
59         /*** SEND SOME DATA ***/
60
61         /* Let's send/receive some dummy data */
62         while(1) {
63                 cnt = 0;
64                 while(1) {
65                         /* Load character, ensure it is not string␣
   ↪termination */
66                         if ((tx = hostBuffer[cnt]) == '\0')
67                                 break;
68                         cnt++;
69                         CT_UART.RBR_THR_REGISTERS = tx;
70
71                         /* Because we are doing loopback, wait until LSR.DR␣
   ↪== 1
72                          * indicating there is data in the RX FIFO */
73                         while ((CT_UART.LINE_STATUS_REGISTER & 0x1) == 0x0);
74
75                         /* Read the value from RBR */
76                         rx = CT_UART.RBR_THR_REGISTERS;
77
78                         /* Wait for TX FIFO to be empty */
```

(continues on next page)

(continued from previous page)

```
                        while (!((CT_UART.INTERRUPT_IDENTIFICATION_REGISTER_
↪FIFO_CONTROL_REGISTER & 0x2) == 0x2));
                }
        }

        /*** DONE SENDING DATA ***/

        /* Disable UART before halting */
        CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER = 0x0;

        /* Halt PRU core */
        __halt();
}
```

uart1.pru1_0.c

Set the following variables so `make` will know what to compile.

Listing 15.77: make

```
bone$ *make TARGET=uart1.pru0*
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=uart1.pru0
-    Stopping PRU 0
-     copying firmware file /tmp/vsx-examples/uart1.pru0.out to /lib/
↪firmware/am335x-pru0-fw
write_init_pins.sh
-    Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR = /dev/remoteproc/pruss-core0
```

Now `make` will compile, load PRU0 and start it. In a terminal window on your host computer run

```
host$ *screen /dev/ttyUSB0 115200*
```

It will initially display the first charters (`H`) and then as you enter characters on the keyboard, the rest of the message will appear.

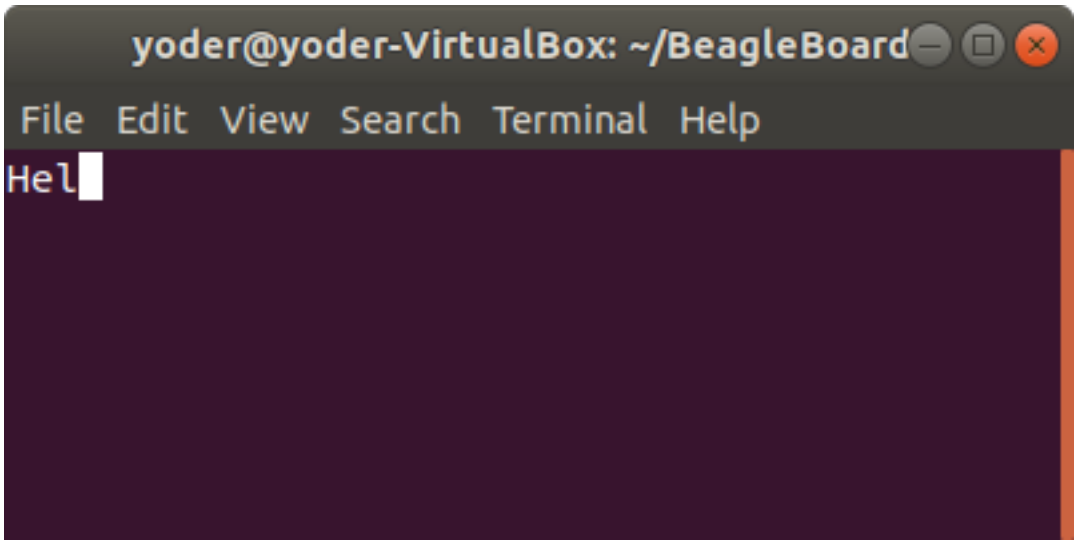


Fig. 15.145: uart1.pru0.c output

Here's the code (`uart1.pru1_0.c`) that does it.

Listing 15.78: uart1.pru1_0.c

```
// From: http://git.ti.com/pru-software-support-package/pru-software-support-package/trees/master/examples/am335x/PRU_Hardware_UART
// This example was converted to the am5729 by changing the names in pru_uart.h
// for the am335x to the more descriptive names for the am5729.
// For example  DLL convertes to DIVISOR_REGISTER_LSB_
#include <stdint.h>
#include <pru_uart.h>
#include "resource_table_empty.h"

/* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily)
 * only going to send 8 at a time */
#define FIFO_SIZE        16
#define MAX_CHARS        8

void main(void)
{
        uint8_t tx;
        uint8_t rx;
        uint8_t cnt;

        /*  hostBuffer points to the string to be printed */
        char* hostBuffer;

        /*** INITIALIZATION ***/

        /* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x oversample
         * 192MHz / 104 / 16 = ~115200 */
        CT_UART.DIVISOR_REGISTER_LSB_ = 104;
        CT_UART.DIVISOR_REGISTER_MSB_ = 0;
        CT_UART.MODE_DEFINITION_REGISTER = 0x0;

        /* Enable Interrupts in UART module. This allows the main thread to poll for
         * Receive Data Available and Transmit Holding Register Empty */
        CT_UART.INTERRUPT_ENABLE_REGISTER = 0x7;

        /* If FIFOs are to be used, select desired trigger level and enable
         * FIFOs by writing to FCR. FIFOEN bit in FCR must be set first before
         * other bits are configured */
        /* Enable FIFOs for now at 1-byte, and flush them */
        CT_UART.INTERRUPT_IDENTIFICATION_REGISTER_FIFO_CONTROL_REGISTER = (0x8) | (0x4) | (0x2) | (0x1);
        //CT_UART.FCR = (0x80) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO trigger

        /* Choose desired protocol settings by writing to LCR */
        /* 8-bit word, 1 stop bit, no parity, no break control and no divisor latch */
        CT_UART.LINE_CONTROL_REGISTER = 3;

        /* Enable loopback for test */
        CT_UART.MODEM_CONTROL_REGISTER = 0x00;

        /* Choose desired response to emulation suspend events by configuring
         * FREE bit and enable UART by setting UTRST and URRST in PWREMU_MGMT */
        /* Allow UART to run free, enable UART TX/RX */
```

(continues on next page)

(continued from previous page)

```
        CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER = 0x6001;

        /*** END INITIALIZATION ***/

        /* Priming the 'hostbuffer' with a message */
        hostBuffer = "Hello!  This is a long string\r\n";

        /*** SEND SOME DATA ***/

        /* Let's send/receive some dummy data */
        while(1) {
                cnt = 0;
                while(1) {
                        /* Load character, ensure it is not string termination */
                        if ((tx = hostBuffer[cnt]) == '\0')
                                break;
                        cnt++;
                        CT_UART.RBR_THR_REGISTERS = tx;

                        /* Because we are doing loopback, wait until LSR.DR == 1
                         * indicating there is data in the RX FIFO */
                        while ((CT_UART.LINE_STATUS_REGISTER & 0x1) == 0x0);

                        /* Read the value from RBR */
                        rx = CT_UART.RBR_THR_REGISTERS;

                        /* Wait for TX FIFO to be empty */
                        while (!((CT_UART.INTERRUPT_IDENTIFICATION_REGISTER_FIFO_CONTROL_REGISTER & 0x2) == 0x2));
                }
        }

        /*** DONE SENDING DATA ***/

        /* Disable UART before halting */
        CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER = 0x0;

        /* Halt PRU core */
        __halt();
}
```

`uart1.pru1_0.c`

Note: I'm using the AI version of the code since it uses variables with more desciptive names.

The first part of the code initializes the UART. Then the line `CT_UART.RBR_THR_REGISTERS = tx;` takes a character in `tx` and sends it to the transmit buffer on the UART. Think of this as the UART version of the `printf()`.

Later the line `while (!((CT_UART.INTERRUPT_IDENTIFICATION_REGISTER_FIFO_CONTROL_REGISTER & 0x2) == 0x2));` waits for the transmitter FIFO to be empty. This makes sure later characters won't overwrite the buffer before they can be sent. The downside is, this will cause your code to wait on the buffer and it might miss an important real-time event.

The line `while ((CT_UART.LINE_STATUS_REGISTER & 0x1) == 0x0);` waits for an input from the UART (possibly missing something) and `rx = CT_UART.RBR_THR_REGISTERS;` reads from the receive register on the UART.

These simple lines should be enough to place in your code to print out debugging information.

Listing 15.79: uart2.pru0.c

```
// From: http://git.ti.com/pru-software-support-package/pru-software-support-package/trees/master/pru_cape/pru_fw/PRU_Hardware_UART

#include <stdint.h>
#include <pru_uart.h>
#include "resource_table_empty.h"

/* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily)
 * only going to send 8 at a time */
#define FIFO_SIZE        16
#define MAX_CHARS        8
#define BUFFER                40

//*****************************************************************************
//    Print Message Out
//      This function take in a string literal of any size and then fill the
//      TX FIFO when it's empty and waits until there is info in the RX FIFO
//      before returning.
//*****************************************************************************
void PrintMessageOut(volatile char* Message)
{
        uint8_t cnt, index = 0;

        while (1) {
                cnt = 0;

                /* Wait until the TX FIFO and the TX SR are completely empty */
                while (!CT_UART.LSR_bit.TEMT);

                while (Message[index] != NULL && cnt < MAX_CHARS) {
                        CT_UART.THR = Message[index];
                        index++;
                        cnt++;
                }
                if (Message[index] == NULL)
                        break;
        }

        /* Wait until the TX FIFO and the TX SR are completely empty */
        while (!CT_UART.LSR_bit.TEMT);

}

//*****************************************************************************
//    IEP Timer Config
//      This function waits until there is info in the RX FIFO and then returns
//      the first character entered.
//*****************************************************************************
char ReadMessageIn(void)
{
        while (!CT_UART.LSR_bit.DR);

        return CT_UART.RBR_bit.DATA;
}
```

(continues on next page)

(continued from previous page)

```

void main(void)
{
        uint32_t i;
        volatile uint32_t not_done = 1;

        char rxBuffer[BUFFER];
        rxBuffer[BUFFER-1] = NULL; // null terminate the string

        /*** INITIALIZATION ***/

        /* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x oversample
         * 192MHz / 104 / 16 = ~115200 */
        CT_UART.DLL = 104;
        CT_UART.DLH = 0;
        CT_UART.MDR_bit.OSM_SEL = 0x0;

        /* Enable Interrupts in UART module. This allows the main thread to poll for
         * Receive Data Available and Transmit Holding Register Empty */
        CT_UART.IER = 0x7;

        /* If FIFOs are to be used, select desired trigger level and enable
         * FIFOs by writing to FCR. FIFOEN bit in FCR must be set first before
         * other bits are configured */
        /* Enable FIFOs for now at 1-byte, and flush them */
        CT_UART.FCR = (0x80) | (0x8) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO trigger

        /* Choose desired protocol settings by writing to LCR */
        /* 8-bit word, 1 stop bit, no parity, no break control and no divisor latch */
        CT_UART.LCR = 3;

        /* If flow control is desired write appropriate values to MCR. */
        /* No flow control for now, but enable loopback for test */
        CT_UART.MCR = 0x00;

        /* Choose desired response to emulation suspend events by configuring
         * FREE bit and enable UART by setting UTRST and URRST in PWREMU_MGMT */
        /* Allow UART to run free, enable UART TX/RX */
        CT_UART.PWREMU_MGMT_bit.FREE = 0x1;
        CT_UART.PWREMU_MGMT_bit.URRST = 0x1;
        CT_UART.PWREMU_MGMT_bit.UTRST = 0x1;

        /* Turn off RTS and CTS functionality */
        CT_UART.MCR_bit.AFE = 0x0;
        CT_UART.MCR_bit.RTS = 0x0;

        /*** END INITIALIZATION ***/

        while(1) {
                /* Print out greeting message */
                PrintMessageOut("Hello you are in the PRU UART demo test please enter some characters\r\n");

                /* Read in characters from user, then echo them back out */
                for (i = 0; i < BUFFER-1 ; i++) {
```

(continues on next page)

(continued from previous page)

```
                        rxBuffer[i] = ReadMessageIn();
                        if(rxBuffer[i] == '\r') {        // Quit early if␣
↪ENTER is hit.
                                rxBuffer[i+1] = NULL;
                                break;
                        }
                }

                PrintMessageOut("you typed:\r\n");
                PrintMessageOut(rxBuffer);
                PrintMessageOut("\r\n");
        }

        /*** DONE SENDING DATA ***/
        /* Disable UART before halting */
        CT_UART.PWREMU_MGMT = 0x0;

        /* Halt PRU core */
        __halt();
}
```

`uart2.pru0.c`

If you want to try `uart2.pru0.c`, run the following:

Listing 15.80: make

```
bone$ *make TARGET=uart2.pru0*
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=uart2.pru0
-    Stopping PRU 0
-     copying firmware file /tmp/vsx-examples/uart2.pru0.out to /lib/
↪firmware/am335x-pru0-fw
write_init_pins.sh
-    Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR = /dev/remoteproc/pruss-core0
```

You will see:

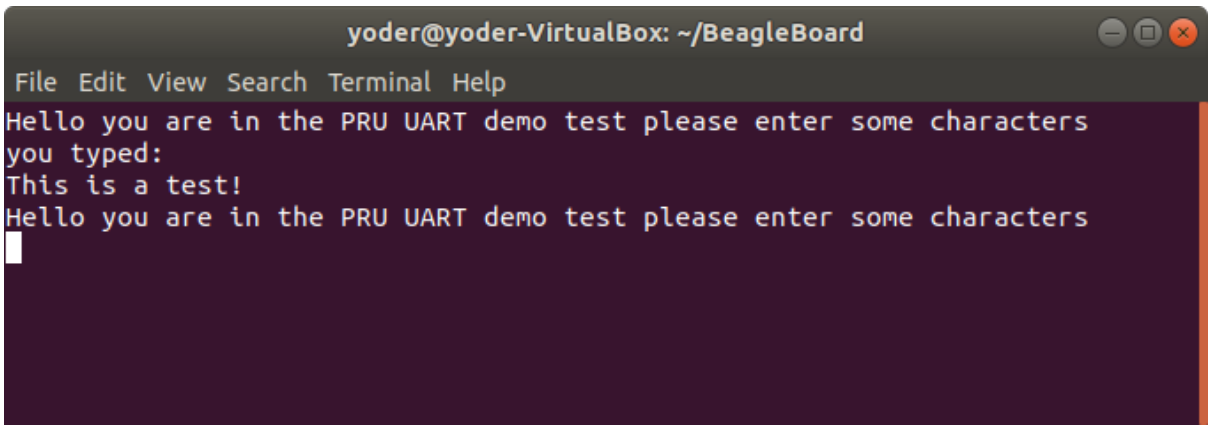


Fig. 15.146: uart2.pru0.c output

Type a few characters and hit ENTER. The PRU will playback what you typed, but it won't echo it as you type.

`uart2.pru0.c` defines `PrintMessageOut()` which is passed a string that is sent to the UART. It

takes advantage of the eight character FIFO on the UART. Be careful using it because it also uses `while (!CT_UART.LSR_bit.TEMT);` to wait for the FIFO to empty, which may cause your code to miss something.

uart2.pru1_0.c is the code that does it.

Listing 15.81: uart2.pru1_0.c

```
// From: http://git.ti.com/pru-software-support-package/pru-software-support-package/trees/master/pru_cape/pru_fw/PRU_Hardware_UART

#include <stdint.h>
#include <pru_uart.h>
#include "resource_table_empty.h"

/* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily)
 * only going to send 8 at a time */
#define FIFO_SIZE        16
#define MAX_CHARS        8
#define BUFFER                40

//*****************************************************************************
//    Print Message Out
//      This function take in a string literal of any size and then fill the
//      TX FIFO when it's empty and waits until there is info in the RX FIFO
//      before returning.
//*****************************************************************************
void PrintMessageOut(volatile char* Message)
{
        uint8_t cnt, index = 0;

        while (1) {
                cnt = 0;

                /* Wait until the TX FIFO and the TX SR are completely empty */
                while (!CT_UART.LINE_STATUS_REGISTER_bit.TEMT);

                while (Message[index] != NULL && cnt < MAX_CHARS) {
                        CT_UART.RBR_THR_REGISTERS = Message[index];
                        index++;
                        cnt++;
                }
                if (Message[index] == NULL)
                        break;
        }

        /* Wait until the TX FIFO and the TX SR are completely empty */
        while (!CT_UART.LINE_STATUS_REGISTER_bit.TEMT);

}

//*****************************************************************************
//    IEP Timer Config
//      This function waits until there is info in the RX FIFO and then returns
//      the first character entered.
//*****************************************************************************
char ReadMessageIn(void)
{
```

(continues on next page)

(continued from previous page)

```
        while (!CT_UART.LINE_STATUS_REGISTER_bit.DR);

        return CT_UART.RBR_THR_REGISTERS_bit.DATA;
}

void main(void)
{
        uint32_t i;
        volatile uint32_t not_done = 1;

        char rxBuffer[BUFFER];
        rxBuffer[BUFFER-1] = NULL; // null terminate the string

        /*** INITIALIZATION ***/

        /* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x oversample
         * 192MHz / 104 / 16 = ~115200 */
        CT_UART.DIVISOR_REGISTER_LSB_ = 104;
        CT_UART.DIVISOR_REGISTER_MSB_ = 0;
        CT_UART.MODE_DEFINITION_REGISTER_bit.OSM_SEL = 0x0;

        /* Enable Interrupts in UART module. This allows the main thread to poll for
         * Receive Data Available and Transmit Holding Register Empty */
        CT_UART.INTERRUPT_ENABLE_REGISTER = 0x7;

        /* If FIFOs are to be used, select desired trigger level and enable
         * FIFOs by writing to FCR. FIFOEN bit in FCR must be set first before
         * other bits are configured */
        /* Enable FIFOs for now at 1-byte, and flush them */
        CT_UART.INTERRUPT_IDENTIFICATION_REGISTER_FIFO_CONTROL_REGISTER = (0x80) | (0x8) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO trigger

        /* Choose desired protocol settings by writing to LCR */
        /* 8-bit word, 1 stop bit, no parity, no break control and no divisor latch */
        CT_UART.LINE_CONTROL_REGISTER = 3;

        /* If flow control is desired write appropriate values to MCR. */
        /* No flow control for now, but enable loopback for test */
        CT_UART.MODEM_CONTROL_REGISTER = 0x00;

        /* Choose desired response to emulation suspend events by configuring
         * FREE bit and enable UART by setting UTRST and URRST in PWREMU_MGMT */
        /* Allow UART to run free, enable UART TX/RX */
        CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER_bit.FREE = 0x1;
        CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER_bit.URRST = 0x1;
        CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER_bit.UTRST = 0x1;

        /* Turn off RTS and CTS functionality */
        CT_UART.MODEM_CONTROL_REGISTER_bit.AFE = 0x0;
        CT_UART.MODEM_CONTROL_REGISTER_bit.RTS = 0x0;

        /*** END INITIALIZATION ***/

        while(1) {
                /* Print out greeting message */
                PrintMessageOut("Hello you are in the PRU UART demo test please enter some characters\r\n");
```

(continues on next page)

(continued from previous page)

```
105
106                 /* Read in characters from user, then echo them back out */
107                 for (i = 0; i < BUFFER-1 ; i++) {
108                         rxBuffer[i] = ReadMessageIn();
109                         if(rxBuffer[i] == '\r') {        // Quit early if␣
↪ENTER is hit.
110                                 rxBuffer[i+1] = NULL;
111                                 break;
112                         }
113                 }
114
115                 PrintMessageOut("you typed:\r\n");
116                 PrintMessageOut(rxBuffer);
117                 PrintMessageOut("\r\n");
118         }
119
120         /*** DONE SENDING DATA ***/
121         /* Disable UART before halting */
122         CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER = 0x0;
123
124         /* Halt PRU core */
125         __halt();
126 }
```

uart2.pru1_0.c

More complex examples can be built using the principles shown in these examples.

Copyright

Listing 15.82: copyright.c

```
1  /*
2   * Copyright (C) 2015 Texas Instruments Incorporated - http://www.ti.com/
3   *
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   *
9   *        * Redistributions of source code must retain the above copyright
10  *          notice, this list of conditions and the following disclaimer.
11  *
12  *        * Redistributions in binary form must reproduce the above copyright
13  *          notice, this list of conditions and the following disclaimer in␣
↪the
14  *          documentation and/or other materials provided with the
15  *          distribution.
16  *
17  *        * Neither the name of Texas Instruments Incorporated nor the names␣
↪of
18  *          its contributors may be used to endorse or promote products␣
↪derived
19  *          from this software without specific prior written permission.
20  *
21  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
22  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
23  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
24  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
25  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
26  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
27  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
```

(continues on next page)

(continued from previous page)

```
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

copyright.c

15.2.5 Building Blocks - Applications

Here are some examples that use the basic PRU building blocks.

The following are resources used in this chapter.

Note: *Resources*

- PRU Optimizing C/C++ Compiler, v2.2, User's Guide
- AM572x Technical Reference Manual (AI)
- AM335x Technical Reference Manual (All others)
- Exploring BeagleBone by Derek Molloy
- WS2812 Data Sheet

Memory Allocation

Problem I want to control where my variables are stored in memory.

Solution Each PRU has is own 8KB of data memory (Data Mem0 and Mem1) and 12KB of shared memory (Shared RAM) as shown in *PRU Block Diagram*.

Each PRU accesses its own DRAM starting at location 0x0000_0000. Each PRU can also access the other PRU's DRAM starting at 0x0000_2000. Both PRUs access the shared RAM at 0x0001_0000. The compiler can control where each of these memories variables are stored.

shared.pro0.c - Examples of Using Different Memory Locations shows how to allocate seven variable in six different locations.

Listing 15.83: shared.pro0.c - Examples of Using Different Memory Locations

```
// From: http://git.ti.com/pru-software-support-package/pru-software-support-
↪package/blobs/master/examples/am335x/PRU_access_const_table/PRU_access_
↪const_table.c
#include <stdint.h>
#include <pru_cfg.h>
#include <pru_ctrl.h>
#include "resource_table_empty.h"

#define PRU_SRAM  __far __attribute__((cregister("PRU_SHAREDMEM", near)))
#define PRU_DMEM0 __far __attribute__((cregister("PRU_DMEM_0_1",  near)))
#define PRU_DMEM1 __far __attribute__((cregister("PRU_DMEM_1_0",  near)))

/* NOTE:  Allocating shared_x to PRU Shared Memory means that other PRU␣
↪cores on
 *        the same subsystem must take care not to allocate data to that␣
↪memory.
```

(continues on next page)

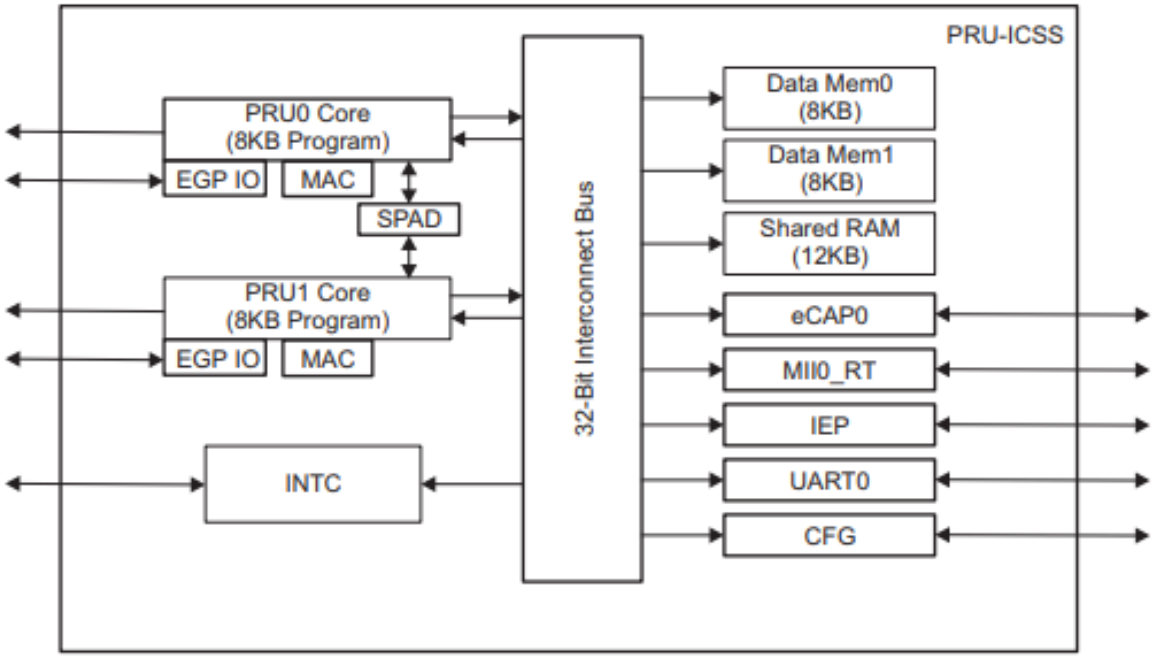


Fig. 15.147: PRU Block Diagram

(continued from previous page)

```
 *                  Users also cannot rely on where in shared memory these variables are placed
 *        so accessing them from another PRU core or from the ARM is an undefined behavior.
 */
volatile uint32_t shared_0;
PRU_SRAM  volatile uint32_t shared_1;
PRU_DMEM0 volatile uint32_t shared_2;
PRU_DMEM1 volatile uint32_t shared_3;
#pragma DATA_SECTION(shared_4, ".bss")
volatile uint32_t shared_4;

/* NOTE:  Here we pick where in memory to store shared_5.  The stack and
 *                  heap take up the first 0x200 words, so we must start after that.
 *                  Since we are hardcoding where things are stored we can share
 *                  this between the PRUs and the ARM.
*/
#define PRU0_DRAM               0x00000                         // Offset to DRAM
// Skip the first 0x200 bytes of DRAM since the Makefile allocates
// 0x100 for the STACK and 0x100 for the HEAP.
volatile unsigned int *shared_5 = (unsigned int *) (PRU0_DRAM + 0x200);


int main(void)
{
        volatile uint32_t shared_6;
        volatile uint32_t shared_7;
        /*****************************************************************/
        /* Access PRU peripherals using Constant Table & PRU header file */
        /*****************************************************************/
```

(continues on next page)

(continued from previous page)

```
41
42          /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
43          CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
44
45          /*****************************************************************/
46          /* Access PRU Shared RAM using Constant Table                    */
47          /*****************************************************************/
48
49          /* C28 defaults to 0x00000000, we need to set bits 23:8 to 0x0100 in␣
    ↪order to have it point to 0x00010000         */
50          PRU0_CTRL.CTPPR0_bit.C28_BLK_POINTER = 0x0100;
51
52          shared_0 =  0xfeef;
53          shared_1 = 0xdeadbeef;
54          shared_2 = shared_2 + 0xfeed;
55          shared_3 = 0xdeed;
56          shared_4 = 0xbeed;
57          shared_5[0] = 0x1234;
58          shared_6 = 0x4321;
59          shared_7 = 0x9876;
60
61          /* Halt PRU core */
62          __halt();
63  }
```

shared.pru0.c

Discussion Here's the line-by-line

Table 15.14: Line-byline for shared.pru0.c

Line	Explanation
7	*PRU_SRAM* is defined here. It will be used later to declare variables in the *Shared RAM* location of memory. Section 5.5.2 on page 75 of the PRU Optimizing C/C++ Compiler, v2.2, User's Guide gives details of the command. The *PRU_SHAREDMEM* refers to the memory section defined in *am335x_pru.cmd* on line 26.
8, 9	These are like the previous line except for the DMEM sections.
16	Variables declared outside of *main()* are put on the heap.
17	Adding *PRU_SRAM* has the variable stored in the shared memory.
18, 19	These are stored in the PRU's local RAM.
20, 21	These lines are for storing in the *.bss* section as declared on line 74 of *am335x_pru.cmd*.
28-31	All the previous examples direct the compiler to an area in memory and the compilers figures out what to put where. With these lines we specify the exact location. Here are start with the PRU_DRAM starting address and add 0x200 to it to avoid the **stack** and the **heap**. The advantage of this technique is you can easily share these variables between the ARM and the two PRUs.
36, 37	Variable declared inside *main()* go on the stack.

Caution: Using the technique of line 28-31 you can put variables anywhere, even where the compiler has put them. Be careful, it's easy to overwrite what the compiler has done

Compile and run the program.

```
bone$ *source shared_setup.sh*
TARGET=shared.pru0
Black Found
P9_31
Current mode for P9_31 is:     pruout
Current mode for P9_31 is:     pruout
```

(continues on next page)

(continued from previous page)

```
P9_29
Current mode for P9_29 is:     pruout
Current mode for P9_29 is:     pruout
P9_30
Current mode for P9_30 is:     pruout
Current mode for P9_30 is:     pruout
P9_28
Current mode for P9_28 is:     pruout
Current mode for P9_28 is:     pruout
bone$ *make*
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=shared.pru0
-    Stopping PRU 0
-     copying firmware file /tmp/vsx-examples/shared.pru0.out to /lib/
↪firmware/am335x-pru0-fw
write_init_pins.sh
-    Starting PRU 0
MODEL   = TI_AM335x_BeagleBone_Black
PROC    = pru
PRUN    = 0
PRU_DIR = /sys/class/remoteproc/remoteproc1
```

Now check the **symbol table** to see where things are allocated.

```
bone $ *grep shared /tmp/vsx-examples/shared.pru0.map*
....
1     0000011c  shared_0
2     00010000  shared_1
1     00000000  shared_2
1     00002000  shared_3
1     00000118  shared_4
1     00000120  shared_5
```

We see, `shared_0` had no directives and was places in the heap that is 0x100 to 0x1ff. `shared_1` was directed to go to the SHAREDMEM, `shared_2` to the start of the local DRAM (which is also the top of the stack). `shared_3` was placed in the DRAM of PRU 1, `shared_4` was placed in the `.bss` section, which is in the **heap**. Finally `shared_5` is a pointer to where the value is stored.

Where are `shared_6` and `shared_7`? They are declared inside `main()` and are therefore placed on the stack at run time. The `shared.map` file shows the compile time allocations. We have to look in the memory itself to see what happen at run time.

Let's fire up `prudebug` (*prudebug - A Simple Debugger for the PRU*) to see where things are.

```
bone$ *sudo ./prudebug*
PRU Debugger v0.25
(C) Copyright 2011, 2013 by Arctica Technologies.  All rights reserved.
Written by Steven Anderson

Using /dev/mem device.
Processor type                AM335x
PRUSS memory address  0x4a300000
PRUSS memory length   0x00080000

        offsets below are in 32-bit byte addresses (not ARM byte addresses)
        PRU            Instruction    Data         Ctrl
        0              0x00034000     0x00000000   0x00022000
        1              0x00038000     0x00002000   0x00024000

PRU0> *d 0*
Absolute addr = 0x0000, offset = 0x0000, Len = 16
[0x0000] 0x0000feed 0x00000000 0x00000000 0x00000000
```

(continues on next page)

(continued from previous page)

```
[0x0010] 0x00000000 0x00000000 0x00000000 0x00000000
[0x0020] 0x00000000 0x00000000 0x00000000 0x00000000
[0x0030] 0x00000000 0x00000000 0x00000000 0x00000000
```

The value of `shared_2` is in memory location 0.

```
PRU0> *dd 0x100*
Absolute addr = 0x0100, offset = 0x0000, Len = 16
[0x0100] 0x00000000 0x00000001 0x00000000 0x00000000
[0x0110] 0x00000000 0x00000000 0x0000beed 0x0000feef
[0x0120] 0x00000200 0x3ec71de3 0x1a013e1a 0xbf2a01a0
[0x0130] 0x111110b0 0x3f811111 0x55555555 0xbfc55555
```

There are `shared_0` and `shared_4` in the heap, but where is `shared_6` and `shared_7`? They are supposed to be on the **stack** that starts at 0.

```
PRU0> dd *0xc0*
Absolute addr = 0x00c0, offset = 0x0000, Len = 16
[0x00c0] 0x00000000 0x00000000 0x00000000 0x00000000
[0x00d0] 0x00000000 0x00000000 0x00000000 0x00000000
[0x00e0] 0x00000000 0x00000000 0x00000000 0x00000000
[0x00f0] 0x00000000 0x00000000 0x00004321 0x00009876
```

There they are; the stack grows from the top. (The heap grows from the bottom.)

```
PRU0> dd *0x2000*
Absolute addr = 0x2000, offset = 0x0000, Len = 16
[0x2000] 0x0000deed 0x00000001 0x00000000 0x557fcfb5
[0x2010] 0xce97bd0f 0x6afb2c8f 0xc7f35df4 0x5afb6dcb
[0x2020] 0x8dec3da3 0xe39a6756 0x642cb8b8 0xcb6952c0
[0x2030] 0x2f22ebda 0x548d97c5 0x9241786f 0x72dfeb86
```

And there is PRU 1's memory with `shared_3`. And finally the shared memory.

```
PRU0> *dd 0x10000*
Absolute addr = 0x10000, offset = 0x0000, Len = 16
[0x10000] 0xdeadbeef 0x0000feed 0x00000000 0x68c44f8b
[0x10010] 0xc372ba7e 0x2ffa993b 0x11c66da5 0xfbf6c5d7
[0x10020] 0x5ada3fcf 0x4a5d0712 0x48576fb7 0x1004796b
[0x10030] 0x2267ebc6 0xa2793aa1 0x100d34dc 0x9ca06d4a
```

The compiler offers great control over where variables are stored. Just be sure if you are hand picking where things are put, not to put them in places used by the compiler.

Auto Initialization of built-in LED Triggers

Problem I see the built-in LEDs blink to their own patterns. How do I turn this off? Can this be automated?

Solution Each built-in LED has a default action (trigger) when the Bone boots up. This is controlled by `/sys/class/leds`.

```
bone$ *cd /sys/class/leds*
bone$ *ls*
beaglebone:green:usr0  beaglebone:green:usr2
beaglebone:green:usr1  beaglebone:green:usr3
```

Here you see a directory for each of the LEDs. Let's pick USR1.

```
bone$ *cd beaglebone\:green\:usr1*
bone$ *ls*
brightness  device  max_brightness  power  subsystem  trigger  uevent
bone$ *cat trigger*
none usb-gadget usb-host rfkill-any rfkill-none kbd-scrolllock kbd-numlock
kbd-capslock kbd-kanalock kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-
↪altlock
kbd-shiftllock kbd-shiftrlock kbd-ctrlllock kbd-ctrlrlock *[mmc0]* timer
oneshot disk-activity disk-read disk-write ide-disk mtd nand-disk heartbeat
backlight gpio cpu cpu0 activity default-on panic netdev phy0rx phy0tx
phy0assoc phy0radio rfkill0
```

Notice `[mmc0]` is in brackets. This means it's the current trigger; it flashes when the built-in flash memory is in use. You can turn this off using:

```
bone$ *echo none > trigger*
bone$ *cat trigger*
*[none]* usb-gadget usb-host rfkill-any rfkill-none kbd-scrolllock kbd-
↪numlock
kbd-capslock kbd-kanalock kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-
↪altlock
kbd-shiftllock kbd-shiftrlock kbd-ctrlllock kbd-ctrlrlock mmc0 timer
oneshot disk-activity disk-read disk-write ide-disk mtd nand-disk heartbeat
backlight gpio cpu cpu0 activity default-on panic netdev phy0rx phy0tx
phy0assoc phy0radio rfkill0
```

Now it is no longer flashing.

How can this be automated so when code is run that needs the trigger off, it's turned off automatically? Here's a trick. Include the following in your code.

```
1 #pragma DATA_SECTION(init_pins, ".init_pins")
2 #pragma RETAIN(init_pins)
3 const char init_pins[] =
4         "/sys/class/leds/beaglebone:green:usr3/trigger\0none\0" \
5         "\0\0";
```

Lines 3 and 4 declare the array `init_pins` to have an entry which is the path to `trigger` and the value that should be 'echoed' into it. Both are NULL terminated. Line 1 says to put this in a section called `.init_pins` and line 2 says to `RETAIN` it. That is don't throw it away if it appears to be unused.

Discussion The above code stores this array in the `.out` file thats created, but that's not enough. You need to run *write_init_pins.sh* on the `.out` file to make the code work. Fortunately the Makefile always runs it.

Listing 15.84: write_init_pins.sh

```
1 #!/bin/bash
2 init_pins=$(readelf -x .init_pins $1 | grep 0x000 | cut -d' ' -f4-7 | xxd -r␣
  ↪-p | tr '\0' '\n' | paste - -)
3 while read -a line; do
4     if [ ${#line[@]} == 2 ]; then
5         echo writing \"${line[1]}\" to \"${line[0]}\"
6         echo ${line[1]} > ${line[0]}
7         sleep 0.1
8     fi
9 done <<< "$init_pins"
```

`write_init_pins.sh`

The `readelf` command extracts the path and value from the `.out` file.

```
bone$ *readelf -x .init_pins /tmp/pru0-gen/shared.out*

Hex dump of section '.init_pins':
  0x000000c0 2f737973 2f636c61 73732f6c 6564732f /sys/class/leds/
  0x000000d0 62656167 6c65626f 6e653a67 7265656e beaglebone:green
  0x000000e0 3a757372 332f7472 69676765 72006e6f :usr3/trigger.no
  0x000000f0 6e650000 0000                       ne....
```

The rest of the command formats it. Finally line 6 echos the `none` into the path.

This can be generalized to initialize other things. The point is, the `.out` file contains everything needed to run the executable.

PWM Generator

One of the simplest things a PRU can to is generate a simple signal starting with a single channel PWM that has a fixed frequency and duty cycle and ending with a multi channel PWM that the ARM can change the frequency and duty cycle on the fly.

Problem I want to generate a PWM signal that has a fixed frequency and duty cycle.

Solution The solution is fairly easy, but be sure to check the *Discussion* section for details on making it work.

pwm1.pru0.c shows the code.

Warning: This code is for the BeagleBone Black. See `pwm1.pru1_1.c` for an example that works on the AI.

Listing 15.85: pwm1.pru0.c

```
1  #include <stdint.h>
2  #include <pru_cfg.h>
3  #include "resource_table_empty.h"
4  #include "prugpio.h"
5
6  volatile register uint32_t __R30;
7  volatile register uint32_t __R31;
8
9  void main(void)
10 {
11         uint32_t gpio = P9_31;         // Select which pin to toggle.;
12
13         /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
14         CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
15
16         while(1) {
17                 __R30 |= gpio;                 // Set the GPIO pin to 1
18                 __delay_cycles(100000000);
19                 __R30 &= ~gpio;                 // Clear the GPIO pin
20                 __delay_cycles(100000000);
21         }
22 }
```

`pwm1.pru0.c`

To run this code you need to configure the pin muxes to output the PRU. If you are on the Black run

```
bone$ config-pin P9_31 pruout
```

On the Pocket run

```
bone$ config-pin P1_36 pruout
```

Note: See *Configuring pins on the AI via device trees* for configuring pins on the AI.

Then, tell `Makefile` which PRU you are compiling for and what your target file is

```
bone$ export TARGET=pwm1.pru0
```

Now you are ready to compile

```
bone$ make
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=pwm1.pru0
-    Stopping PRU 0
-     copying firmware file /tmp/vsx-examples/pwm1.pru0.out to /lib/firmware/
↪am335x-pru0-fw
write_init_pins.sh
-    Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR = /sys/class/remoteproc/remoteproc1
```

Now attach an LED (or oscilloscope) to `P9_31` on the Black or `P1.36` on the Pocket. You should see a squarewave.

Discussion Since this is our first example we'll discuss the many parts in detail.

Listing 15.86: pwm1.pru0.c

```
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t gpio = P9_31;         // Select which pin to toggle.;

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        while(1) {
                __R30 |= gpio;                 // Set the GPIO pin to 1
                __delay_cycles(100000000);
                __R30 &= ~gpio;                 // Clear the GPIO pin
                __delay_cycles(100000000);
        }
}
```

`pwm1.pru0.c`

Line-by-line of pwm1.pru0.c is a line-by-line expanation of the c code.

Table 15.15: Line-by-line of pwm1.pru0.c

Line	Explanation
1	Standard c-header include
2	Include for the PRU. The compiler knows where to find this since the *Makefile* says to look for includes in */usr/lib/ti/pru-software-support-package*
3	The file *resource_table_empty.h* is used by the PRU loader. Generally we'll use the same file, and don't need to modify it.
4	This include has addresses for the GPIO ports and some bit positions for some of the headers.

Here's what's in `resource_table_empty.h`

Listing 15.87: resource_table_empty.c

```
/*
 *  ======== resource_table_empty.h ========
 *
 *  Define the resource table entries for all PRU cores. This will be
 *  incorporated into corresponding base images, and used by the remoteproc
 *  on the host-side to allocated/reserve resources.  Note the remoteproc
 *  driver requires that all PRU firmware be built with a resource table.
 *
 *  This file contains an empty resource table.  It can be used either as:
 *
 *        1) A template, or
 *        2) As-is if a PRU application does not need to configure PRU_INTC
 *                or interact with the rpmsg driver
 *
 */

#ifndef _RSC_TABLE_PRU_H_
#define _RSC_TABLE_PRU_H_

#include <stddef.h>
#include <rsc_types.h>

struct my_resource_table {
        struct resource_table base;

        uint32_t offset[1]; /* Should match 'num' in actual definition */
};

#pragma DATA_SECTION(pru_remoteproc_ResourceTable, ".resource_table")
#pragma RETAIN(pru_remoteproc_ResourceTable)
struct my_resource_table pru_remoteproc_ResourceTable = {
        1,          /* we're the first version that implements this */
        0,          /* number of entries in the table */
        0, 0,       /* reserved, must be zero */
        0,          /* offset[0] */
};

#endif /* _RSC_TABLE_PRU_H_ */
```

`resource_table_empty.c`

Table 15.16: Line-by-line (continuted)

Line	Explanation
6-7	`__R30` and `__R31` are two variables that refer to the PRU output (`__R30`) and input (`__R31`) registers. When you write something to `__R30` it will show up on the corresponding output pins. When you read from `__R31` you read the data on the input pins. NOTE: Both names begin with two underscore's. Section 5.7.2 of the PRU Optimizing C/C++ Compiler, v2.2, User's Guide gives more details.
11	This line selects which GPIO pin to toggle. The table below shows which bits in `__R30` map to which pins
14	*CT_CFG.SYSCFG_bit.STANDBY_INIT* is set to *0* to enable the OCP master port. More details on this and thousands of other regesters see the TI AM335x TRM. Section 4 is on the PRU and section 4.5 gives details for all the registers.

Bit 0 is the LSB.

Table 15.17: Mapping bit positions to pin names

PRU	Bit	Black pin	Pocket pin
0	0	P9_31	P1.36
0	1	P9_29	P1.33
0	2	P9_30	P2.32
0	3	P9_28	P2.30
0	4	P9_42b	P1.31
0	5	P9_27	P2.34
0	6	P9_41b	P2.28
0	7	P9_25	P1.29
0	14	P8_12(out) P8_16(in)	P2.24
0	15	P8_11(out) P8_15(in)	P2.33
1	0	P8_45	
1	1	P8_46	
1	2	P8_43	
1	3	P8_44	
1	4	P8_41	
1	5	P8_42	
1	6	P8_39	
1	7	P8_40	
1	8	P8_27	P2.35
1	9	P8_29	P2.01
1	10	P8_28	P1.35
1	11	P8_30	P1.04
1	12	P8_21	
1	13	P8_20	
1	14		P1.32
1	15		P1.30
1	16	P9_26(in)|	

Note: See *Configuring pins on the AI via device trees* for all the PRU pins on the AI.

Since we are running on PRU 0, and we're using `0x0001`, that is bit 0, we'll be toggling `P9_31`.

Table 15.18: Line-by-line (continued again)

Line	Explanation
17	Here is where the action is. This line reads `__R30` and then ORs it with `gpio`, setting the bits where there is a 1 in `gpio` and leaving the bits where there is a 0. Thus we are setting the bit we selected. Finally the new value is written back to `__R30`.
18	`__delay_cycles` is an ((intrinsic function)) that delays with number of cycles passed to it. Each cycle is 5ns, and we are delaying 100,000,000 cycles which is 500,000,000ns, or 0.5 seconds.
19	This is like line 17, but `~gpio` inverts all the bits in `gpio` so that where we had a 1, there is now a 0. This 0 is then ANDed with `__R30` setting the corresponding bit to 0. Thus we are clearing the bit we selected.

Tip: You can read more about intrinsics in section 5.11 of the (PRU Optimizing C/C++ Compiler, v2.2, User's Guide.)

When you run this code and look at the output you will see something like the following figure.

Notice the on time (`+Width(1)`) is 500ms, just as we predicted. The off time is 498ms, which is only 2ms off from our prediction. The standard deviation is 0, or only 380as, which is 380 * 10^-18^!.

You can see how fast the PRU can run by setting both of the `__delay_cycles` to 0. This results in the next figure.

Notice the period is 15ns which gives us a frequency of about 67MHz. At this high frequency the breadboard that I'm using distorts the waveform so it's no longer a squarewave. The **on** time is 5.3ns and the **off** time is 9.8ns. That means **__R30 |= gpio** took only one 5ns cycle and `__R30 &= ~gpio` also only took one cycle, but there is also an extra cycle needed for the loop. This means the compiler was able to implement the `while` loop in just three 5ns instructions! Not bad.

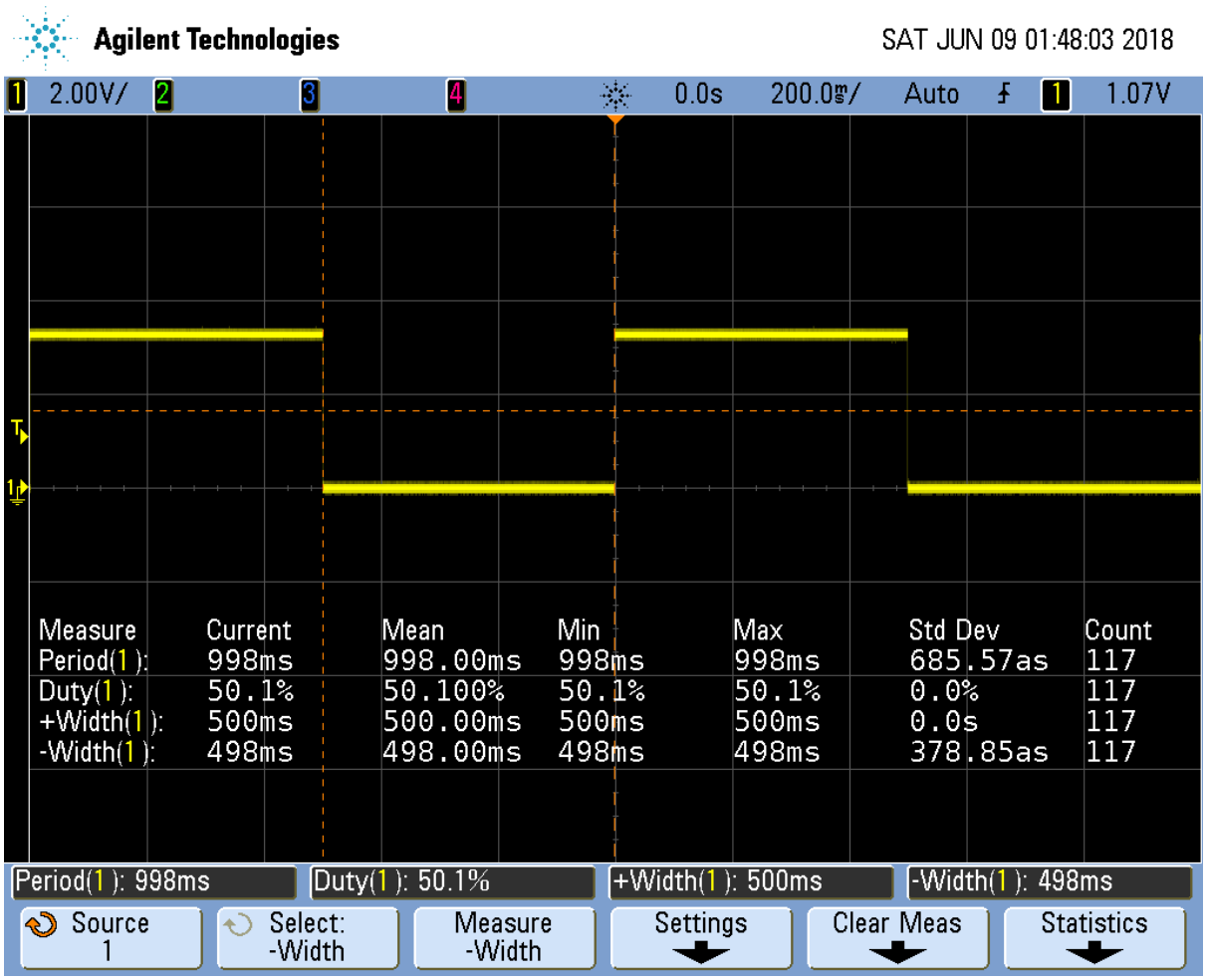


Fig. 15.148: Output of pwm1.pru0.c with 100,000,000 delays cycles giving a 1s period

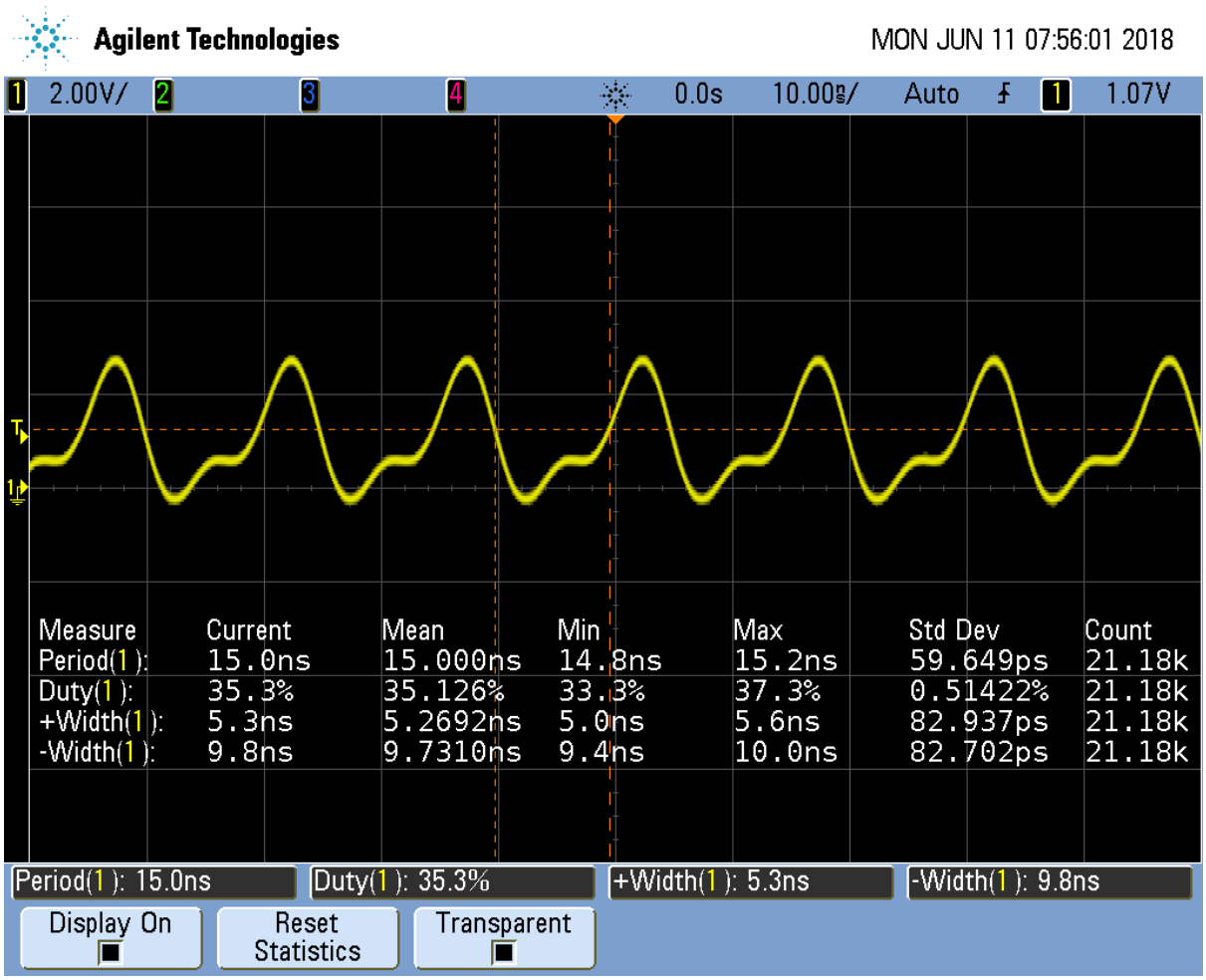


Fig. 15.149: Output of pwm1.pru0c with 0 delay cycles

We want a square wave, so we need to add a delay to correct for the delay of looping back.

Here's the code that does just that.

Listing 15.88: pwm2.pru0.c

```
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t gpio = P9_31;         // Select which pin to toggle.;

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        while (1) {
                __R30 |= gpio;                 // Set the GPIO pin to 1
                __delay_cycles(1);         // Delay one cycle to correct for loop time
                __R30 &= ~gpio;                 // Clear the GPIO pin
                __delay_cycles(0);
        }
}
```

pwm2.pru0.c

The output now looks like:

It's not hard to adjust the two `__delay_cycles` to get the desired frequency and duty cycle.

Controlling the PWM Frequency

Problem You would like to control the frequency and duty cycle of the PWM without recompiling.

Solution Have the PRU read the **on** and **off** times from a shared memory location. Each PRU has is own 8KB of data memory (DRAM) and 12KB of shared memory (SHAREDMEM) that the ARM processor can also access. See *PRU Block Diagram*.

The DRAM 0 address is 0x0000 for PRU 0. The same DRAM appears at address 0x4A300000 as seen from the ARM processor.

Tip: See page 184 of the AM335x TRM (184).

We take the previous PRU code and add the lines

```
#define PRU0_DRAM             0x00000                 // Offset to DRAM
volatile unsigned int *pru0_dram = PRU0_DRAM;
```

to define a pointer to the DRAM.

Note: The *volatile* keyword is used here to tell the compiler the value this points to may change, so don't make any assumptions while optimizing.

Later in the code we use

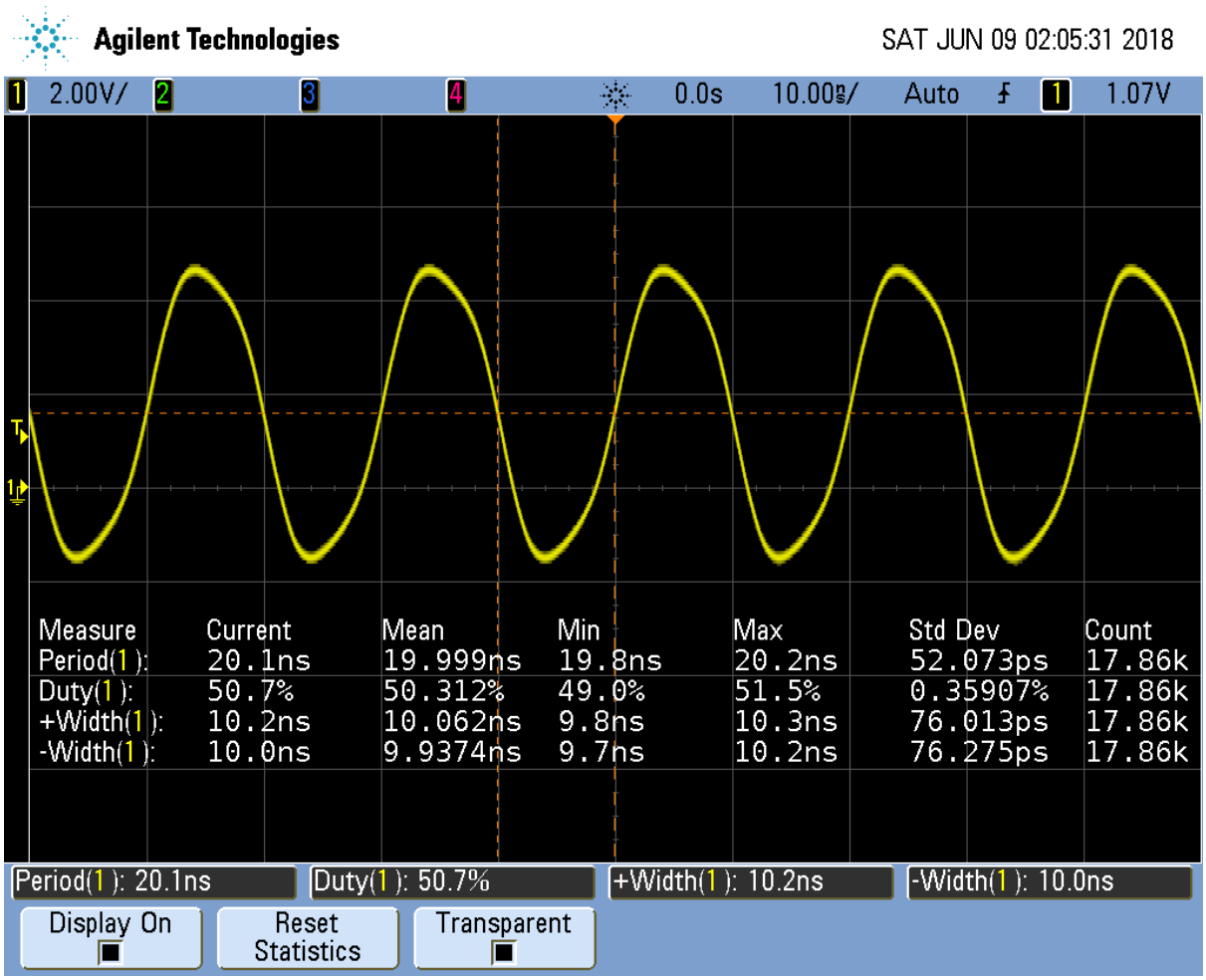


Fig. 15.150: Output of pwm2.pru0.c corrected delay

```
pru0_dram[ch] = on[ch];              // Copy to DRAM0 so the ARM can change it
pru0_dram[ch+MAXCH] = off[ch];  // Copy after the on array
```

to write the *on* and *off* times to the DRAM. Then inside the *while* loop we use

```
onCount[ch] = pru0_dram[2*ch];          // Read from DRAM0
offCount[ch]= pru0_dram[2*ch+1];
```

to read from the DRAM when resetting the counters. Now, while the PRU is running, the ARM can write values into the DRAM and change the PWM on and off times. *pwm4.pru0.c* is the whole code.

Listing 15.89: pwm4.pru0.c

```
// This code does MAXCH parallel PWM channels.
// It's period is 3 us
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"

#define PRU0_DRAM                0x00000                         // Offset to DRAM
// Skip the first 0x200 byte of DRAM since the Makefile allocates
// 0x100 for the STACK and 0x100 for the HEAP.
volatile unsigned int *pru0_dram = (unsigned int *) (PRU0_DRAM + 0x200);

#define MAXCH        4        // Maximum number of channels per PRU

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t ch;
        uint32_t on[]  = {1, 2, 3, 4};         // Number of cycles to stay on
        uint32_t off[] = {4, 3, 2, 1};         // Number to stay off
        uint32_t onCount[MAXCH];                 // Current count
        uint32_t offCount[MAXCH];

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        // Initialize the channel counters.
        for(ch=0; ch<MAXCH; ch++) {
                pru0_dram[2*ch  ] = on[ch];                // Copy to DRAM0 so the ARM can change it
                pru0_dram[2*ch+1] = off[ch];        // Interleave the on and off values
                onCount[ch] = on[ch];
                offCount[ch]= off[ch];
        }

        while (1) {
                for(ch=0; ch<MAXCH; ch++) {
                        if(onCount[ch]) {
                                onCount[ch]--;
                                __R30 |= 0x1<<ch;                // Set the GPIO pin to 1
                        } else if(offCount[ch]) {
                                offCount[ch]--;
                                __R30 &= ~(0x1<<ch);        // Clear the GPIO pin
                        } else {
```

(continues on next page)

(continued from previous page)

```
                                onCount[ch] = pru0_dram[2*ch]; // Read from DRAM0
                                offCount[ch]= pru0_dram[2*ch+1];
                        }
                }
        }
}
```

pwm4.pru0.c

Here is code that runs on the ARM side to set the on and off time values.

Listing 15.90: pwm-test.c

```
/*
 *
 *  pwm tester
 *        The on cycle and off cycles are stored in each PRU's Data memory
 *
 */

#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>

#define MAXCH 4

#define PRU_ADDR                0x4A300000              // Start of PRU memory Page 184 am335x TRM
#define PRU_LEN                         0x80000                 //Length of PRU memory
#define PRU0_DRAM               0x00000                 // Offset to DRAM
#define PRU1_DRAM               0x02000
#define PRU_SHAREDMEM       0x10000                 // Offset to shared memory

unsigned int        *pru0DRAM_32int_ptr;            // Points to the start of local DRAM
unsigned int        *pru1DRAM_32int_ptr;            // Points to the start of local DRAM
unsigned int        *prusharedMem_32int_ptr;        // Points to the start of the shared memory

/*****************************************************************************
* int start_pwm_count(int ch, int countOn, int countOff)
*
* Starts a pwm pulse on for countOn and off for countOff to a single channel (ch)
*****************************************************************************/
int start_pwm_count(int ch, int countOn, int countOff) {
        unsigned int *pruDRAM_32int_ptr = pru0DRAM_32int_ptr;

        printf("countOn: %d, countOff: %d, count: %d\n",
                countOn, countOff, countOn+countOff);
        // write to PRU shared memory
        pruDRAM_32int_ptr[2*(ch)+0] = countOn;          // On time
        pruDRAM_32int_ptr[2*(ch)+1] = countOff;         // Off time
        return 0;
}
```

(continues on next page)

(continued from previous page)

```
39
40 int main(int argc, char *argv[])
41 {
42         unsigned int        *pru;                   // Points to start of PRU␣
   ↪memory.
43         int         fd;
44         printf("Servo tester\n");
45
46         fd = open ("/dev/mem", O_RDWR | O_SYNC);
47         if (fd == -1) {
48                 printf ("ERROR: could not open /dev/mem.\n\n");
49                 return 1;
50         }
51         pru = mmap (0, PRU_LEN, PROT_READ | PROT_WRITE, MAP_SHARED, fd, PRU_
   ↪ADDR);
52         if (pru == MAP_FAILED) {
53                 printf ("ERROR: could not map memory.\n\n");
54                 return 1;
55         }
56         close(fd);
57         printf ("Using /dev/mem.\n");
58
59         pru0DRAM_32int_ptr =     pru + PRU0_DRAM/4 + 0x200/4;         //␣
   ↪Points to 0x200 of PRU0 memory
60         pru1DRAM_32int_ptr =     pru + PRU1_DRAM/4 + 0x200/4;         //␣
   ↪Points to 0x200 of PRU1 memory
61         prusharedMem_32int_ptr = pru + PRU_SHAREDMEM/4;         // Points to␣
   ↪start of shared memory
62
63         int i;
64         for(i=0; i<MAXCH; i++) {
65                 start_pwm_count(i, i+1, 20-(i+1));
66         }
67
68         if(munmap(pru, PRU_LEN)) {
69                 printf("munmap failed\n");
70         } else {
71                 printf("munmap succeeded\n");
72         }
73 }
```

`pwm-test.c`

A quick check on the 'scope shows *Four Channel PWM with ARM control*.

From the 'scope you see a 1 cycle **on** time results in a 450ns wide pulse and a 3.06us period is 326KHz, much slower than the 10ns pulse we saw before. But it may be more than fast enough for many applications. For example, most servos run at 50Hz.

But we can do better.

Loop Unrolling for Better Performance

Problem The ARM controlled PRU code runs too slowly.

Solution Simple loop unrolling can greatly improve the speed. `pwm5.pru0.c` is our unrolled version.

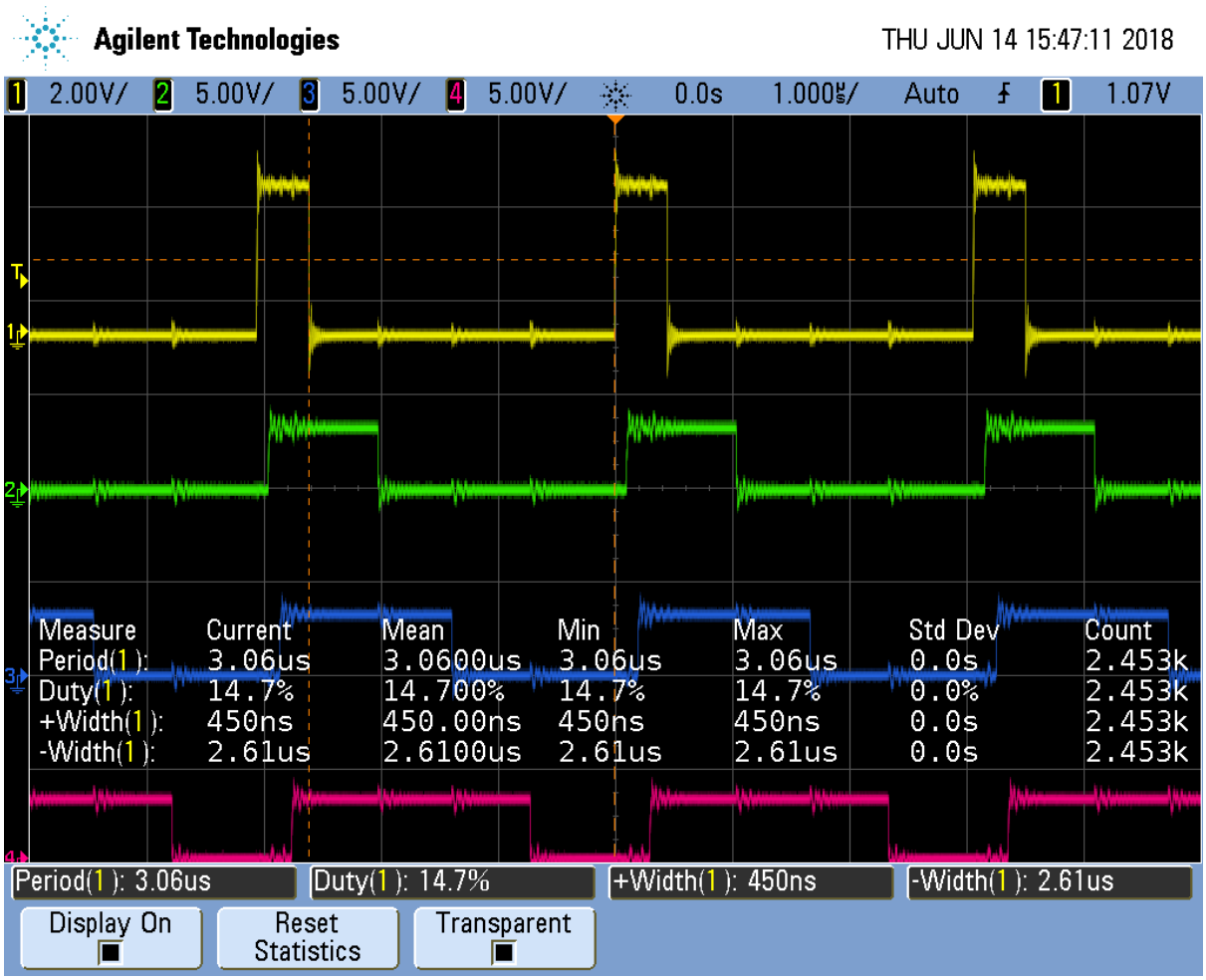


Fig. 15.151: Four Channel PWM with ARM control

Listing 15.91: pwm5.pru0.c Unrolled

```
// This code does MAXCH parallel PWM channels.
// It's period is 510ns.
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"

#define PRU0_DRAM               0x00000                         // Offset to DRAM
// Skip the first 0x200 byte of DRAM since the Makefile allocates
// 0x100 for the STACK and 0x100 for the HEAP.
volatile unsigned int *pru0_dram = (unsigned int *) (PRU0_DRAM + 0x200);

#define MAXCH        4        // Maximum number of channels per PRU

#define update(ch) \
                        if(onCount[ch]) {                           \
                                onCount[ch]--;                          \
                                __R30 |= 0x1<<ch;                   \
                        } else if(offCount[ch]) {           \
                                offCount[ch]--;                          \
                                __R30 &= ~(0x1<<ch);            \
                        } else {                                            \
                                onCount[ch] = pru0_dram[2*ch];          \
                                offCount[ch]= pru0_dram[2*ch+1];            \
                        }

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t ch;
        uint32_t on[]  = {1, 2, 3, 4};
        uint32_t off[] = {4, 3, 2, 1};
        uint32_t onCount[MAXCH], offCount[MAXCH];

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

#pragma UNROLL(MAXCH)
        for(ch=0; ch<MAXCH; ch++) {
                pru0_dram[2*ch  ] = on[ch];                 // Copy to DRAM0 so the ARM can change it
                pru0_dram[2*ch+1] = off[ch];        // Interleave the on and off values
                onCount[ch] = on[ch];
                offCount[ch]= off[ch];
        }

        while (1) {
                update(0)
                update(1)
                update(2)
                update(3)
        }
}
```

`pwm5.pru0.c`

The output of `pwm5.pru0.c` is in the figure below.

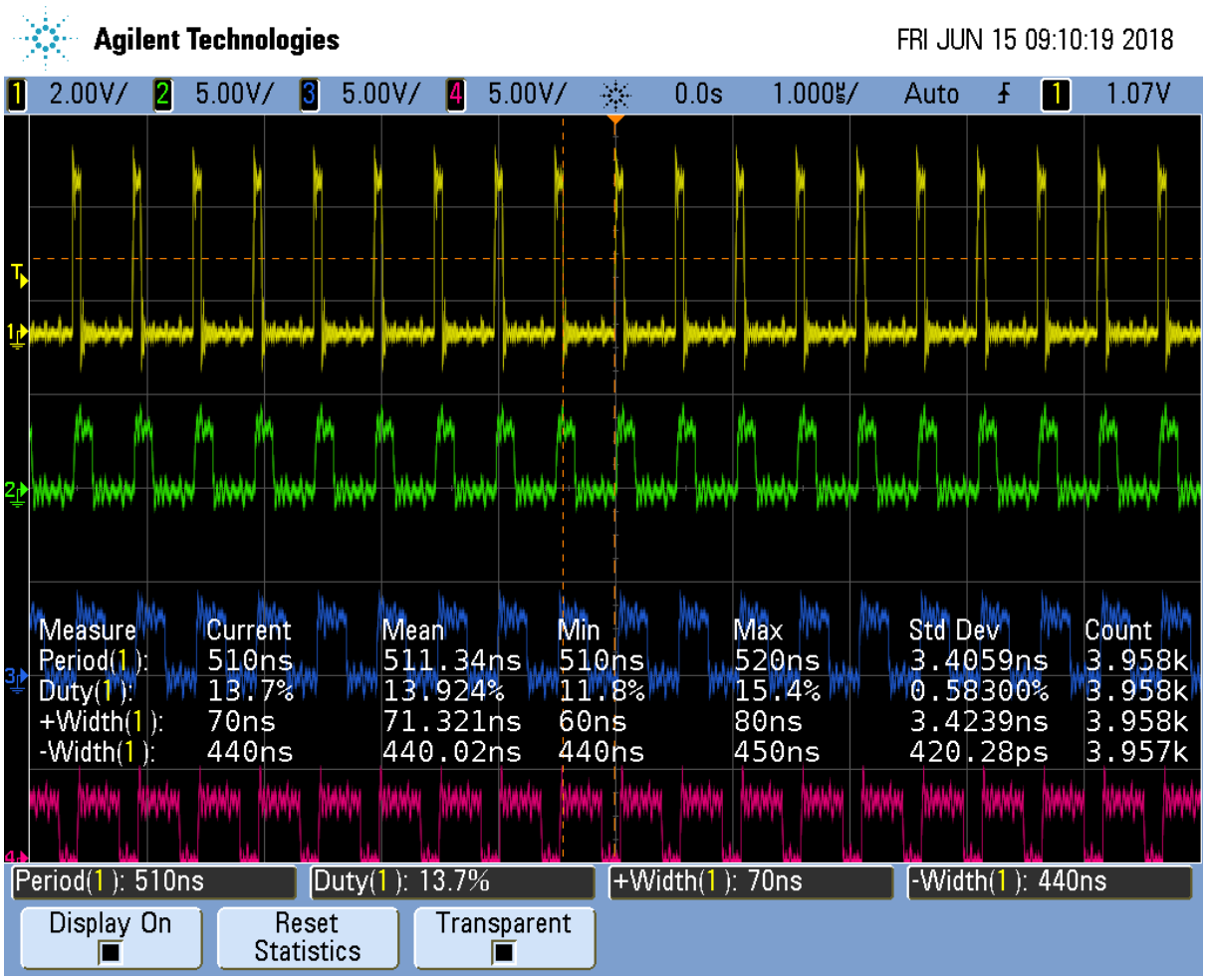


Fig. 15.152: pwm5.pru0.c Unrolled version of pwm4.pru0.c

It's running about 6 times faster than `pwm4.pru0.c`.

Table 15.19: pwm4.pru0.c vs. pwm5.pru0.c

Measure	pwm4.pru0.c time	pwm5.pru0.c time	Speedup	pwm5.pru0.c w/o UNROLL	Speedup
Period	3.06μs	510ns	6x	1.81μs	~1.7x
Width+	450ns	70ns	~6x	1.56μs	~.3x

Not a bad speed up for just a couple of simple changes.

Discussion Here's how it works. First look at line 39. You see `#pragma UNROLL(MAXCH)` which is a `pragma` that tells the compiler to unroll the loop that follows. We are unrolling it `MAXCH` times (four times in this example). Just removing the `pragma` causes the speedup compared to the `pwm4.pru0.c` case to drop from 6x to only 1.7x.

We also have our `for` loop inside the `while` loop that can be unrolled. Unfortunately `UNROLL()` doesn't work on it, therefore we have to do it by hand. We could take the loop and just copy it three times, but that would make it harder to maintain the code. Instead I converted the loop into a `#define` (lines 14-24) and invoked `update()` as needed (lines 48-51). This is not a function call. Whenever the preprocessor sees the `update()` it copies the code an then it's compiled.

This unrolling gets us an impressive 6x speedup.

Making All the Pulses Start at the Same Time

Problem I have a mutlichannel PWM working, but the pulses aren't synchronized, that is they don't all start at the same time.

Solution *pwm5.pru0 Zoomed In* is a zoomed in version of the previous figure. Notice the pulse in each channel starts about 15ns later than the channel above it.

The solution is to declare `Rtmp` (line 35) which holds the value for `___R30`.

Listing 15.92: pwm6.pru0.c Sync'ed Version of pwm5.pru0.c

```
1  // This code does MAXCH parallel PWM channels.
2  // All channels start at the same time. It's period is 510ns
3  #include <stdint.h>
4  #include <pru_cfg.h>
5  #include "resource_table_empty.h"
6
7  #define PRU0_DRAM                0x00000                         // Offset to␣
   ↪DRAM
8  // Skip the first 0x200 byte of DRAM since the Makefile allocates
9  // 0x100 for the STACK and 0x100 for the HEAP.
10 volatile unsigned int *pru0_dram = (unsigned int *) (PRU0_DRAM + 0x200);
11
12 #define MAXCH        4        // Maximum number of channels per PRU
13
14 #define update(ch) \
15                          if(onCount[ch]) {                          \
16                                  onCount[ch]--;                          \
17                                  Rtmp |= 0x1<<ch;                   \
18                          } else if(offCount[ch]) {          \
19                                  offCount[ch]--;                          \
20                                  Rtmp &= ~(0x1<<ch);          \
21                          } else {                                          \
22                                  onCount[ch] = pru0_dram[2*ch];          \
23                                  offCount[ch]= pru0_dram[2*ch+1];          \
24                          }
```

(continues on next page)

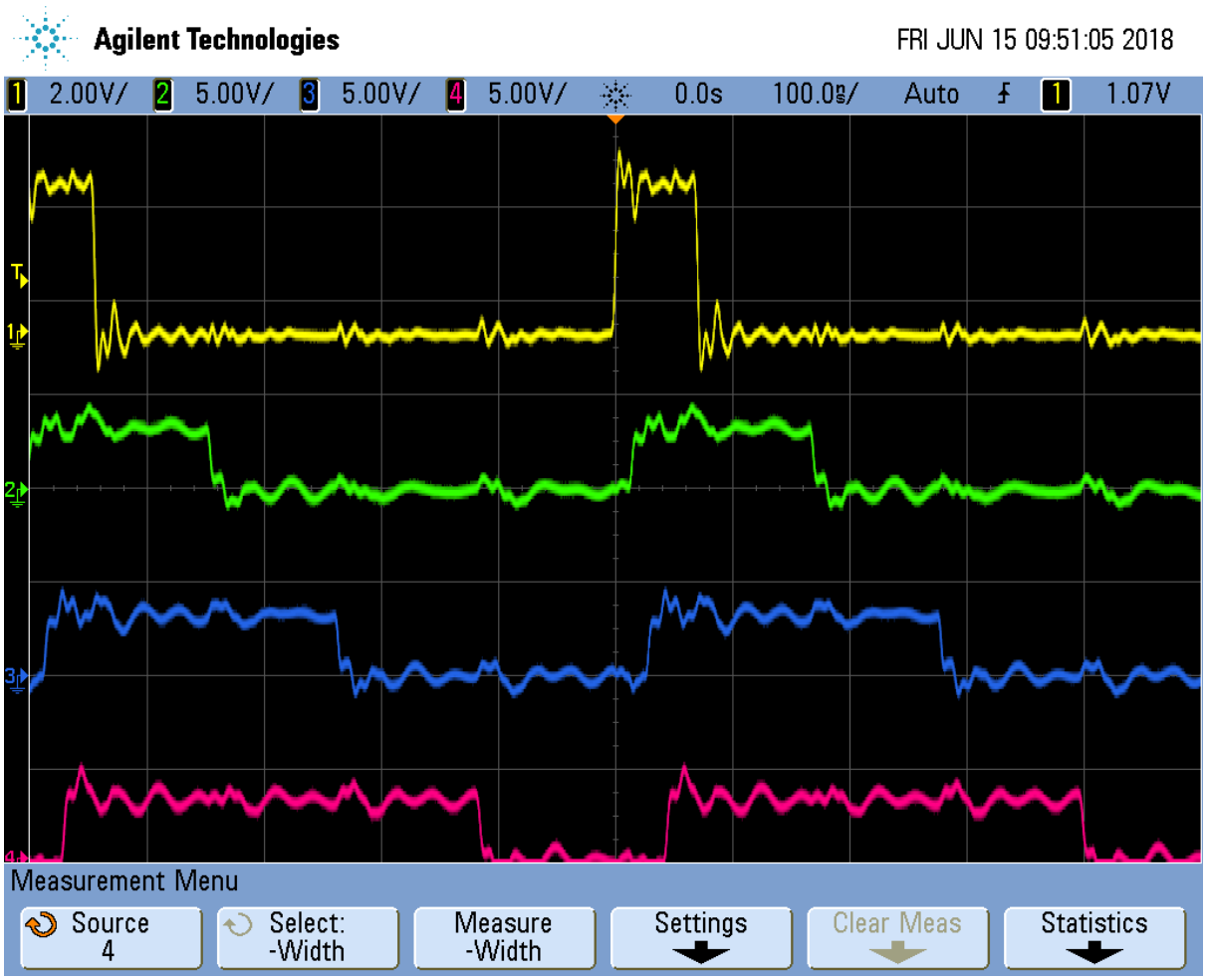


Fig. 15.153: pwm5.pru0 Zoomed In

(continued from previous page)

```
25
26 volatile register uint32_t __R30;
27 volatile register uint32_t __R31;
28
29 void main(void)
30 {
31         uint32_t ch;
32         uint32_t on[]  = {1, 2, 3, 4};
33         uint32_t off[] = {4, 3, 2, 1};
34         uint32_t onCount[MAXCH], offCount[MAXCH];
35         register uint32_t Rtmp;
36
37         /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
38         CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
39
40 #pragma UNROLL(MAXCH)
41         for(ch=0; ch<MAXCH; ch++) {
42                 pru0_dram[2*ch  ] = on[ch];                // Copy to DRAM0␣
   ↪so the ARM can change it
43                 pru0_dram[2*ch+1] = off[ch];        // Interleave the on and␣
   ↪off values
44                 onCount[ch] = on[ch];
45                 offCount[ch]= off[ch];
46         }
47         Rtmp = __R30;
48
49         while (1) {
50                 update(0)
51                 update(1)
52                 update(2)
53                 update(3)
54                 __R30 = Rtmp;
55         }
56 }
```

`pwm6.pru0.c` Sync'ed Version of `pwm5.pru0.c`

Each channel writes it's value to `Rtmp` (lines 17 and 20) and then after each channel has updated, `Rtmp` is copied to `__R30` (line 54).

Discussion The following figure shows the channel are sync'ed. Though the period is slightly longer than before.

Adding More Channels via PRU 1

Problem You need more output channels, or you need to shorten the period.

Solution PRU 0 can output up to eight output pins (see *Mapping bit positions to pin names*). The code presented so far can be easily extended to use the eight output pins.

But what if you need more channels? You can always use PRU1, it has 14 output pins.

Or, what if four channels is enough, but you need a shorter period. Everytime you add a channel, the overall period gets longer. Twice as many channels means twice as long a period. If you move half the channels to PRU 1, you will make the period half as long.

Here's the code (`pwm7.pru0.c`)

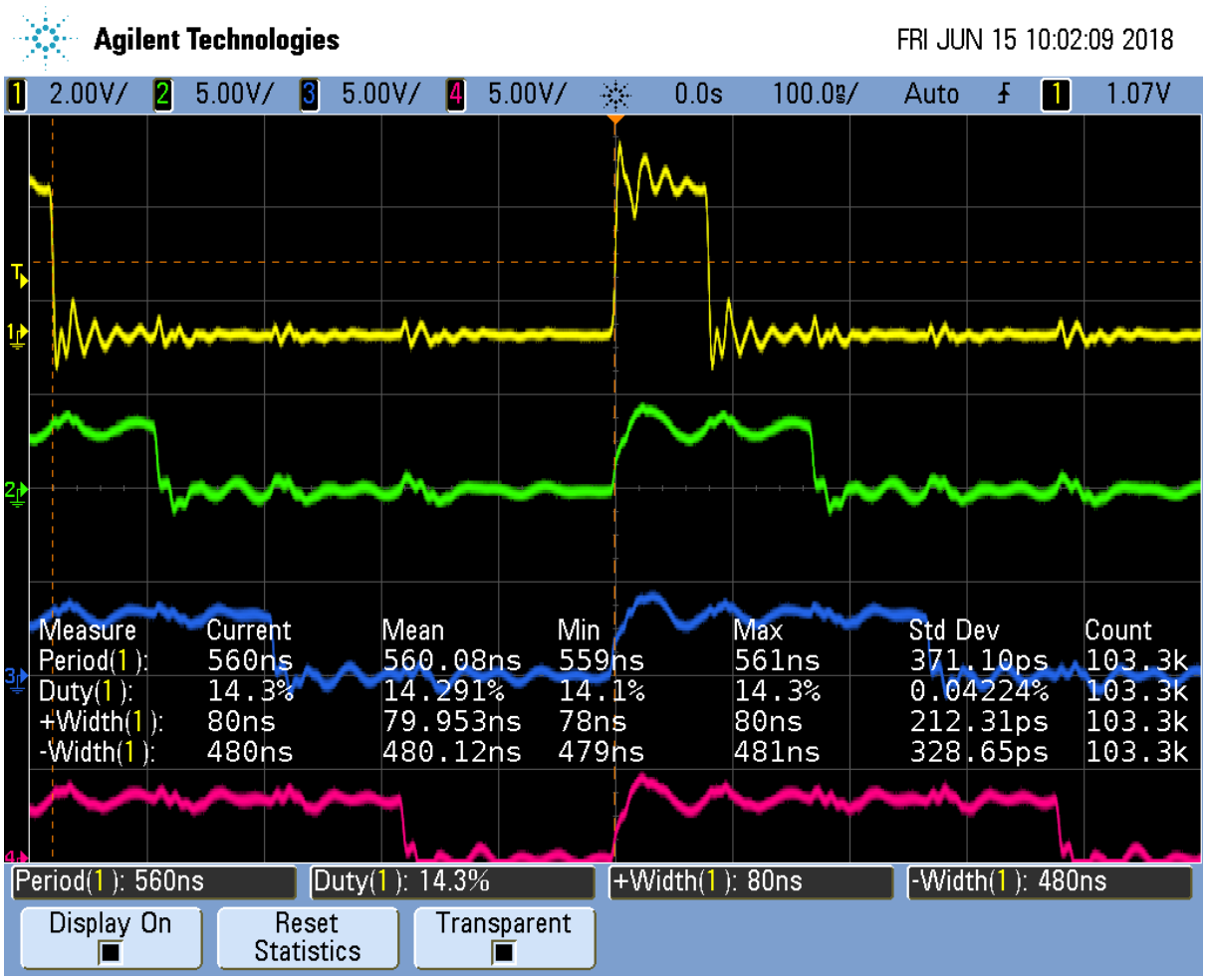


Fig. 15.154: pwm6.pru0 Synchronized Channels

Listing 15.93: pwm7.pru0.c Using Both PRUs

```
// This code does MAXCH parallel PWM channels on both PRU 0 and PRU 1
// All channels start at the same time. But the PRU 1 ch have a difference period
// It's period is 370ns
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"

#define PRUNUM 0

#define PRU0_DRAM               0x00000                         // Offset to DRAM
// Skip the first 0x200 byte of DRAM since the Makefile allocates
// 0x100 for the STACK and 0x100 for the HEAP.
volatile unsigned int *pru0_dram = (unsigned int *) (PRU0_DRAM + 0x200);

#define MAXCH        2        // Maximum number of channels per PRU

#define update(ch) \
                        if(onCount[ch]) {                           \
                                onCount[ch]--;                          \
                                Rtmp |= 0x1<<ch;                \
                        } else if(offCount[ch]) {       \
                                offCount[ch]--;                         \
                                Rtmp &= ~(0x1<<ch);         \
                        } else {                                        \
                                onCount[ch] = pru0_dram[2*ch];          \
                                offCount[ch]= pru0_dram[2*ch+1];        \
                        }

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t ch;
        uint32_t on[]  = {1, 2, 3, 4};
        uint32_t off[] = {4, 3, 2, 1};
        uint32_t onCount[MAXCH], offCount[MAXCH];
        register uint32_t Rtmp;

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

#pragma UNROLL(MAXCH)
        for(ch=0; ch<MAXCH; ch++) {
                pru0_dram[2*ch  ] = on [ch+PRUNUM*MAXCH];       // Copy to DRAM0 so the ARM can change it
                pru0_dram[2*ch+1] = off[ch+PRUNUM*MAXCH];       // Interleave the on and off values
                onCount[ch] = on [ch+PRUNUM*MAXCH];
                offCount[ch]= off[ch+PRUNUM*MAXCH];
        }
        Rtmp = __R30;

        while (1) {
                update(0)
                update(1)
                __R30 = Rtmp;
        }
```

(continues on next page)

(continued from previous page)

```
}
```

pwm7.pru0.c Using Both PRUs

Be sure to run `pwm7_setup.sh` to get the correct pins configured.

Listing 15.94: pwm7_setup.sh

```
#!/bin/bash
#
export TARGET=pwm7.pru0
echo TARGET=$TARGET

# Configure the PRU pins based on which Beagle is running
machine=$(awk '{print $NF}' /proc/device-tree/model)
echo -n $machine
if [ $machine = "Black" ]; then
    echo " Found"
    pins="P9_31 P9_29 P8_45 P8_46"
elif [ $machine = "Blue" ]; then
    echo " Found"
    pins=""
elif [ $machine = "PocketBeagle" ]; then
    echo " Found"
    pins="P1_36 P1_33"
else
    echo " Not Found"
    pins=""
fi

for pin in $pins
do
    echo $pin
    config-pin $pin pruout
    config-pin -q $pin
done
```

pw7_setup.sh

This makes sure the PRU 1 pins are properly configured.

Here we have a second `pwm7` file. `pwm7.pru1.c` is identical to `pwm7.pru0.c` except `PRUNUM` is set to 1, instead of 0.

Compile and run the two files with:

```
bone$ *make TARGET=pwm7.pru0; make TARGET=pwm7.pru1*
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=pwm7.pru0
-    Stopping PRU 0
-     copying firmware file /tmp/vsx-examples/pwm7.pru0.out to /lib/firmware/
↪am335x-pru0-fw
write_init_pins.sh
-    Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR = /sys/class/remoteproc/remoteproc1
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=pwm7.pru1
-    Stopping PRU 1
-     copying firmware file /tmp/vsx-examples/pwm7.pru1.out to /lib/firmware/
↪am335x-pru1-fw
```

(continues on next page)

(continued from previous page)

```
write_init_pins.sh
-    Starting PRU 1
MODEL   = TI_AM335x_BeagleBone_Black
PROC    = pru
PRUN    = 1
PRU_DIR = /sys/class/remoteproc/remoteproc2
```

This will first stop, compile and start PRU 0, then do the same for PRU 1.

Moving half of the channels to PRU1 dropped the period from 510ns to 370ns, so we gained a bit.

Discussion There weren't many changes to be made. Line 15 we set MAXCH to 2. Lines 44-48 is where the big change is.

```
pru0_dram[2*ch  ] = on [ch+PRUNUN*MAXCH];       // Copy to DRAM0 so the ARM␣
↪can change it
pru0_dram[2*ch+1] = off[ch+PRUNUN*MAXCH];       // Interleave the on and off␣
↪values
onCount[ch] = on [ch+PRUNUN*MAXCH];
offCount[ch]= off[ch+PRUNUN*MAXCH];
```

If we are compiling for PRU 0, `on[ch+PRUNUN*MAXCH]` becomes `on[ch+0*2]` which is `on[ch]` which is what we had before. But now if we are on PRU 1 it becomes `on[ch+1*2]` which is `on[ch+2]`. That means we are picking up the second half of the `on` and `off` arrays. The first half goes to PRU 0, the second to PRU 1. So the same code can be used for both PRUs, but we get slightly different behavior.

Running the code you will see the next figure.

What's going on there, the first channels look fine, but the PRU 1 channels are blurred. To see what's happening, let's stop the oscilloscope.

The stopped display shows that the four channels are doing what we wanted, except The PRU 0 channels have a period of 370ns while the PRU 1 channels at 330ns. It appears the compiler has optimied the two PRUs slightly differently.

Synchronizing Two PRUs

Problem I need to synchronize the two PRUs so they run together.

Solution Use the Interrupt Controller (INTC). It allows one PRU to signal the other. Page 225 of the AM335x TRM 225 has details of how it works. Here's the code for PRU 0, which at the end of the `while` loop signals PRU 1 to start(`pwm8.pru0.c`).

Listing 15.95: pwm8.pru0.c PRU 0 using INTC to send a signal to PRU 1

```
1  // This code does MAXCH parallel PWM channels on both PRU 0 and PRU 1
2  // All channels start at the same time.
3  // It's period is 430ns
4  #include <stdint.h>
5  #include <pru_cfg.h>
6  #include <pru_intc.h>
7  #include <pru_ctrl.h>
8  #include "resource_table_empty.h"
9
10 #define PRUNUM 0
11
12 #define PRU0_DRAM               0x00000                         // Offset to␣
↪DRAM
```

(continues on next page)

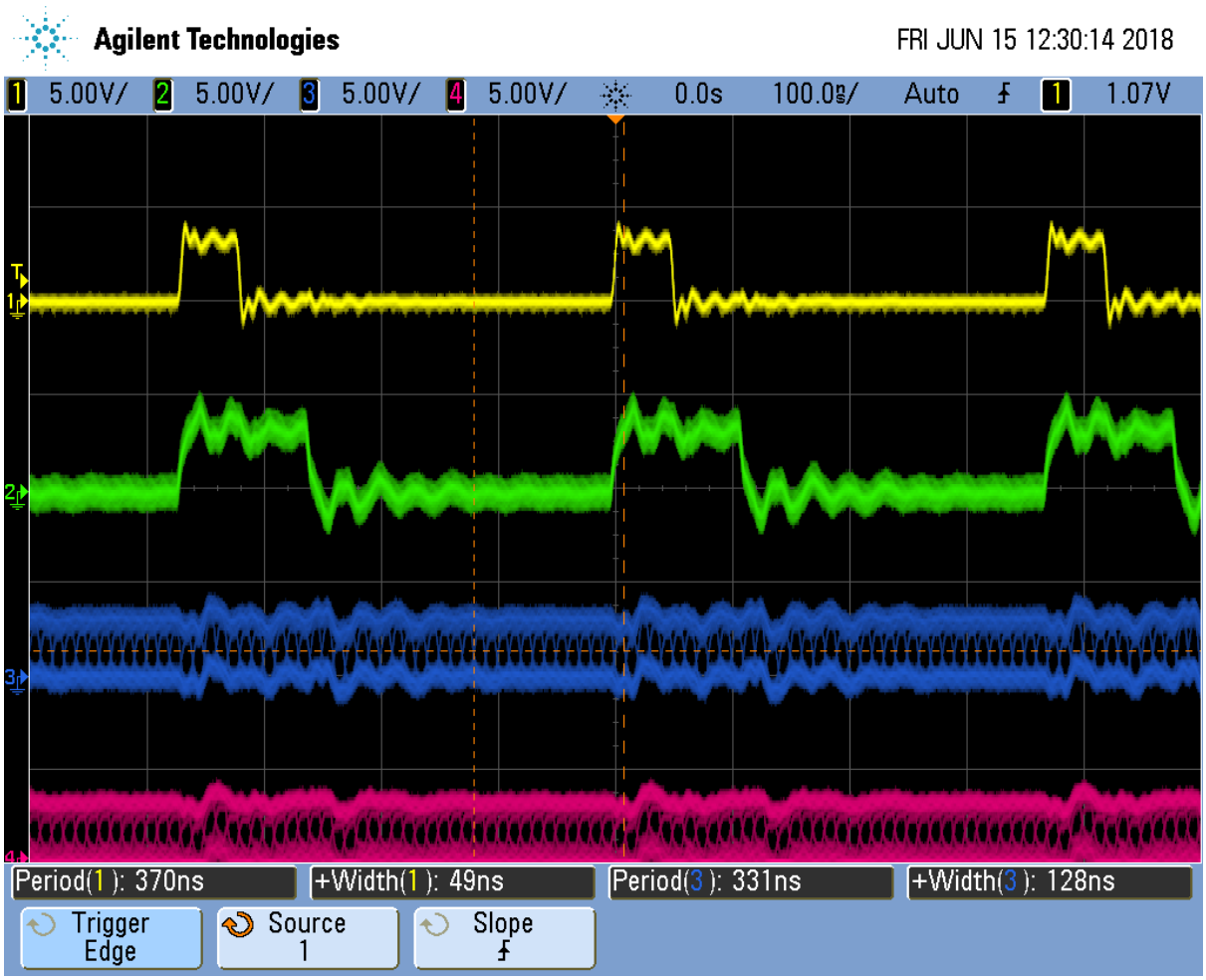


Fig. 15.155: pwm7.pru0 Two PRUs running

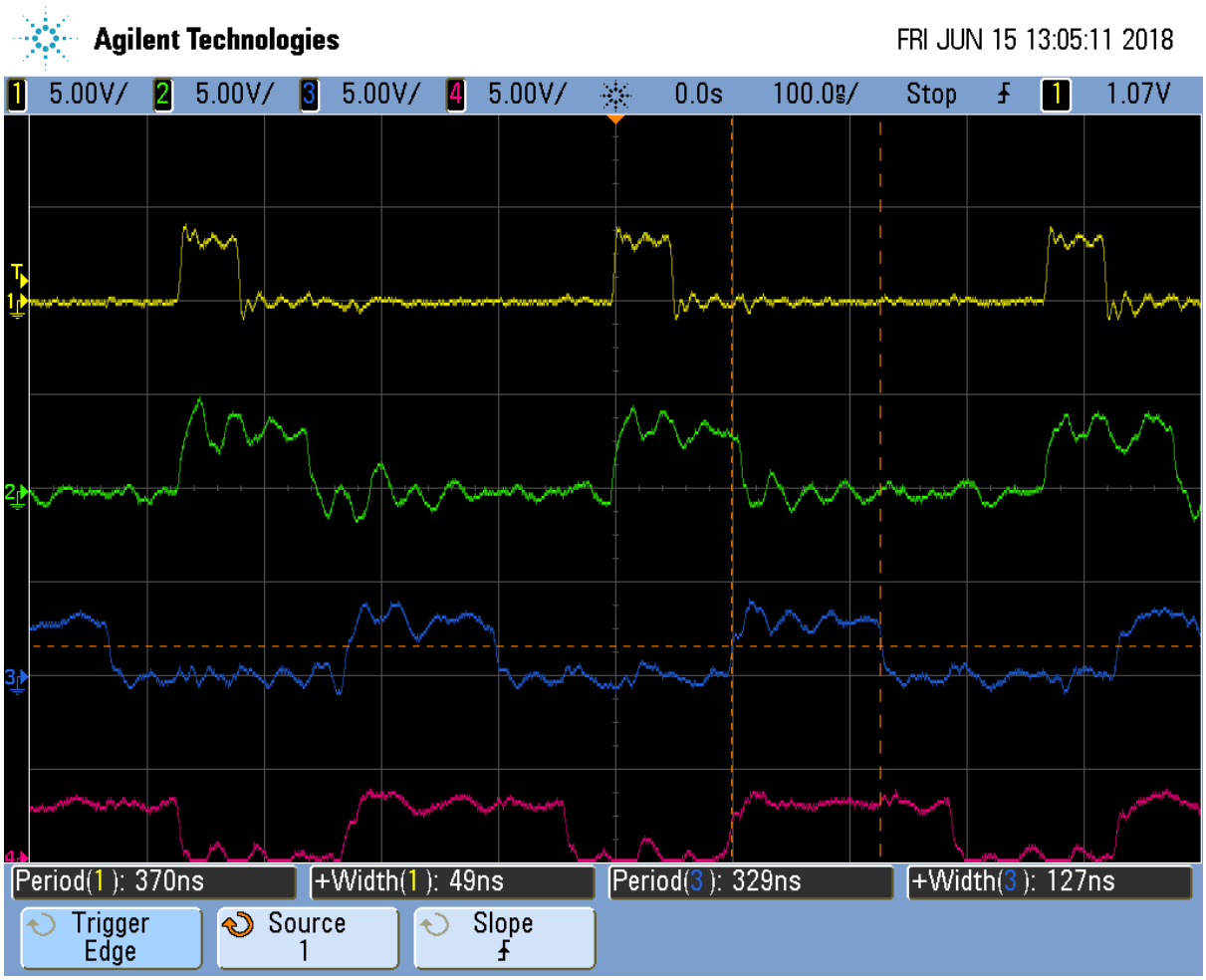


Fig. 15.156: pwm7.pru0 Two PRUs stopped

(continued from previous page)

```
// Skip the first 0x200 byte of DRAM since the Makefile allocates
// 0x100 for the STACK and 0x100 for the HEAP.
volatile unsigned int *pru0_dram = (unsigned int *) (PRU0_DRAM + 0x200);

#define MAXCH        2        // Maximum number of channels per PRU

#define update(ch) \
                        if(onCount[ch]) {                          \
                                onCount[ch]--;                          \
                                Rtmp |= 0x1<<ch;                  \
                        } else if(offCount[ch]) {         \
                                offCount[ch]--;                          \
                                Rtmp &= ~(0x1<<ch);         \
                        } else {                                          \
                                onCount[ch] = pru0_dram[2*ch];         \
                                offCount[ch]= pru0_dram[2*ch+1];         \
                        }

volatile register uint32_t __R30;
volatile register uint32_t __R31;

// Initialize interrupts so the PRUs can be syncronized.
// PRU1 is started first and then waits for PRU0
// PRU0 is then started and tells PRU1 when to start going
void configIntc(void) {
        __R31 = 0x00000000;                                          // Clear any pending PRU-generated events
        CT_INTC.CMR4_bit.CH_MAP_16 = 1;                // Map event 16 to channel 1
        CT_INTC.HMR0_bit.HINT_MAP_1 = 1;        // Map channel 1 to host 1
        CT_INTC.SICR = 16;                                          // Ensure event 16 is cleared
        CT_INTC.EISR = 16;                                          // Enable event 16
        CT_INTC.HIEISR |= (1 << 0);                          // Enable Host interrupt 1
        CT_INTC.GER = 1;                                          // Globally enable host interrupts
}

void main(void)
{
        uint32_t ch;
        uint32_t on[]  = {1, 2, 3, 4};
        uint32_t off[] = {4, 3, 2, 1};
        uint32_t onCount[MAXCH], offCount[MAXCH];
        register uint32_t Rtmp;

        CT_CFG.GPCFG0 = 0x0000;                                  // Configure GPI and GPO as Mode 0 (Direct Connect)
        configIntc();                                                  // Configure INTC

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

#pragma UNROLL(MAXCH)
        for(ch=0; ch<MAXCH; ch++) {
                pru0_dram[2*ch  ] = on [ch+PRUNUM*MAXCH];        // Copy to DRAM0 so the ARM can change it
                pru0_dram[2*ch+1] = off[ch+PRUNUM*MAXCH];        // Interleave the on and off values
```

(continues on next page)

(continued from previous page)

```
                onCount[ch] = on [ch+PRUNUM*MAXCH];
                offCount[ch]= off[ch+PRUNUM*MAXCH];
        }
        Rtmp = __R30;

        while (1) {
                __R30 = Rtmp;
                update(0)
                update(1)
#define PRU0_PRU1_EVT 16
                __R31 = (PRU0_PRU1_EVT-16) | (0x1<<5);        //Tell PRU 1 to start
                __delay_cycles(1);
        }
}
```

pwm8.pru0.c PRU 0 using INTC to send a signal to PRU 1

PRU 2's code waits for PRU 0 before going.

Listing 15.96: pwm8.pru1.c PRU 1 waiting for INTC from PRU 0

```
// This code does MAXCH parallel PWM channels on both PRU 0 and PRU 1
// All channels start at the same time.
// It's period is 430ns
#include <stdint.h>
#include <pru_cfg.h>
#include <pru_intc.h>
#include <pru_ctrl.h>
#include "resource_table_empty.h"

#define PRUNUM 1

#define PRU0_DRAM               0x00000                         // Offset to DRAM
// Skip the first 0x200 byte of DRAM since the Makefile allocates
// 0x100 for the STACK and 0x100 for the HEAP.
volatile unsigned int *pru0_dram = (unsigned int *) (PRU0_DRAM + 0x200);

#define MAXCH        2        // Maximum number of channels per PRU

#define update(ch) \
                        if(onCount[ch]) {                           \
                                onCount[ch]--;                          \
                                Rtmp |= 0x1<<ch;                    \
                        } else if(offCount[ch]) {           \
                                offCount[ch]--;                          \
                                Rtmp &= ~(0x1<<ch);         \
                        } else {                                        \
                                onCount[ch] = pru0_dram[2*ch];          \
                                offCount[ch]= pru0_dram[2*ch+1];          \
                        }

volatile register uint32_t __R30;
volatile register uint32_t __R31;

// Initialize interrupts so the PRUs can be syncronized.
// PRU1 is started first and then waits for PRU0
// PRU0 is then started and tells PRU1 when to start going

void main(void)
{
```

(continues on next page)

(continued from previous page)

```
        uint32_t ch;
        uint32_t on[]  = {1, 2, 3, 4};
        uint32_t off[] = {4, 3, 2, 1};
        uint32_t onCount[MAXCH], offCount[MAXCH];
        register uint32_t Rtmp;

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

#pragma UNROLL(MAXCH)
        for(ch=0; ch<MAXCH; ch++) {
                pru0_dram[2*ch  ] = on [ch+PRUNUM*MAXCH];         // Copy to␣
↪DRAM0 so the ARM can change it
                pru0_dram[2*ch+1] = off[ch+PRUNUM*MAXCH];         //␣
↪Interleave the on and off values
                onCount[ch] = on [ch+PRUNUM*MAXCH];
                offCount[ch]= off[ch+PRUNUM*MAXCH];
        }
        Rtmp = __R30;

        while (1) {
                while((__R31 & (0x1<<31))==0) {                 // Wait for␣
↪PRU 0
                }
                CT_INTC.SICR = 16;                                          //␣
↪Clear event 16
                __R30 = Rtmp;
                update(0)
                update(1)
        }
}
```

`pwm8.pru1.c` PRU 1 waiting for INTC from PRU 0

In `pwm8.pru0.c` PRU 1 waits for a signal from PRU 0, so be sure to start PRU 1 first.

```
bone$ *make TARGET=pwm8.pru0; make TARGET=pwm8.pru1*
```

Discussion The figure below shows the two PRUs are synchronized, though there is some extra overhead in the process so the period is longer.

This isn't much different from the previous examples.

Table 15.20: pwm8.pru0.c changes from pwm7.pru0.c

PRU	Line	Change
0	37-45	For PRU 0 these define `configInitc()` which initializes the interrupts. See page 226 of the AM335x TRM for a diagram explaining events, channels, hosts, etc.
0	55-56	Set a configuration register and call *configInitc*.
1	59-61	PRU 1 then waits for PRU 0 to signal it. Bit 31 of `__R31` corresponds to the Host-1 channel which `configInitc()` set up. We also clear event 16 so PRU 0 can set it again.
0	74-75	On PRU 0 this generates the interrupt to send to PRU 1. I found PRU 1 was slow to respond to the interrupt, so I put this code at the end of the loop to give time for the signal to get to PRU 1.

This ends the multipart pwm example.

Reading an Input at Regular Intervals

Problem You have an input pin that needs to be read at regular intervals.

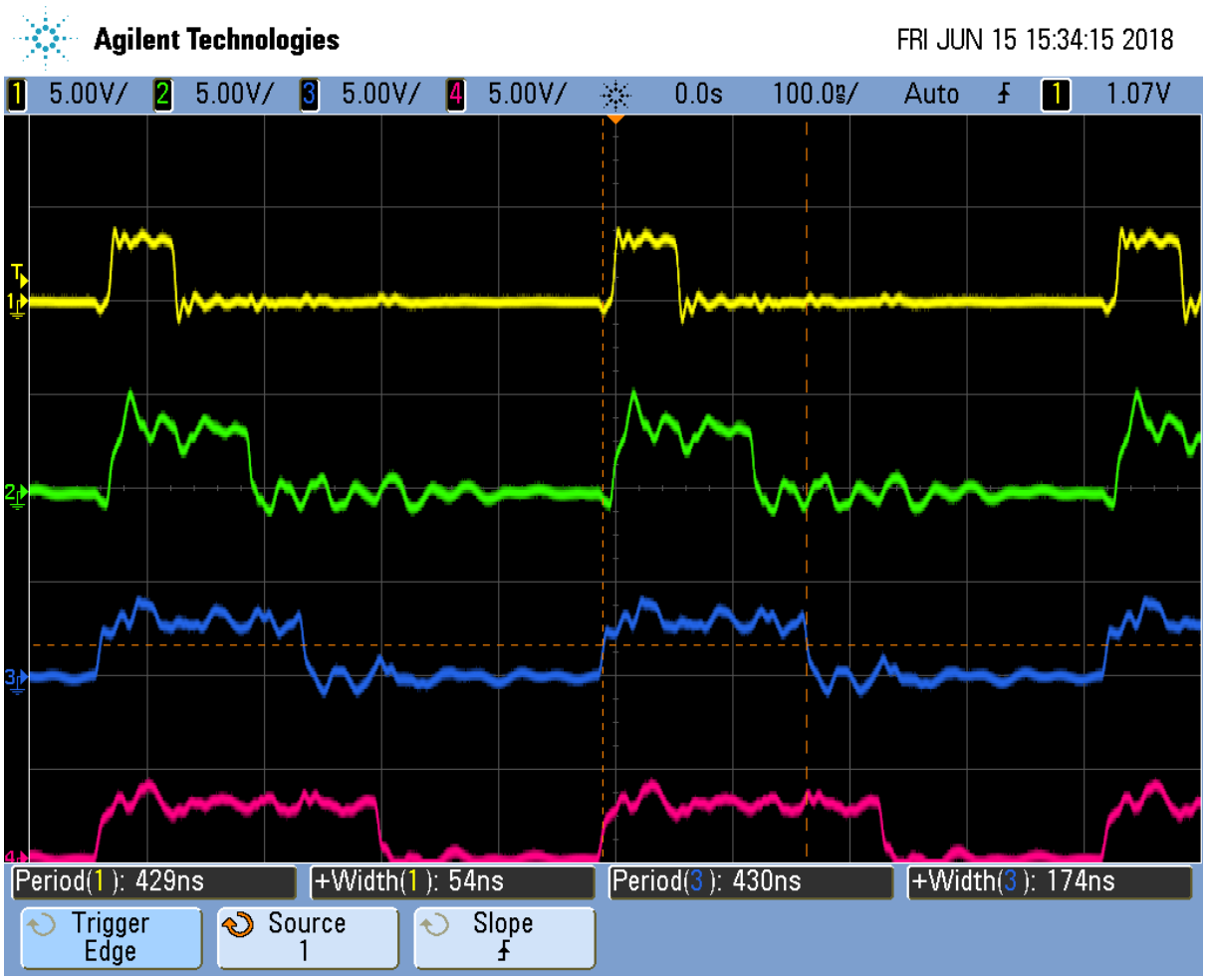


Fig. 15.157: pwm8.pru0 PRUs synced

Solution You can use the __R31 register to read an input pin. Let's use the following pins.

Table 15.21: Input/Output pins

Direction	Bit number	Black	AI (ICSS2)	Pocket
out	0	P9_31	P8_44	P1.36
in	7	P9_25	P8_36	P1.29

These values came from *Mapping bit positions to pin names*.

Configure the pins with `input_setup.sh`.

Listing 15.97: input_setup.sh

```
#!/bin/bash
#
export TARGET=input.pru0
echo TARGET=$TARGET

# Configure the PRU pins based on which Beagle is running
machine=$(awk '{print $NF}' /proc/device-tree/model)
echo -n $machine
if [ $machine = "Black" ]; then
    echo " Found"
    config-pin P9_31 pruout
    config-pin -q P9_31
    config-pin P9_25 pruin
    config-pin -q P9_25
elif [ $machine = "Blue" ]; then
    echo " Found"
    pins=""
elif [ $machine = "PocketBeagle" ]; then
    echo " Found"
    config-pin P1_36 pruout
    config-pin -q P1_36
    config-pin P1_29 pruin
    config-pin -q P1_29
else
    echo " Not Found"
    pins=""
fi
```

`input_setup.sh`

The following code reads the input pin and writes its value to the output pin.

Listing 15.98: input.pru0.c

```
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t led;
        uint32_t sw;

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        led = 0x1<<0;        // P9_31 or P1_36
```

(continues on next page)

(continued from previous page)

```
        sw   = 0x1<<7;          // P9_25 or P1_29

        while (1) {
                if((__R31&sw) == sw) {
                        __R30 |= led;                   // Turn on LED
                } else
                        __R30 &= ~led;                   // Turn off LED
        }
}
```

input.pru0.c

Discussion Just remember that ___R30 is for outputs and ___R31 is for inputs.

Analog Wave Generator

Problem I want to generate an analog output, but only have GPIO pins.

Solution The Beagle doesn't have a built-in analog to digital converter. You could get a USB Audio Dongle which are under $10. But here we'll take another approach.

Earlier we generated a PWM signal. Here we'll generate a PWM whose duty cycle changes with time. A small duty cycle for when the output signal is small and a large duty cycle for when it is large.

This example was inspired by A PRU Sin Wave Generator in chapter 13 of Exploring BeagleBone by Derek Molloy.

Here's the code.

Listing 15.99: sine.pru0.c

```
// Generate an analog waveform and use a filter to reconstruct it.
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include <math.h>

#define MAXT        100         // Maximum number of time samples
#define SAWTOOTH        // Pick which waveform

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t onCount;                   // Current count for 1 out
        uint32_t offCount;                   // count for 0 out
        uint32_t i;
        uint32_t waveform[MAXT]; // Waveform to be produced

        // Generate a periodic wave in an array of MAXT values
#ifdef SAWTOOTH
        for(i=0; i<MAXT; i++) {
                waveform[i] = i*100/MAXT;
        }
#endif
#ifdef TRIANGLE
        for(i=0; i<MAXT/2; i++) {
                waveform[i]         = 2*i*100/MAXT;
```

(continues on next page)

(continued from previous page)

```
29                  waveform[MAXT-i-1] = 2*i*100/MAXT;
30          }
31  #endif
32  #ifdef SINE
33          float gain = 50.0f;
34          float bias = 50.0f;
35          float freq = 2.0f * 3.14159f / MAXT;
36          for (i=0; i<MAXT; i++){
37                  waveform[i] = (uint32_t)(bias+gain*sin(i*freq));
38          }
39  #endif
40
41          /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
42          CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
43
44          while (1) {
45                  // Generate a PWM signal whose duty cycle matches
46                  // the amplitude of the signal.
47                  for(i=0; i<MAXT; i++) {
48                          onCount = waveform[i];
49                          offCount = 100 - onCount;
50                          while(onCount--) {
51                                  __R30 |= 0x1;                   // Set the GPIO
    ↪pin to 1
52                          }
53                          while(offCount--) {
54                                  __R30 &= ~(0x1);          // Clear the GPIO pin
55                          }
56                  }
57          }
58  }
```

`sine.pru0.c`

Set the `#define` at line 7 to the number of samples in one cycle of the waveform and set the `#define` at line 8 to which waveform and then run `make`.

Discussion The code has two parts. The first part (lines 21 to 39) generate the waveform to be output. The `#define``s let you select which waveform you want to generate. Since the output is a percent duty cycle, the values in ``waveform[]` must be between 0 and 100 inclusive. The waveform is only generated once, so this part of the code isn't time critical.

The second part (lines 44 to 54) uses the generated data to set the duty cycle of the PWM on a cycle-by-cycle basis. This part is time critical; the faster we can output the values, the higher the frequency of the output signal.

Suppose you want to generate a sawtooth waveform like the one shown in *Continuous Sawtooth Waveform*.

You need to sample the waveform and store one cycle. *Sampled Sawtooth Waveform* shows a sampled version of the sawtooth. You need to generate `MAXT` samples; here we show 20 samples, which may be enough. In the code `MAXT` is set to 100.

There's a lot going on here; let's take it line by line.

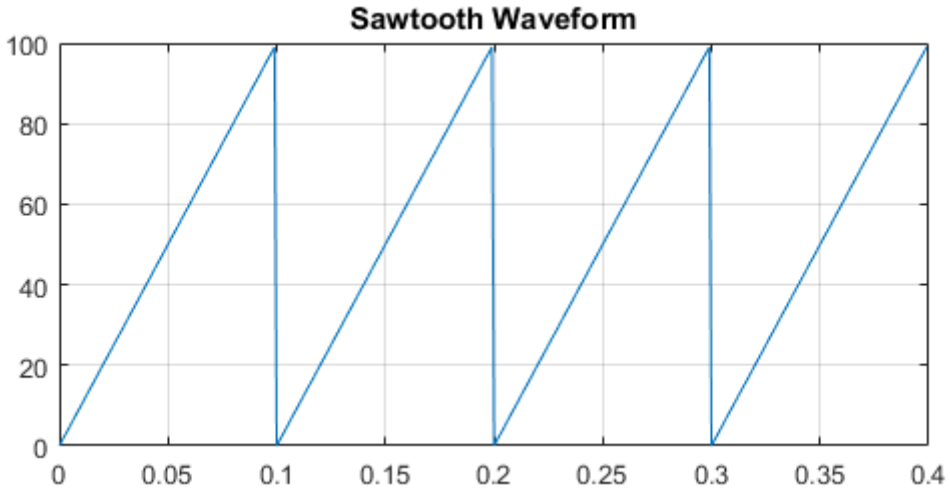


Fig. 15.158: Continuous Sawtooth Waveform

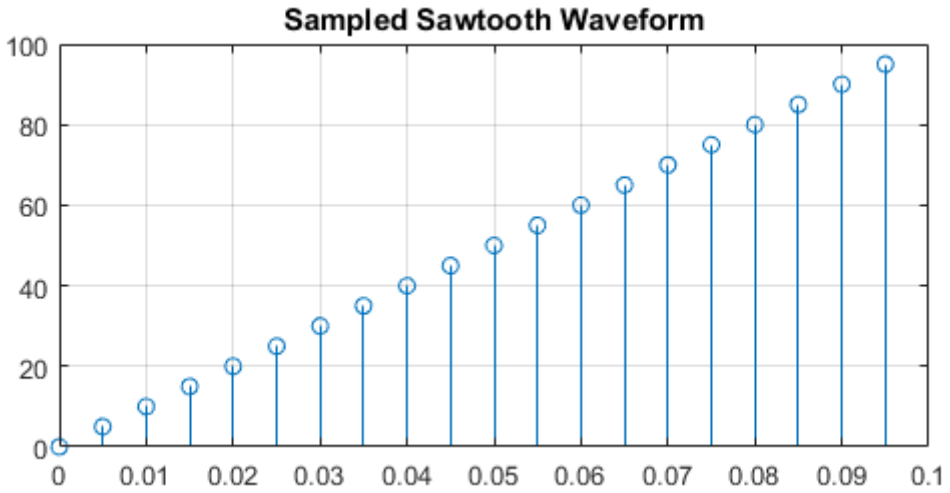


Fig. 15.159: Sampled Sawtooth Waveform

Table 15.22: Line-by-line of sine.pru0.c

Line	Explanation
2-5	Standard c-header includes
7	Number for samples in one cycle of the analog waveform
8	Which waveform to use. We've defined SAWTOOTH, TRIANGLE and SINE, but you can define your own too.
10-11	Declaring registers `pass:[__R30]` and `pass:[__R31]`.
15-16	`onCount` counts how many cycles the PWM should be 1 and `offCount` counts how many it should be off.
18	`waveform[]` stores the analog waveform being output.
21-24	`SAWTOOTH` is the simplest of the waveforms. Each sample is the duty cycle at that time and must therefore be between 0 and 100.
26-31	`TRIANGLE` is also a simple waveform.
32-39	`SINE` generates a sine wave and also introduces floating point. Yes, you can use floating point, but the PRUs don't have floating point hardware, rather, it's all done in software. This mean using floating point will make your code much bigger and slower. Slower doesn't matter in this part, and bigger isn't bigger than our instruction memory, so we're OK.
47	Here the `for` loop looks up each value of the generated waveform.
48,49	`onCount` is the number of cycles to be at 1 and `offCount` is the number of cycles to be 0. The two add to 100, one full cycle.
50-52	Stay on for `onCount` cycles.
53-55	Now turn off for `offCount` cycles, then loop back and look up the next cycle count.

Unfiltered Sawtooth Waveform shows the output of the code.

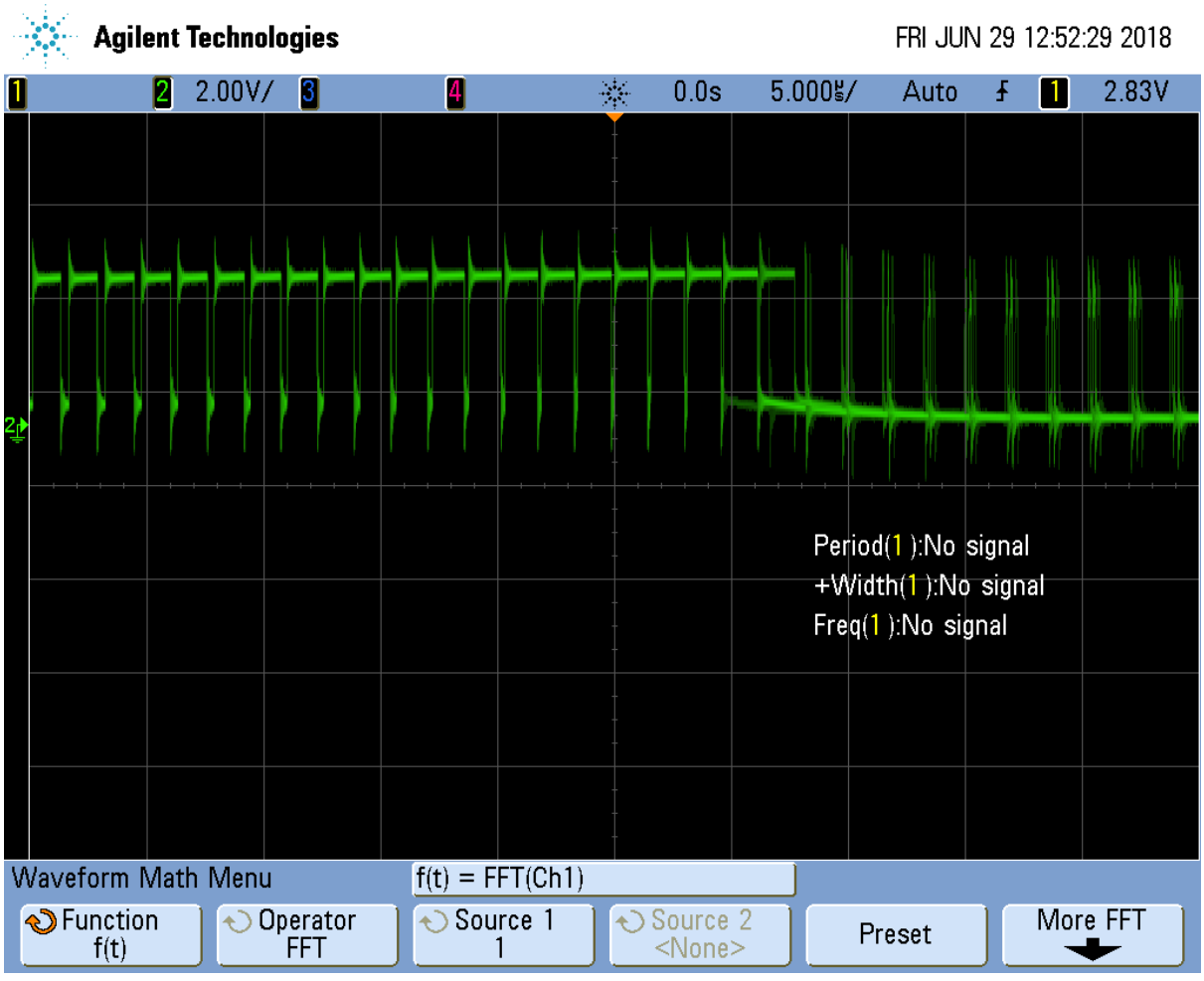


Fig. 15.160: Unfiltered Sawtooth Waveform

It doesn't look like a sawtooth; but if you look at the left side you will see each cycle has a longer and longer

on time. The duty cycle is increasing. Once it's almost 100% duty cycle, it switches to a very small duty cycle. Therefore it's output what we programmed, but what we want is the average of the signal. The left hand side has a large (and increasing) average which would be for top of the sawtooth. The right hand side has a small average, which is what you want for the start of the sawtooth.

A simple low-pass filter, built with one resistor and one capacitor will do it. *Low-Pass Filter Wiring Diagram* shows how to wire it up.

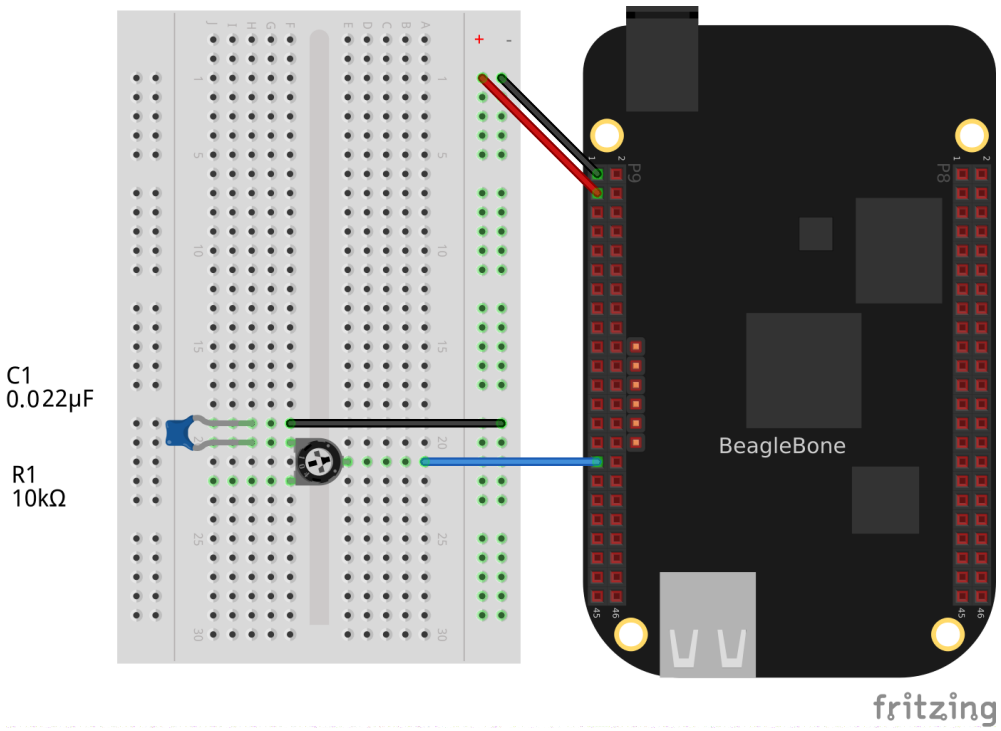


Fig. 15.161: Low-Pass Filter Wiring Diagram

Note: I used a 10K variable resistor and a 0.022uF capacitor. Probe the circuit between the resistor and the capacitor and adjust the resistor until you get a good looking waveform.

Reconstructed Sawtooth Waveform shows the results for filtered the SAWTOOTH.

Now that looks more like a sawtooth wave. The top plot is the time-domain plot of the output of the low-pass filter. The bottom plot is the FFT of the top plot, therefore it's the frequency domain. We are getting a sawtooth with a frequency of about 6.1KHz. You can see the fundamental frequency on the bottom plot along with several harmonics.

The top looks like a sawtooth wave, but there is a high freqnecy superimposed on it. We are only using a simple first-order filter. You could lower the cutoff freqnecy by adjusting the resistor. You'll see something like *Reconstructed Sawtooth Waveform with Lower Cutoff Frequency*.

The high frequencies have been reduced, but the corner of the waveform has been rounded. You can also adjust the cutoff to a higher frequency and you'll get a sharper corner, but you'll also get more high frequencies. See *Reconstructed Sawtooth Waveform with Higher Cutoff Frequency*

Adjust to taste, though the real solution is to build a higher order filter. Search for _second order **filter** and you'll find some nice circuits.

You can adjust the frequency of the signal by adjusting `MAXT`. A smaller `MAXT` will give a higher frequency. I've gotten good results with `MAXT` as small as 20.

You can also get a triangle waveform by setting the `#define`. *Reconstructed Triangle Waveform* shows the output signal.

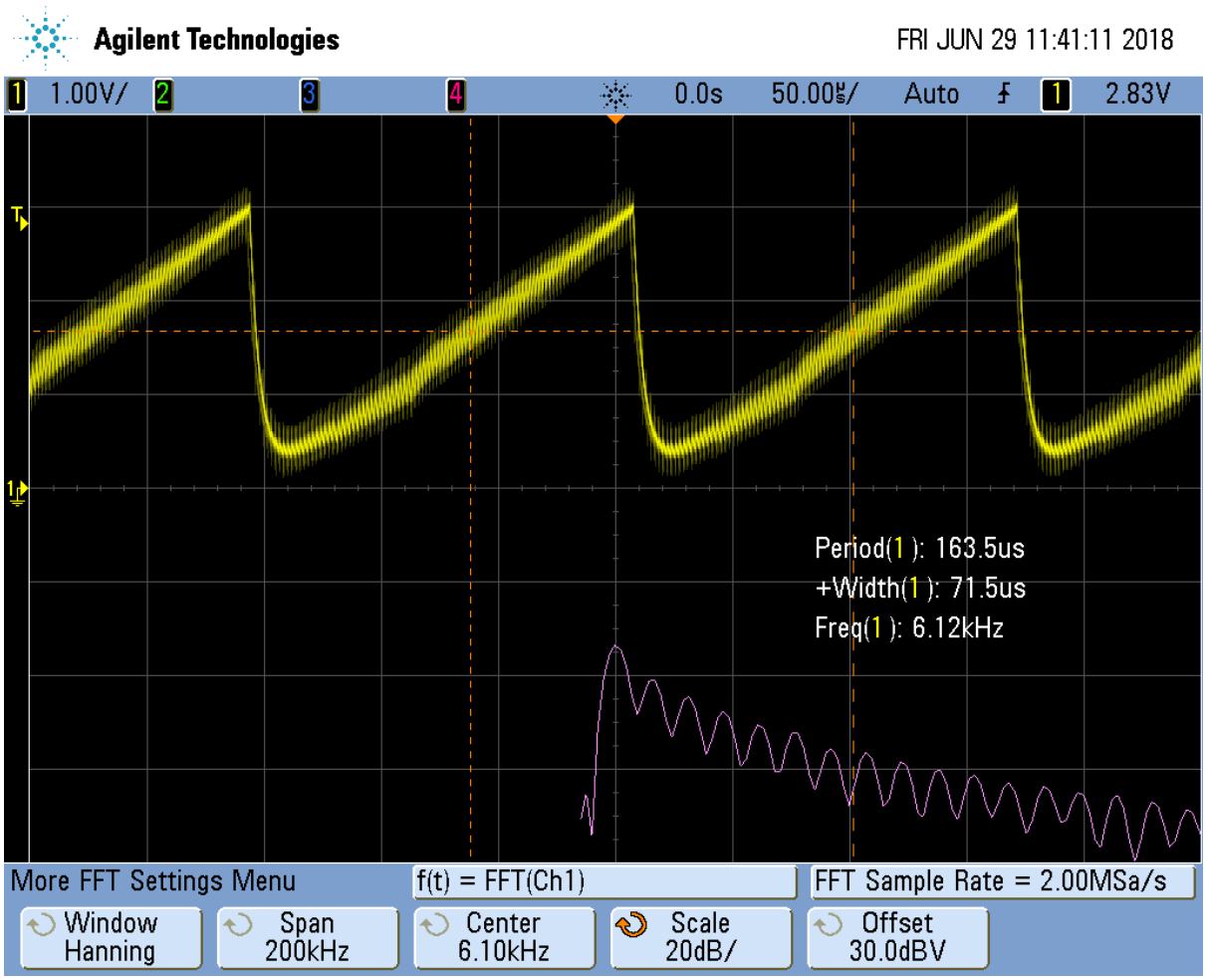


Fig. 15.162: Reconstructed Sawtooth Waveform

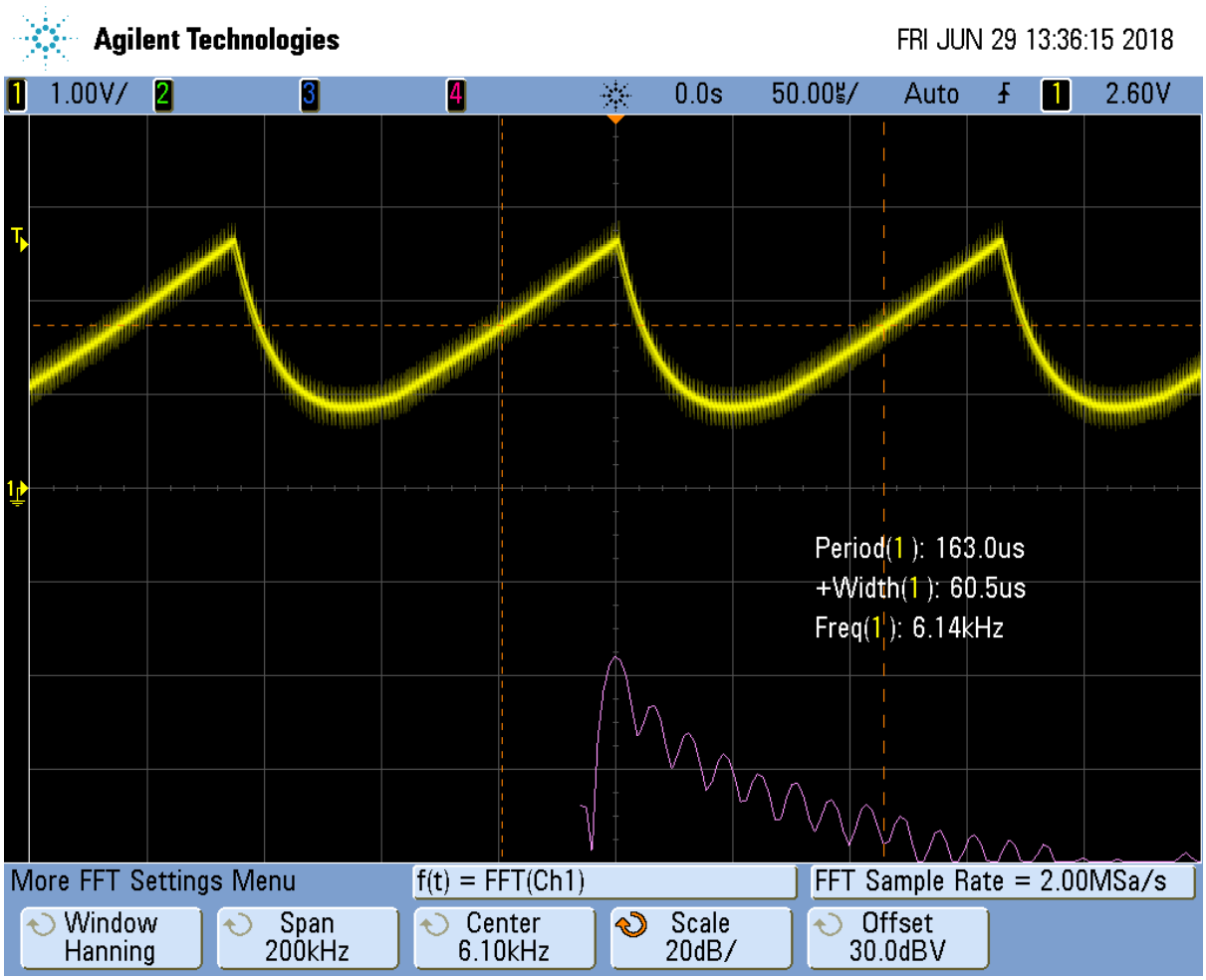


Fig. 15.163: Reconstructed Sawtooth Waveform with Lower Cutoff Frequency



Fig. 15.164: Reconstructed Sawtooth Waveform with Higher Cutoff Frequency

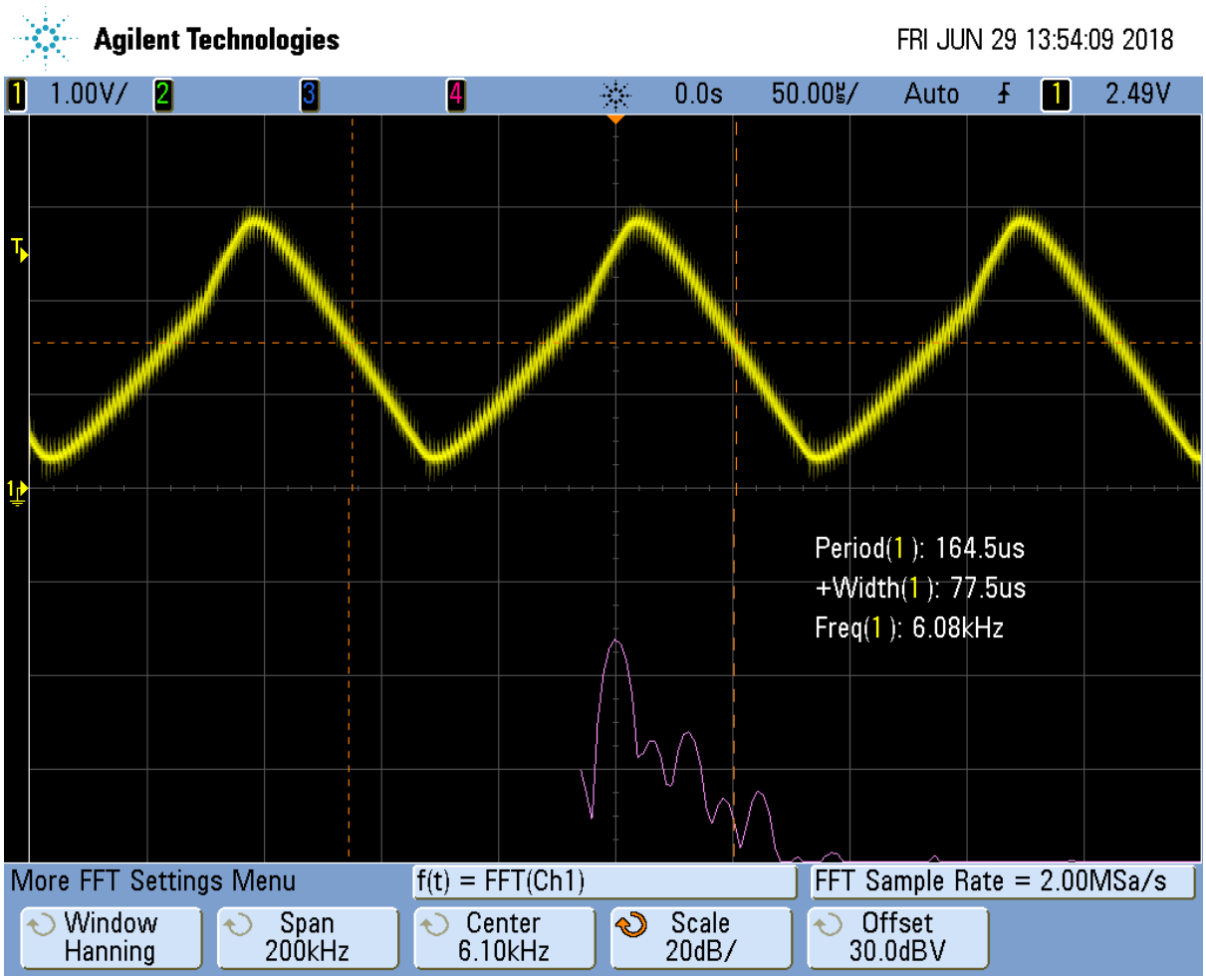


Fig. 15.165: Reconstructed Triangle Waveform

And also the sine wave as shown in *Reconstructed Sinusoid Waveform*.

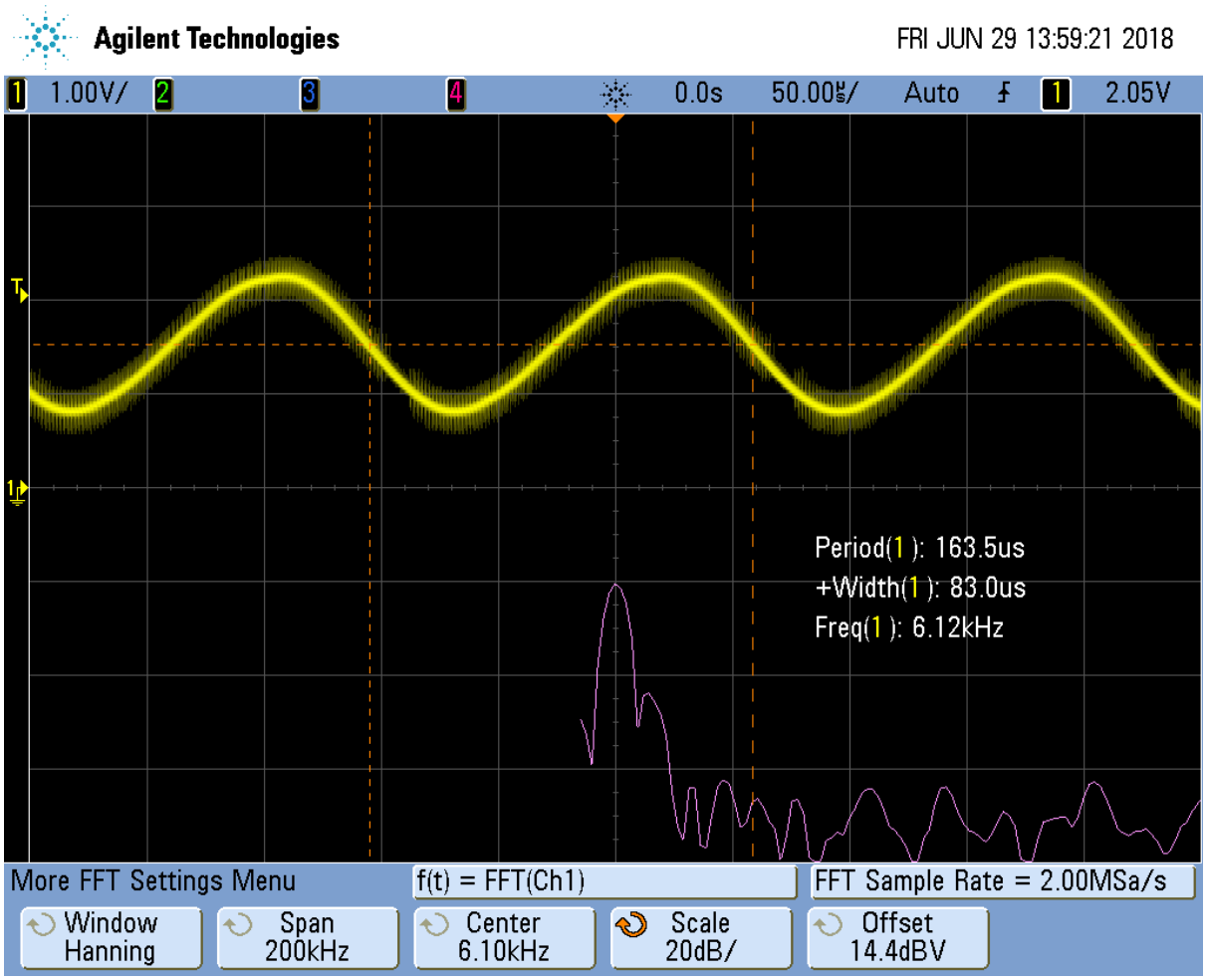


Fig. 15.166: Reconstructed Sinusoid Waveform

Notice on the bottom plot the harmonics are much more suppressed.

Generating the sine waveform uses **floats**. This requires much more code. You can look in */tmp/vsx-examples/sine.pru0.map* to see how much memory is being used. */tmp/vsx-examples/sine.pru0.map for Sine Wave* shows the first few lines for the sine wave.

Listing 15.100: /tmp/vsx-examples/sine.pru0.map for Sine Wave

```
1  ********************************************************************************
2  PRU Linker Unix v2.1.5
3  ********************************************************************************
4  >> Linked Fri Jun 29 13:58:08 2018
5
6  OUTPUT FILE NAME:   </tmp/pru0-gen/sine1.out>
7  ENTRY POINT SYMBOL: "_c_int00_noinit_noargs_noexit"  address: 00000000
8
9
10 MEMORY CONFIGURATION
11
12          name            origin    length      used     unused   attr    fill
13 ----------------------  --------  ---------  --------  --------  ----  ------
   ↪--
14 PAGE 0:
15   PRU_IMEM              00000000   00002000  000018c0  00000740  RWIX
16
```

(continues on next page)

(continued from previous page)

```
PAGE 1:
  PRU_DMEM_0_1          00000000   00002000  00000154  00001eac  RWIX
  PRU_DMEM_1_0          00002000   00002000  00000000  00002000  RWIX

PAGE 2:
  PRU_SHAREDMEM         00010000   00003000  00000000  00003000  RWIX
  PRU_INTC              00020000   00001504  00000000  00001504  RWIX
  PRU_CFG               00026000   00000044  00000044  00000000  RWIX
  PRU_UART              00028000   00000038  00000000  00000038  RWIX
  PRU_IEP               0002e000   0000031c  00000000  0000031c  RWIX
  PRU_ECAP              00030000   00000060  00000000  00000060  RWIX
  RSVD27                00032000   00000100  00000000  00000100  RWIX
  RSVD21                00032400   00000100  00000000  00000100  RWIX
  L3OCMC                40000000   00010000  00000000  00010000  RWIX
  MCASP0_DMA            46000000   00000100  00000000  00000100  RWIX
  UART1                 48022000   00000088  00000000  00000088  RWIX
  UART2                 48024000   00000088  00000000  00000088  RWIX
  I2C1                  4802a000   000000d8  00000000  000000d8  RWIX
  MCSPI0                48030000   000001a4  00000000  000001a4  RWIX
  DMTIMER2              48040000   0000005c  00000000  0000005c  RWIX
  MMCHS0                48060000   00000300  00000000  00000300  RWIX
  MBX0                  480c8000   00000140  00000000  00000140  RWIX
  SPINLOCK              480ca000   00000880  00000000  00000880  RWIX
  I2C2                  4819c000   000000d8  00000000  000000d8  RWIX
  MCSPI1                481a0000   000001a4  00000000  000001a4  RWIX
  DCAN0                 481cc000   000001e8  00000000  000001e8  RWIX
  DCAN1                 481d0000   000001e8  00000000  000001e8  RWIX
  PWMSS0                48300000   000002c4  00000000  000002c4  RWIX
  PWMSS1                48302000   000002c4  00000000  000002c4  RWIX
  PWMSS2                48304000   000002c4  00000000  000002c4  RWIX
  RSVD13                48310000   00000100  00000000  00000100  RWIX
  RSVD10                48318000   00000100  00000000  00000100  RWIX
  TPCC                  49000000   00001098  00000000  00001098  RWIX
  GEMAC                 4a100000   0000128c  00000000  0000128c  RWIX
  DDR                   80000000   00000100  00000000  00000100  RWIX


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.text:_c_int00*
*          0    00000000    00000014
                  00000000    00000014     rtspruv3_le.lib : boot_special.
↪obj (.text:_c_int00_noinit_noargs_noexit)

.text      0    00000014    000018ac
                  00000014    00000374     rtspruv3_le.lib : sin.obj (.
↪text:sin)
                  00000388    00000314                     : frcmpyd.obj (.
↪text:__TI_frcmpyd)
                  0000069c    00000258                     : frcaddd.obj (.
↪text:__TI_frcaddd)
                  000008f4    00000254                     : mpyd.obj (.
↪text:__pruabi_mpyd)
                  00000b48    00000248                     : addd.obj (.
↪text:__pruabi_addd)
                  00000d90    000001c8                     : mpyf.obj (.
↪text:__pruabi_mpyf)
                  00000f58    00000100                     : modf.obj (.
↪text:modf)
```

(continues on next page)

(continued from previous page)

```
                  00001058    000000b4                     : gtd.obj (.text:_
↪_pruabi_gtd)
                  0000110c    000000b0                     : ged.obj (.text:_
↪_pruabi_ged)
                  000011bc    000000b0                     : ltd.obj (.text:_
↪_pruabi_ltd)
                  0000126c    000000b0     sine1.obj (.text:main)
                  0000131c    000000a8     rtspruv3_le.lib : frcmpyf.obj (.
↪text:__TI_frcmpyf)
                  000013c4    000000a0                     : fixdu.obj (.
↪text:__pruabi_fixdu)
                  00001464    0000009c                     : round.obj (.
↪text:__pruabi_nround)
                  00001500    00000090                     : eqld.obj (.
↪text:__pruabi_eqd)
                  00001590    0000008c                     : renormd.obj (.
↪text:__TI_renormd)
                  0000161c    0000008c                     : fixdi.obj (.
↪text:__pruabi_fixdi)
                  000016a8    00000084                     : fltid.obj (.
↪text:__pruabi_fltid)
                  0000172c    00000078                     : cvtfd.obj (.
↪text:__pruabi_cvtfd)
                  000017a4    00000050                     : fltuf.obj (.
↪text:__pruabi_fltuf)
                  000017f4    0000002c                     : asri.obj (.
↪text:__pruabi_asri)
                  00001820    0000002c                     : subd.obj (.
↪text:__pruabi_subd)
                  0000184c    00000024                     : mpyi.obj (.
↪text:__pruabi_mpyi)
                  00001870    00000020                     : negd.obj (.
↪text:__pruabi_negd)
                  00001890    00000020                     : trunc.obj (.
↪text:__pruabi_trunc)
                  000018b0    00000008                     : exit.obj (.
↪text:abort)
                  000018b8    00000008                     : exit.obj (.
↪text:loader_exit)

.stack      1    00000000    00000100     UNINITIALIZED
                  00000000    00000004     rtspruv3_le.lib : boot.obj (.
↪stack)
                  00000004    000000fc     --HOLE--

.cinit      1    00000000    00000000     UNINITIALIZED

.fardata    1    00000100    00000040
                  00000100    00000040     rtspruv3_le.lib : sin.obj (.
↪fardata:R$1)

.resource_table
*           1    00000140    00000014
                  00000140    00000014     sine1.obj (.resource_table:retain)

.creg.PRU_CFG.noload.near
*           2    00026000    00000044     NOLOAD SECTION
                  00026000    00000044     sine1.obj (.creg.PRU_CFG.noload.
↪near)

.creg.PRU_CFG.near
```

(continues on next page)

(continued from previous page)

```
*          2    00026044    00000000     UNINITIALIZED

.creg.PRU_CFG.noload.far
*          2    00026044    00000000     NOLOAD SECTION

.creg.PRU_CFG.far
*          2    00026044    00000000     UNINITIALIZED


SEGMENT ATTRIBUTES

    id tag      seg value
    -- ---      --- -----
     0 PHA_PAGE 1   1
     1 PHA_PAGE 2   1


GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name

page  address   name
----  -------   ----
0     000018b8  C$$EXIT
2     00026000  CT_CFG
abs   481cc000  __PRU_CREG_BASE_DCAN0
abs   481d0000  __PRU_CREG_BASE_DCAN1
abs   80000000  __PRU_CREG_BASE_DDR
abs   48040000  __PRU_CREG_BASE_DMTIMER2
abs   4a100000  __PRU_CREG_BASE_GEMAC
abs   4802a000  __PRU_CREG_BASE_I2C1
abs   4819c000  __PRU_CREG_BASE_I2C2
abs   40000000  __PRU_CREG_BASE_L3OCMC
abs   480c8000  __PRU_CREG_BASE_MBX0
abs   46000000  __PRU_CREG_BASE_MCASP0_DMA
abs   48030000  __PRU_CREG_BASE_MCSPI0
abs   481a0000  __PRU_CREG_BASE_MCSPI1
abs   48060000  __PRU_CREG_BASE_MMCHS0
abs   00026000  __PRU_CREG_BASE_PRU_CFG
abs   00000000  __PRU_CREG_BASE_PRU_DMEM_0_1
abs   00002000  __PRU_CREG_BASE_PRU_DMEM_1_0
abs   00030000  __PRU_CREG_BASE_PRU_ECAP
abs   0002e000  __PRU_CREG_BASE_PRU_IEP
abs   00020000  __PRU_CREG_BASE_PRU_INTC
abs   00010000  __PRU_CREG_BASE_PRU_SHAREDMEM
abs   00028000  __PRU_CREG_BASE_PRU_UART
abs   48300000  __PRU_CREG_BASE_PWMSS0
abs   48302000  __PRU_CREG_BASE_PWMSS1
abs   48304000  __PRU_CREG_BASE_PWMSS2
abs   48318000  __PRU_CREG_BASE_RSVD10
abs   48310000  __PRU_CREG_BASE_RSVD13
abs   00032400  __PRU_CREG_BASE_RSVD21
abs   00032000  __PRU_CREG_BASE_RSVD27
abs   480ca000  __PRU_CREG_BASE_SPINLOCK
abs   49000000  __PRU_CREG_BASE_TPCC
abs   48022000  __PRU_CREG_BASE_UART1
abs   48024000  __PRU_CREG_BASE_UART2
abs   0000000e  __PRU_CREG_DCAN0
abs   0000000f  __PRU_CREG_DCAN1
abs   0000001f  __PRU_CREG_DDR
abs   00000001  __PRU_CREG_DMTIMER2
abs   00000009  __PRU_CREG_GEMAC
abs   00000002  __PRU_CREG_I2C1
```

(continues on next page)

(continued from previous page)

```
abs    00000011  __PRU_CREG_I2C2
abs    0000001e  __PRU_CREG_L3OCMC
abs    00000016  __PRU_CREG_MBX0
abs    00000008  __PRU_CREG_MCASP0_DMA
abs    00000006  __PRU_CREG_MCSPI0
abs    00000010  __PRU_CREG_MCSPI1
abs    00000005  __PRU_CREG_MMCHS0
abs    00000004  __PRU_CREG_PRU_CFG
abs    00000018  __PRU_CREG_PRU_DMEM_0_1
abs    00000019  __PRU_CREG_PRU_DMEM_1_0
abs    00000003  __PRU_CREG_PRU_ECAP
abs    0000001a  __PRU_CREG_PRU_IEP
abs    00000000  __PRU_CREG_PRU_INTC
abs    0000001c  __PRU_CREG_PRU_SHAREDMEM
abs    00000007  __PRU_CREG_PRU_UART
abs    00000012  __PRU_CREG_PWMSS0
abs    00000013  __PRU_CREG_PWMSS1
abs    00000014  __PRU_CREG_PWMSS2
abs    0000000a  __PRU_CREG_RSVD10
abs    0000000d  __PRU_CREG_RSVD13
abs    00000015  __PRU_CREG_RSVD21
abs    0000001b  __PRU_CREG_RSVD27
abs    00000017  __PRU_CREG_SPINLOCK
abs    0000001d  __PRU_CREG_TPCC
abs    0000000b  __PRU_CREG_UART1
abs    0000000c  __PRU_CREG_UART2
1      00000100  __TI_STACK_END
abs    00000100  __TI_STACK_SIZE
0      0000069c  __TI_frcaddd
0      00000388  __TI_frcmpyd
0      0000131c  __TI_frcmpyf
0      00001590  __TI_renormd
abs    ffffffff  __binit__
abs    ffffffff  __c_args__
0      00000b48  __pruabi_addd
0      000017f4  __pruabi_asri
0      0000172c  __pruabi_cvtfd
0      00001500  __pruabi_eqd
0      0000161c  __pruabi_fixdi
0      000013c4  __pruabi_fixdu
0      000016a8  __pruabi_fltid
0      000017a4  __pruabi_fltuf
0      0000110c  __pruabi_ged
0      00001058  __pruabi_gtd
0      000011bc  __pruabi_ltd
0      000008f4  __pruabi_mpyd
0      00000d90  __pruabi_mpyf
0      0000184c  __pruabi_mpyi
0      00001870  __pruabi_negd
0      00001464  __pruabi_nround
0      00001820  __pruabi_subd
0      00001890  __pruabi_trunc
0      00000000  _c_int00_noinit_noargs_noexit
1      00000000  _stack
0      000018b0  abort
abs    ffffffff  binit
0      0000126c  main
0      00000f58  modf
1      00000140  pru_remoteproc_ResourceTable
0      00000014  sin
```

(continues on next page)

(continued from previous page)

```
GLOBAL SYMBOLS: SORTED BY Symbol Address

page  address   name
----  -------   ----
0     00000000  _c_int00_noinit_noargs_noexit
0     00000014  sin
0     00000388  __TI_frcmpyd
0     0000069c  __TI_frcaddd
0     000008f4  __pruabi_mpyd
0     00000b48  __pruabi_addd
0     00000d90  __pruabi_mpyf
0     00000f58  modf
0     00001058  __pruabi_gtd
0     0000110c  __pruabi_ged
0     000011bc  __pruabi_ltd
0     0000126c  main
0     0000131c  __TI_frcmpyf
0     000013c4  __pruabi_fixdu
0     00001464  __pruabi_nround
0     00001500  __pruabi_eqd
0     00001590  __TI_renormd
0     0000161c  __pruabi_fixdi
0     000016a8  __pruabi_fltid
0     0000172c  __pruabi_cvtfd
0     000017a4  __pruabi_fltuf
0     000017f4  __pruabi_asri
0     00001820  __pruabi_subd
0     0000184c  __pruabi_mpyi
0     00001870  __pruabi_negd
0     00001890  __pruabi_trunc
0     000018b0  abort
0     000018b8  C$$EXIT
1     00000000  _stack
1     00000100  __TI_STACK_END
1     00000140  pru_remoteproc_ResourceTable
2     00026000  CT_CFG
abs   00000000  __PRU_CREG_BASE_PRU_DMEM_0_1
abs   00000000  __PRU_CREG_PRU_INTC
abs   00000001  __PRU_CREG_DMTIMER2
abs   00000002  __PRU_CREG_I2C1
abs   00000003  __PRU_CREG_PRU_ECAP
abs   00000004  __PRU_CREG_PRU_CFG
abs   00000005  __PRU_CREG_MMCHS0
abs   00000006  __PRU_CREG_MCSPI0
abs   00000007  __PRU_CREG_PRU_UART
abs   00000008  __PRU_CREG_MCASP0_DMA
abs   00000009  __PRU_CREG_GEMAC
abs   0000000a  __PRU_CREG_RSVD10
abs   0000000b  __PRU_CREG_UART1
abs   0000000c  __PRU_CREG_UART2
abs   0000000d  __PRU_CREG_RSVD13
abs   0000000e  __PRU_CREG_DCAN0
abs   0000000f  __PRU_CREG_DCAN1
abs   00000010  __PRU_CREG_MCSPI1
abs   00000011  __PRU_CREG_I2C2
abs   00000012  __PRU_CREG_PWMSS0
abs   00000013  __PRU_CREG_PWMSS1
abs   00000014  __PRU_CREG_PWMSS2
abs   00000015  __PRU_CREG_RSVD21
abs   00000016  __PRU_CREG_MBX0
```

(continues on next page)

(continued from previous page)

```
abs    00000017  __PRU_CREG_SPINLOCK
abs    00000018  __PRU_CREG_PRU_DMEM_0_1
abs    00000019  __PRU_CREG_PRU_DMEM_1_0
abs    0000001a  __PRU_CREG_PRU_IEP
abs    0000001b  __PRU_CREG_RSVD27
abs    0000001c  __PRU_CREG_PRU_SHAREDMEM
abs    0000001d  __PRU_CREG_TPCC
abs    0000001e  __PRU_CREG_L3OCMC
abs    0000001f  __PRU_CREG_DDR
abs    00000100  __TI_STACK_SIZE
abs    00002000  __PRU_CREG_BASE_PRU_DMEM_1_0
abs    00010000  __PRU_CREG_BASE_PRU_SHAREDMEM
abs    00020000  __PRU_CREG_BASE_PRU_INTC
abs    00026000  __PRU_CREG_BASE_PRU_CFG
abs    00028000  __PRU_CREG_BASE_PRU_UART
abs    0002e000  __PRU_CREG_BASE_PRU_IEP
abs    00030000  __PRU_CREG_BASE_PRU_ECAP
abs    00032000  __PRU_CREG_BASE_RSVD27
abs    00032400  __PRU_CREG_BASE_RSVD21
abs    40000000  __PRU_CREG_BASE_L3OCMC
abs    46000000  __PRU_CREG_BASE_MCASP0_DMA
abs    48022000  __PRU_CREG_BASE_UART1
abs    48024000  __PRU_CREG_BASE_UART2
abs    4802a000  __PRU_CREG_BASE_I2C1
abs    48030000  __PRU_CREG_BASE_MCSPI0
abs    48040000  __PRU_CREG_BASE_DMTIMER2
abs    48060000  __PRU_CREG_BASE_MMCHS0
abs    480c8000  __PRU_CREG_BASE_MBX0
abs    480ca000  __PRU_CREG_BASE_SPINLOCK
abs    4819c000  __PRU_CREG_BASE_I2C2
abs    481a0000  __PRU_CREG_BASE_MCSPI1
abs    481cc000  __PRU_CREG_BASE_DCAN0
abs    481d0000  __PRU_CREG_BASE_DCAN1
abs    48300000  __PRU_CREG_BASE_PWMSS0
abs    48302000  __PRU_CREG_BASE_PWMSS1
abs    48304000  __PRU_CREG_BASE_PWMSS2
abs    48310000  __PRU_CREG_BASE_RSVD13
abs    48318000  __PRU_CREG_BASE_RSVD10
abs    49000000  __PRU_CREG_BASE_TPCC
abs    4a100000  __PRU_CREG_BASE_GEMAC
abs    80000000  __PRU_CREG_BASE_DDR
abs    ffffffff  __binit__
abs    ffffffff  __c_args__
abs    ffffffff  binit

[100 symbols]
```

```
lines=1..22
```

Notice line 15 shows 0x18c0 bytes are being used for instructions. That's 6336 in decimal.

Now compile for the sawtooth and you see only 444 byes are used. Floating-point requires over 5K more bytes. Use with care. If you are short on instruction space, you can move the table generation to the ARM and just copy the table to the PRU.

WS2812 (NeoPixel) driver

Problem You have an Adafruit NeoPixel LED string or Adafruit NeoPixel LED matrix and want to light it up.

Solution NeoPixel is Adafruit's name for the WS2812 Intelligent control LED. Each NeoPixel contains a Red, Green and Blue LED with a PWM controller that can dim each one individually making a rainbow of colors possible. The NeoPixel is driven by a single serial line. The timing on the line is very sensesitive, which make the PRU a perfect candidate for driving it.

Wire the input to `P9_29` and power to 3.3V and ground to ground as shown in *NeoPixel Wiring*.

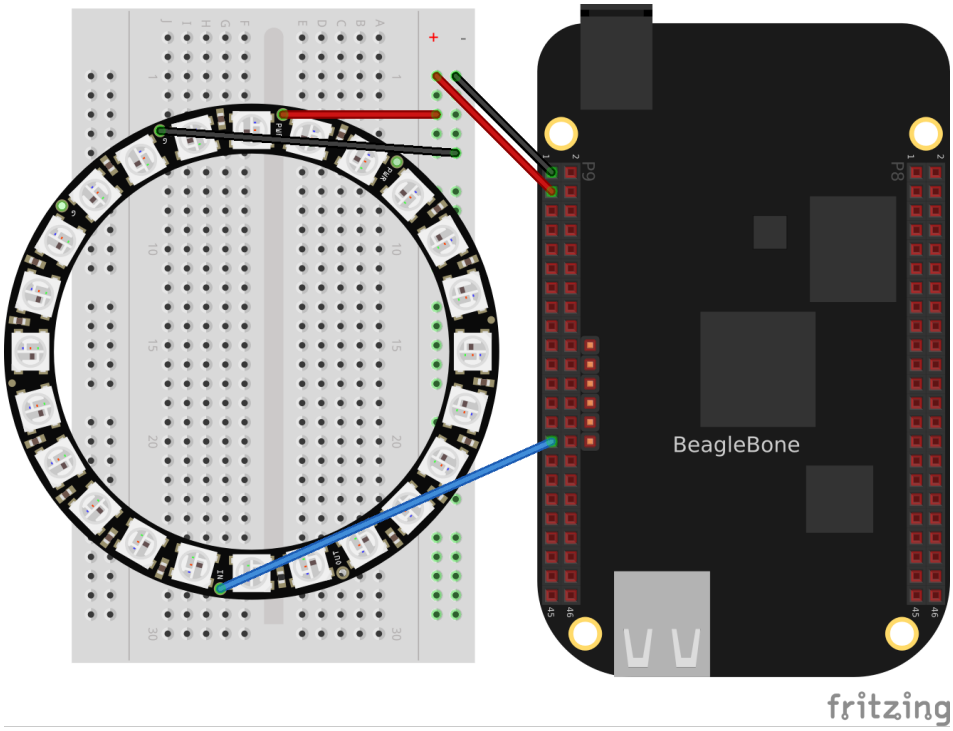


Fig. 15.167: NeoPixel Wiring

Test your wiring with the simple code in *neo1.pru0.c - Code to turn all NeoPixels's white* which to turns all pixels white.

Listing 15.101: neo1.pru0.c - Code to turn all NeoPixels's white

```
// Control a ws2812 (NeoPixel) display, All on or all off
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"

#define STR_LEN 24
#define         oneCyclesOn                 700/5        // Stay on 700ns
#define oneCyclesOff        800/5
#define zeroCyclesOn        350/5
#define zeroCyclesOff        600/5
#define resetCycles                60000/5        // Must be at least 50u, use 60u
#define gpio P9_29                                // output pin

#define ONE

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
```

(continues on next page)

(continued from previous page)

```
23          CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
24
25          uint32_t i;
26          for(i=0; i<STR_LEN*3*8; i++) {
27  #ifdef ONE
28                  __R30 |= gpio;                  // Set the GPIO pin to 1
29                  __delay_cycles(oneCyclesOn-1);
30                  __R30 &= ~gpio;                  // Clear the GPIO pin
31                  __delay_cycles(oneCyclesOff-2);
32  #else
33                  __R30 |= gpio;                  // Set the GPIO pin to 1
34                  __delay_cycles(zeroCyclesOn-1);
35                  __R30 &= ~gpio;                  // Clear the GPIO pin
36                  __delay_cycles(zeroCyclesOff-2);
37  #endif
38          }
39          // Send Reset
40          __R30 &= ~gpio;         // Clear the GPIO pin
41          __delay_cycles(resetCycles);
42
43          __halt();
44  }
```

neo1.pru0.c

Discussion *NeoPixel bit sequence* (taken from WS2812 Data Sheet) shows the following waveforms are used to send a bit of data.

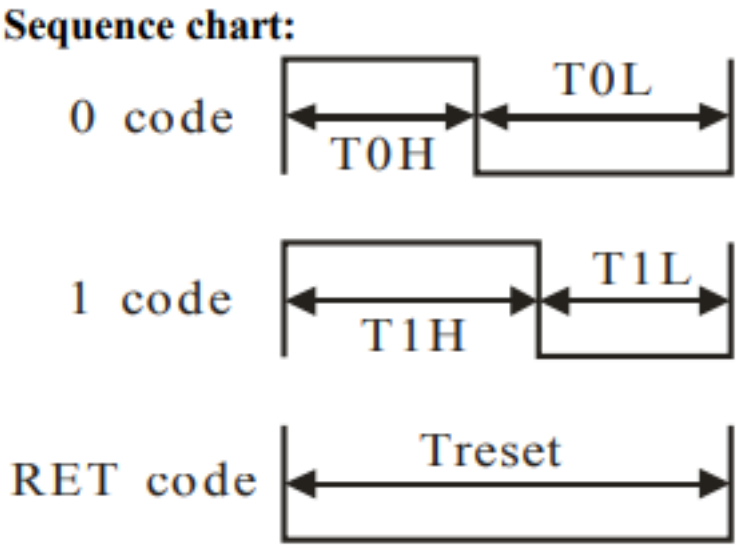


Fig. 15.168: NeoPixel bit sequence

Table 15.23: Where the times are:

Label	Time in ns
T0H	350
T0L	800
T1H	700
T1L	600
Treset	>50,000

The code in *neo1.pru0.c - Code to turn all NeoPixels's white* define these times in lines 7-10. The */5* is because

each instruction take 5ns. Lines 27-30 then set the output to 1 for the desired time and then to 0 and keeps repeating it for the entire string length. *NeoPixel zero timing* shows the waveform for sending a 0 value. Note the times are spot on.

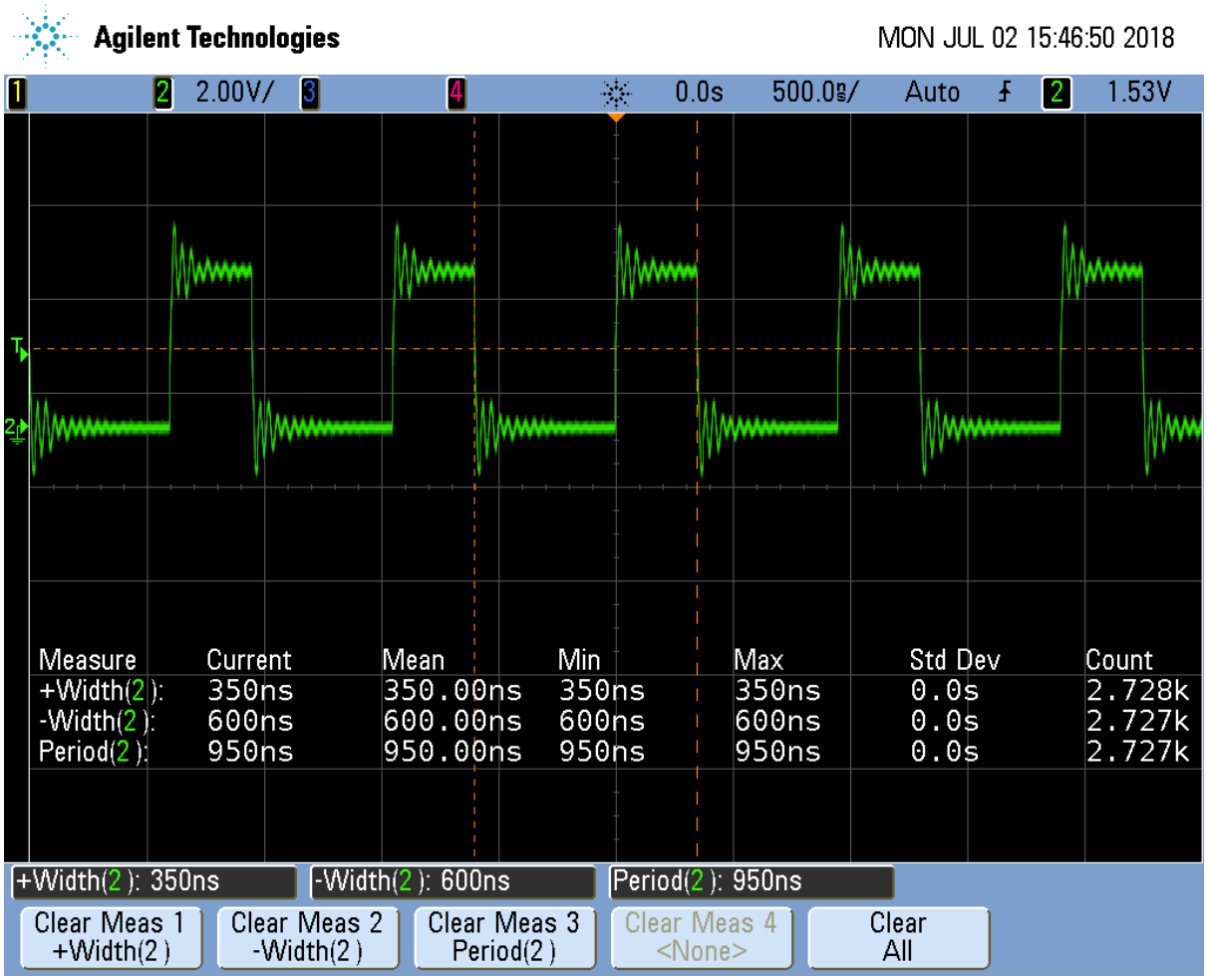


Fig. 15.169: NeoPixel zero timing

Each NeoPixel listens for a RGB value. Once a value has arrived all other values that follow are passed on to the next NeoPixel which does the same thing. That way you can individually control all of the NeoPixels.

Lines 38-40 send out a reset pulse. If a NeoPixel sees a reset pulse it will grab the next value for itself and start over again.

Setting NeoPixels to Different Colors

Problem I want to set the LEDs to different colors.

Solution Wire your NeoPixels as shown in *NeoPixel Wiring* then run the code in *neo2.pru0.c - Code to turn on green, red, blue*.

Listing 15.102: neo2.pru0.c - Code to turn on green, red, blue

```
1 // Control a ws2812 (neo pixel) display, green, red, blue, green, ...
2 #include <stdint.h>
3 #include <pru_cfg.h>
4 #include "resource_table_empty.h"
5 #include "prugpio.h"
```

(continues on next page)

(continued from previous page)

```

#define STR_LEN 3
#define        oneCyclesOn                700/5        // Stay on 700ns
#define oneCyclesOff        800/5
#define zeroCyclesOn        350/5
#define zeroCyclesOff        600/5
#define resetCycles                60000/5        // Must be at least 50u, use 60u
#define gpio P9_29                                // output pin

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        uint32_t color[STR_LEN] = {0x0f0000, 0x000f00, 0x0000f};        // green, red, blue
        int i, j;

        for(j=0; j<STR_LEN; j++) {
                for(i=23; i>=0; i--) {
                        if(color[j] & (0x1<<i)) {
                                __R30 |= gpio;                // Set the GPIO pin to 1
                                __delay_cycles(oneCyclesOn-1);
                                __R30 &= ~gpio;                // Clear the GPIO pin
                                __delay_cycles(oneCyclesOff-2);
                        } else {
                                __R30 |= gpio;                // Set the GPIO pin to 1
                                __delay_cycles(zeroCyclesOn-1);
                                __R30 &= ~gpio;                // Clear the GPIO pin
                                __delay_cycles(zeroCyclesOff-2);
                        }
                }
        }
        // Send Reset
        __R30 &= ~gpio;        // Clear the GPIO pin
        __delay_cycles(resetCycles);

        __halt();
}
```

neo2.pru0.c

This will make the first LED green, the second red and the third blue.

Discussion *NeoPixel data sequence* shows the sequence of bits used to control the green, red and blue values.

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Fig. 15.170: NeoPixel data sequence

Note: The usual order for colors is RGB (red, green, blue), but the NeoPixels use GRB (green, red, blue).

Line-by-line for neo2.pru0.c is the line-by-line for `neo2.pru0.c`.

Table 15.24: Line-by-line for neo2.pru0.c

Line 23	Explanation Define the string of colors to be output. Here the ordering of the bits is the same as *NeoPixel data sequence*, GRB.
26	Loop for each color to output.
27	Loop for each bit in an GRB color.
28	Get the j^th^ color and mask off all but the i^th^ bit. *(0x1:ref:'i)* takes the value *0x1* and shifts it left *i* bits. When anded (&) with *color[j]* it will zero out all but the i^th^ bit. If the result of the operation is 1, the *if* is done, otherwise the *else* is done.
29-32	Send a 1.
34-37	Send a 0.
42-43	Send a reset pulse once all the colors have been sent.

Note: This will only change the first `STR_LEN` LEDs. The LEDs that follow will not be changed.

Controlling Arbitrary LEDs

Problem I want to change the 10^th^ LED and not have to change the others.

Solution You need to keep an array of colors for the whole string in the PRU. Change the color of any pixels you want in the array and then send out the whole string to the LEDs. *neo3.pru0.c - Code to animate a red pixel running around a ring of blue* shows an example animates a red pixel running around a ring of blue background. *Neo3 Video* shows the code in action.

Listing 15.103: neo3.pru0.c - Code to animate a red pixel running around a ring of blue

```
1  // Control a ws2812 (neo pixel) display, green, red, blue, green, ...
2  #include <stdint.h>
3  #include <pru_cfg.h>
4  #include "resource_table_empty.h"
5  #include "prugpio.h"
6
7  #define STR_LEN 24
8  #define      oneCyclesOn               700/5        // Stay on 700ns
9  #define oneCyclesOff        800/5
10 #define zeroCyclesOn        350/5
11 #define zeroCyclesOff        600/5
12 #define resetCycles               60000/5        // Must be at least 50u,␣
   ↪use 60u
13 #define gpio P9_29                               // output pin
14
15 #define SPEED 20000000/5               // Time to wait between updates
16
17 volatile register uint32_t __R30;
18 volatile register uint32_t __R31;
19
20 void main(void)
21 {
22         uint32_t background = 0x00000f;
23         uint32_t foreground = 0x000f00;
24
```

(continues on next page)

(continued from previous page)

```
        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        uint32_t color[STR_LEN];        // green, red, blue
        int i, j;
        int k, oldk = 0;;
        // Set everything to background
        for(i=0; i<STR_LEN; i++) {
                color[i] = background;
        }

        while(1) {
                // Move forward one position
                for(k=0; k<STR_LEN; k++) {
                        color[oldk] = background;
                        color[k]    = foreground;
                        oldk=k;

                        // Output the string
                        for(j=0; j<STR_LEN; j++) {
                                for(i=23; i>=0; i--) {
                                        if(color[j] & (0x1<<i)) {
                                                __R30 |= gpio;
↪                 // Set the GPIO pin to 1
                                                __delay_cycles(oneCyclesOn-
↪1);
                                                __R30 &= ~gpio;
↪                 // Clear the GPIO pin
                                                __delay_cycles(oneCyclesOff-
↪2);
                                        } else {
                                                __R30 |= gpio;
↪                 // Set the GPIO pin to 1
                                                __delay_cycles(zeroCyclesOn-
↪1);
                                                __R30 &= ~gpio;
↪                 // Clear the GPIO pin
                                                __delay_cycles(zeroCyclesOff-
↪2);
                                        }
                                }
                        }
                        // Send Reset
                        __R30 &= ~gpio;        // Clear the GPIO pin
                        __delay_cycles(resetCycles);

                        // Wait
                        __delay_cycles(SPEED);
                }
        }
}
```

neo3.pru0.c

Neo3 Video neo3.pru0.c - Simple animation

Discussion

Table 15.25: Here's the highlights.

Line	Explanation
32,33	Initiallize the array of colors.
38-41	Update the array.
44-58	Send the array to the LEDs.
60-61	Send a reset.
64	Wait a bit.

Controlling NeoPixels Through a Kernel Driver

Problem You want to control your NeoPixels through a kernel driver so you can control it through a `/dev` interface.

Solution The rpmsg_pru driver provides a way to pass data between the ARM processor and the PRUs. It's already included on current images. *neo4.pru0.c - Code to talk to the PRU via rpmsg_pru* shows an example.

Listing 15.104: neo4.pru0.c - Code to talk to the PRU via rpmsg_pru

```
// Use rpmsg to control the NeoPixels via /dev/rpmsg_pru30
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>                         // atoi
#include <string.h>
#include <pru_cfg.h>
#include <pru_intc.h>
#include <rsc_types.h>
#include <pru_rpmsg.h>
#include "resource_table_0.h"
#include "prugpio.h"

volatile register uint32_t __R30;
volatile register uint32_t __R31;

/* Host-0 Interrupt sets bit 30 in register R31 */
#define HOST_INT                        ((uint32_t) 1 << 30)

/* The PRU-ICSS system events used for RPMsg are defined in the Linux device tree
 * PRU0 uses system event 16 (To ARM) and 17 (From ARM)
 * PRU1 uses system event 18 (To ARM) and 19 (From ARM)
 */
#define TO_ARM_HOST                         16
#define FROM_ARM_HOST               17

/*
* Using the name 'rpmsg-pru' will probe the rpmsg_pru driver found
* at linux-x.y.z/drivers/rpmsg/rpmsg_pru.c
*/
#define CHAN_NAME                       "rpmsg-pru"
#define CHAN_DESC                       "Channel 30"
#define CHAN_PORT                       30

/*
 * Used to make sure the Linux drivers are ready for RPMsg communication
 * Found at linux-x.y.z/include/uapi/linux/virtio_config.h
 */
#define VIRTIO_CONFIG_S_DRIVER_OK       4

char payload[RPMSG_BUF_SIZE];
```

(continues on next page)

(continued from previous page)

```

#define STR_LEN 24
#define        oneCyclesOn                700/5        // Stay on for 700ns
#define oneCyclesOff        600/5
#define zeroCyclesOn        350/5
#define zeroCyclesOff        800/5
#define resetCycles                51000/5        // Must be at least 50u, use 51u
#define out P9_29                                // Bit number to output on

#define SPEED 20000000/5                // Time to wait between updates

uint32_t color[STR_LEN];        // green, red, blue

/*
 * main.c
 */
void main(void)
{
        struct pru_rpmsg_transport transport;
        uint16_t src, dst, len;
        volatile uint8_t *status;

        uint8_t r, g, b;
        int i, j;
        // Set everything to background
        for(i=0; i<STR_LEN; i++) {
                color[i] = 0x010000;
        }

        /* Allow OCP master port access by the PRU so the PRU can read external memories */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        /* Clear the status of the PRU-ICSS system event that the ARM will use to 'kick' us */
#ifdef CHIP_IS_am57xx
        CT_INTC.SICR_bit.STATUS_CLR_INDEX = FROM_ARM_HOST;
#else
        CT_INTC.SICR_bit.STS_CLR_IDX = FROM_ARM_HOST;
#endif

        /* Make sure the Linux drivers are ready for RPMsg communication */
        status = &resourceTable.rpmsg_vdev.status;
        while (!(*status & VIRTIO_CONFIG_S_DRIVER_OK));

        /* Initialize the RPMsg transport structure */
        pru_rpmsg_init(&transport, &resourceTable.rpmsg_vring0, &resourceTable.rpmsg_vring1, TO_ARM_HOST, FROM_ARM_HOST);

        /* Create the RPMsg channel between the PRU and ARM user space using the transport structure. */
        while (pru_rpmsg_channel(RPMSG_NS_CREATE, &transport, CHAN_NAME, CHAN_DESC, CHAN_PORT) != PRU_RPMSG_SUCCESS);
        while (1) {
                /* Check bit 30 of register R31 to see if the ARM has kicked us */
                if (__R31 & HOST_INT) {
                        /* Clear the event status */
#ifdef CHIP_IS_am57xx
                        CT_INTC.SICR_bit.STATUS_CLR_INDEX = FROM_ARM_HOST;
```

(continues on next page)

(continued from previous page)

```
#else
                        CT_INTC.SICR_bit.STS_CLR_IDX = FROM_ARM_HOST;
#endif
                        /* Receive all available messages, multiple messages can be sent per kick */
                        while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) == PRU_RPMSG_SUCCESS) {
                                char *ret;        // rest of payload after front character is removed
                                int index;        // index of LED to control
                                // Input format is:  index red green blue
                                index = atoi(payload);
                                // Update the array, but don't write it out.
                                if((index >=0) & (index < STR_LEN)) {
                                        ret = strchr(payload, ' ');         // Skip over index
                                        r = strtol(&ret[1], NULL, 0);
                                        ret = strchr(&ret[1], ' ');         // Skip over r, etc.
                                        g = strtol(&ret[1], NULL, 0);
                                        ret = strchr(&ret[1], ' ');
                                        b = strtol(&ret[1], NULL, 0);

                                        color[index] = (g<<16)|(r<<8)|b;         // String wants GRB
                                }
                                // When index is -1, send the array to the LED string
                                if(index == -1) {
                                        // Output the string
                                            for(j=0; j<STR_LEN; j++) {
                                                    // Cycle through each bit
                                                    for(i=23; i>=0; i--) {
                                                            if(color[j] & (0x1<<i)) {
                                                                    __R30 |= out;                // Set the GPIO pin to 1
                                                                    __delay_cycles(oneCyclesOn-1);
                                                                    __R30 &= ~out;        // Clear the GPIO pin
                                                                    __delay_cycles(oneCyclesOff-14);
                                                            } else {
                                                                    __R30 |= out;                // Set the GPIO pin to 1
                                                                    __delay_cycles(zeroCyclesOn-1);
                                                                    __R30 &= ~(out);        // Clear the GPIO pin
                                                                    __delay_cycles(zeroCyclesOff-14);
                                                            }
                                                    }
                                            }
                                            // Send Reset
                                            __R30 &= ~out;        // Clear the GPIO pin
                                            __delay_cycles(resetCycles);

                                            // Wait
```

(continues on next page)

(continued from previous page)

```
                                                __delay_cycles(SPEED);
                                    }

                                }
                            }
                    }
}
```

`neo4.pru0.c`

Run the code as usual.

```
bone$ make TARGET=neo4.pru0
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=neo4.pru0
-    Stopping PRU 0
-     copying firmware file /tmp/vsx-examples/neo4.pru0.out to /lib/firmware/
↪am335x-pru0-fw
write_init_pins.sh
-    Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR = /sys/class/remoteproc/remoteproc1

bone$ echo 0 0xff 0 127 > /dev/rpmsg_pru30
bone$ echo -1 > /dev/rpmsg_pru30
```

`/dev/rpmsg_pru30` is a device driver that lets the ARM talk to the PRU. The first `echo` says to set the 0^th^ LED to RGB value 0xff 0 127. (Note: you can mix hex and decimal.) The second `echo` tells the driver to send the data to the LEDs. Your 0^th^ LED should now be lit.

Discussion There's a lot here. I'll just hit some of the highlights in *Line-by-line for neo4.pru0.c*.

Table 15.26: Line-by-line for neo4.pru0.c

Line	Explanation
30	The *CHAN_NAME* of *rpmsg-pru* matches that *prmsg_pru* driver that is is already installed. This connects this PRU to the driver.
32	The *CHAN_PORT* tells it to use port 30. That's why we use */dev/rpmsg_pru30*
40	*payload[]* is the buffer that receives the data from the ARM.
42-48	Same as the previous NeoPixel examples.
52	*color[]* is the state to be sent to the LEDs.
66-68	*color[]* is initialized.
70-85	Here are a number of details needed to set up the channel between the PRU and the ARM.
88	Here we wait until the ARM sends us some numbers.
99	Receive all the data from the ARM, store it in *payload[]*.
101-111	The data sent is: index red green blue. Pull off the index. If it's in the right range, pull off the red, green and blue values.
113	The NeoPixels want the data in GRB order. Shift and OR everything together.
116-133	If the *index* = -1, send the contents of *color* to the LEDs. This code is same as before.

You can now use programs running on the ARM to send colors to the PRU.

neo-rainbow.py - A python program using /dev/rpmsg_pru30 shows an example.

Listing 15.105: neo-rainbow.py - A python program using /dev/rpmsg_pru30

```
#!/usr/bin/python3
from time import sleep
import math
```

(continues on next page)

(continued from previous page)

```
4
5  len = 24
6  amp = 12
7  f = 25
8  shift = 3
9  phase = 0
10
11 # Open a file
12 fo = open("/dev/rpmsg_pru30", "wb", 0)
13
14 while True:
15     for i in range(0, len):
16         r = (amp * (math.sin(2*math.pi*f*(i-phase-0*shift)/len) + 1)) + 1;
17         g = (amp * (math.sin(2*math.pi*f*(i-phase-1*shift)/len) + 1)) + 1;
18         b = (amp * (math.sin(2*math.pi*f*(i-phase-2*shift)/len) + 1)) + 1;
19         fo.write(b"%d %d %d %d\n" % (i, r, g, b))
20         # print("0 0 127 %d" % (i))
21
22     fo.write(b"-1 0 0 0\n");
23     phase = phase + 1
24     sleep(0.05)
25
26 # Close opened file
27 fo.close()
```

neo-rainbow.py

Line 19 writes the data to the PRU. Be sure to have a newline, or space after the last number, or you numbers will get blurred together.

Switching from pru0 to pru1 with rpmsg_pru There are three things you need to change when switching from pru0 to pru1 when using rpmsg_pru.

1. The include on line 10 is switched to `#include "resource_table_1.h"` (0 is switched to a 1)
2. Line 17 is switched to `#define HOST_INT ((uint32_t) 1 << 31)` (30 is switched to 31.)
3. Lines 23 and 24 are switched to:

```
#define TO_ARM_HOST                    18
#define FROM_ARM_HOST          19
```

These changes switch to the proper channel numbers to use pru1 instead of pru0.

RGB LED Matrix - No Integrated Drivers

Problem You have a RGB LED matrix (*RGB LED Matrix – No Integrated Drivers (Falcon Christmas)*) and want to know at a low level how the PRU works.

Solution Here is the datasheet, but the best description I've found for the RGB Matrix is from Adafruit. I've reproduced it here, with adjustments for the 64x32 matrix we are using.

---

information

There's zero documentation out there on how these matrices work, and no public datasheets or spec sheets so we are going to try to document how they work.

First thing to notice is that there are 2048 RGB LEDs in a 64x32 matrix. Like pretty much every matrix out there, you can't drive all 2048 at once. One reason is that would require a lot of current, another reason is that it would be really expensive to have so many pins. Instead, the matrix is divided into 16 interleaved

sections/strips. The first section is the 1^st^ 'line' and the 17^th^ 'line' (64 x 2 RGB LEDs = 128 RGB LEDs), the second is the 2^nd^ and 18^th^ line, etc until the last section which is the 16^th^ and 32^nd^ line. You might be asking, why are the lines paired this way? wouldn't it be nicer to have the first section be the 1^st^ and 2^nd^ line, then 3^rd^ and 4^th^, until the 15^th^ and 16^th^? The reason they do it this way is so that the lines are interleaved and look better when refreshed, otherwise we'd see the stripes more clearly.

So, on the PCB is 24 LED driver chips. These are like 74HC595s but they have 16 outputs and they are constant current. 16 outputs * 24 chips = 384 LEDs that can be controlled at once, and 128 * 3 (R G and B) = 384. So now the design comes together: You have 384 outputs that can control one line at a time, with each of 384 R, G and B LEDs either on or off. The controller (say an FPGA or microcontroller) selects which section to currently draw (using LA, LB, LC and LD address pins - 4 bits can have 16 values). Once the address is set, the controller clocks out 384 bits of data (48 bytes) and latches it. Then it increments the address and clocks out another 384 bits, etc until it gets to address #15, then it sets the address back to #0

https://cdn-learn.adafruit.com/downloads/pdf/32x16-32x32-rgb-led-matrix.pdf

---

That gives a good overview, but there are a few details missing. *rgb_python.py - Python code for driving RGB LED matrix* is a functioning python program that gives a nice high-level view of how to drive the display.

Listing 15.106: rgb_python.py - Python code for driving RGB LED matrix

```
#!/usr/bin/env python3
import Adafruit_BBIO.GPIO as GPIO

# Define which functions are connect to which pins
OE="P1_29"      # Output Enable, active low
LAT="P1_36"     # Latch, toggle after clocking in a row of pixels
CLK="P1_33"     # Clock, toggle after each pixel

# Input data pins
R1="P2_10"  # R1, G1, B1 are for the top rows (1-16) of pixels
G1="P2_8"
B1="P2_6"

R2="P2_4"   # R2, G2, B2 are for the bottom rows (17-32) of pixels
G2="P2_2"
B2="P2_1"

LA="P2_32"  # Address lines for which row (1-16 or 17-32) to update
LB="P2_30"
LC="P1_31"
LD="P2_34"

# Set everything as output ports
GPIO.setup(OE,  GPIO.OUT)
GPIO.setup(LAT, GPIO.OUT)
GPIO.setup(CLK, GPIO.OUT)

GPIO.setup(R1, GPIO.OUT)
GPIO.setup(G1, GPIO.OUT)
GPIO.setup(B1, GPIO.OUT)
GPIO.setup(R2, GPIO.OUT)
GPIO.setup(G2, GPIO.OUT)
GPIO.setup(B2, GPIO.OUT)

GPIO.setup(LA, GPIO.OUT)
GPIO.setup(LB, GPIO.OUT)
GPIO.setup(LC, GPIO.OUT)
GPIO.setup(LD, GPIO.OUT)

GPIO.output(OE,  0)     # Enable the display
GPIO.output(LAT, 0)     # Set latch to low
```

(continues on next page)

(continued from previous page)

```

while True:
    for bank in range(64):
        GPIO.output(LA, bank>>0&0x1)    # Select rows
        GPIO.output(LB, bank>>1&0x1)
        GPIO.output(LC, bank>>2&0x1)
        GPIO.output(LD, bank>>3&0x1)

        # Shift the colors out.  Here we only have four different
        # colors to keep things simple.
        for i in range(16):
            GPIO.output(R1,  1)     # Top row, white
            GPIO.output(G1,  1)
            GPIO.output(B1,  1)

            GPIO.output(R2,  1)     # Bottom row, red
            GPIO.output(G2,  0)
            GPIO.output(B2,  0)

            GPIO.output(CLK, 0)     # Toggle clock
            GPIO.output(CLK, 1)

            GPIO.output(R1,  0)     # Top row, black
            GPIO.output(G1,  0)
            GPIO.output(B1,  0)

            GPIO.output(R2,  0)     # Bottom row, green
            GPIO.output(G2,  1)
            GPIO.output(B2,  0)

            GPIO.output(CLK, 0)     # Toggle clock
            GPIO.output(CLK, 1)

        GPIO.output(OE,  1)     # Disable display while updating
        GPIO.output(LAT, 1)     # Toggle latch
        GPIO.output(LAT, 0)
        GPIO.output(OE,  0)     # Enable display
```

rgb_python.py

Be sure to run the *rgb_python_setup.sh* script before running the python code.

Listing 15.107: rgb_python_setup.sh

```
#!/bin/bash
# Setup for 64x32 RGB Matrix
export TARGET=rgb1.pru0
echo TARGET=$TARGET

# Configure the PRU pins based on which Beagle is running
machine=$(awk '{print $NF}' /proc/device-tree/model)
echo -n $machine
if [ $machine = "Black" ]; then
    echo " Found"
    pins=""
elif [ $machine = "Blue" ]; then
    echo " Found"
    pins=""
elif [ $machine = "PocketBeagle" ]; then
    echo " Found"
    prupins="P2_32 P1_31 P1_33 P1_29 P2_30 P2_34 P1_36"
    gpiopins="P2_10 P2_06 P2_04 P2_01 P2_08 P2_02"
```

(continues on next page)

(continued from previous page)

```
    # Uncomment for J2
    # gpiopins="$gpiopins P2_27 P2_25 P2_05 P2_24 P2_22 P2_18"
else
    echo " Not Found"
    pins=""
fi

for pin in $prupins
do
    echo $pin
    # config-pin $pin pruout
    config-pin $pin gpio
    config-pin $pin out
    config-pin -q $pin
done

for pin in $gpiopins
do
    echo $pin
    config-pin $pin gpio
    config-pin $pin out
    config-pin -q $pin
done
```

`rgb_python_setup.sh`

Make sure line 29 is commented out and line 30 is uncommented. Later we'll configure for _pruout_, but for now the python code doesn't use the PRU outs.

```
# config-pin $pin pruout
config-pin $pin out
```

Your display should look like *Display running rgb_python.py*.

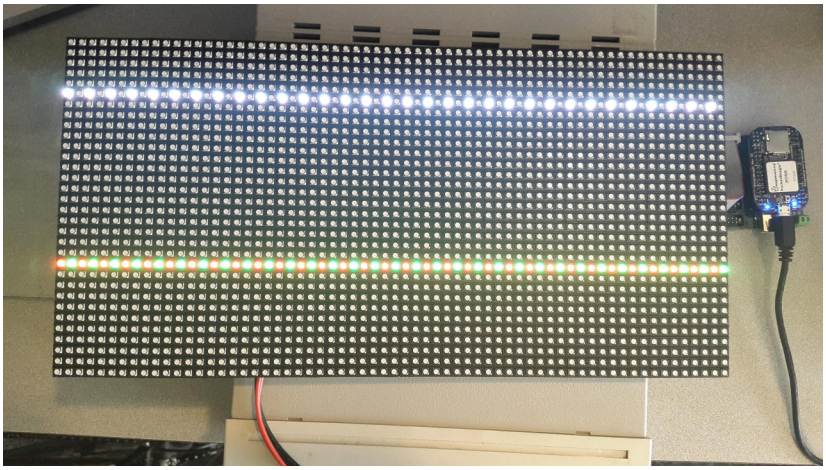


Fig. 15.171: Display running rgb_python.py

So why do only two lines appear at a time? That's how the display works. Currently lines 6 and 22 are showing, then a moment later 7 and 23 show, etc. The display can only display two lines at a time, so it cycles through all the lines. Unfortunately, python is too slow to make the display appear all at once. Here's where the PRU comes in.

:ref:`blocks_rgb1` is the PRU code to drive the RGB LED matrix. Be sure to run `bone$ source rgb_setup.sh` first.

Listing 15.108: PRU code for driving the RGB LED matrix

```
// This code drives the RGB LED Matrix on the 1st Connector
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"
#include "rgb_pocket.h"

#define DELAY 10        // Number of cycles (5ns each) to wait after a write

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        // Set up the pointers to each of the GPIO ports
        uint32_t *gpio[] = {
                        (uint32_t *) GPIO0,
                        (uint32_t *) GPIO1,
                        (uint32_t *) GPIO2,
                        (uint32_t *) GPIO3
                };

        uint32_t i, row;

        while(1) {
            for(row=0; row<16; row++) {
                    // Set the row address
                        // Here we take advantage of the select bits (LA,LB,
↪LC,LD)
                        // being sequential in the R30 register (bits 2,3,4,
↪5)
                        // We shift row over so it lines up with the select␣
↪bits
                        // Oring (|=) with R30 sets bits to 1 and
                        // Anding (&=) clears bits to 0, the 0xffc mask␣
↪makes sure the
                        // other bits aren't changed.
                __R30 |=  row<<pru_sel0;
                __R30 &= (row<<pru_sel0)|0xffc3;

                for(i=0; i<64; i++) {
                        // Top row white
                        // Combining these to one write works because they␣
↪are all in
                        // the same gpio port
                          gpio[r11_gpio][GPIO_SETDATAOUT] = r11_pin | g11_
↪pin | b11_pin;
                        __delay_cycles(DELAY);;

                          // Bottom row red
                          gpio[r12_gpio][GPIO_SETDATAOUT]   = r12_pin;
                        __delay_cycles(DELAY);
                          gpio[r12_gpio][GPIO_CLEARDATAOUT] = g12_pin | b12_
↪pin;
                        __delay_cycles(DELAY);

                __R30 |=  pru_clock;        // Toggle clock
                        __delay_cycles(DELAY);
                        __R30 &= ~pru_clock;
                        __delay_cycles(DELAY);
```

(continues on next page)

(continued from previous page)

```
54
55                          // Top row black
56                          gpio[r11_gpio][GPIO_CLEARDATAOUT] = r11_pin | g11_
    ↪pin | b11_pin;
57                          __delay_cycles(DELAY);
58
59                            // Bottom row green
60                          gpio[r12_gpio][GPIO_CLEARDATAOUT] = r12_pin | b12_
    ↪pin;
61                          __delay_cycles(DELAY);
62                            gpio[r12_gpio][GPIO_SETDATAOUT]   = g12_pin;
63                          __delay_cycles(DELAY);
64
65                  __R30 |=  pru_clock;        // Toggle clock
66                          __delay_cycles(DELAY);
67                          __R30 &= ~pru_clock;
68                          __delay_cycles(DELAY);
69                  }
70                  __R30 |=  pru_oe;         // Disable display
71                           __delay_cycles(DELAY);
72                  __R30 |=  pru_latch;      // Toggle latch
73                           __delay_cycles(DELAY);
74                  __R30 &= ~pru_latch;
75                           __delay_cycles(DELAY);
76                  __R30 &= ~pru_oe;         // Enable display
77                  __delay_cycles(DELAY);
78              }
79          }
80  }
```

`rgb1.pru0.c`

The results are shown in *Display running rgb1.c on PRU 0*.

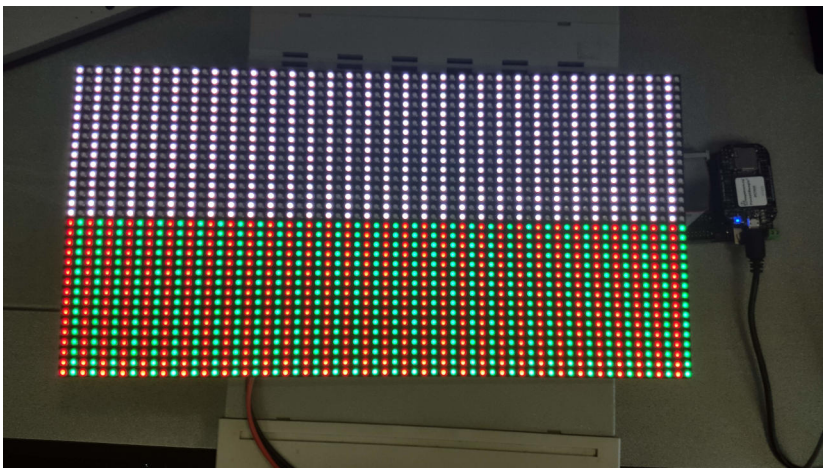


Fig. 15.172: Display running rgb1.c on PRU 0

The PRU is fast enough to quickly write to the display so that it appears as if all the LEDs are on at once.

Discussion There are a lot of details needed to make this simple display work. Let's go over some of them.

First, the connector looks like *RGB Matrix J1 connector*.

Notice the labels on the connect match the labels in the code. *PocketScroller pin table* shows how the pins on the display are mapped to the pins on the PocketBeagle.

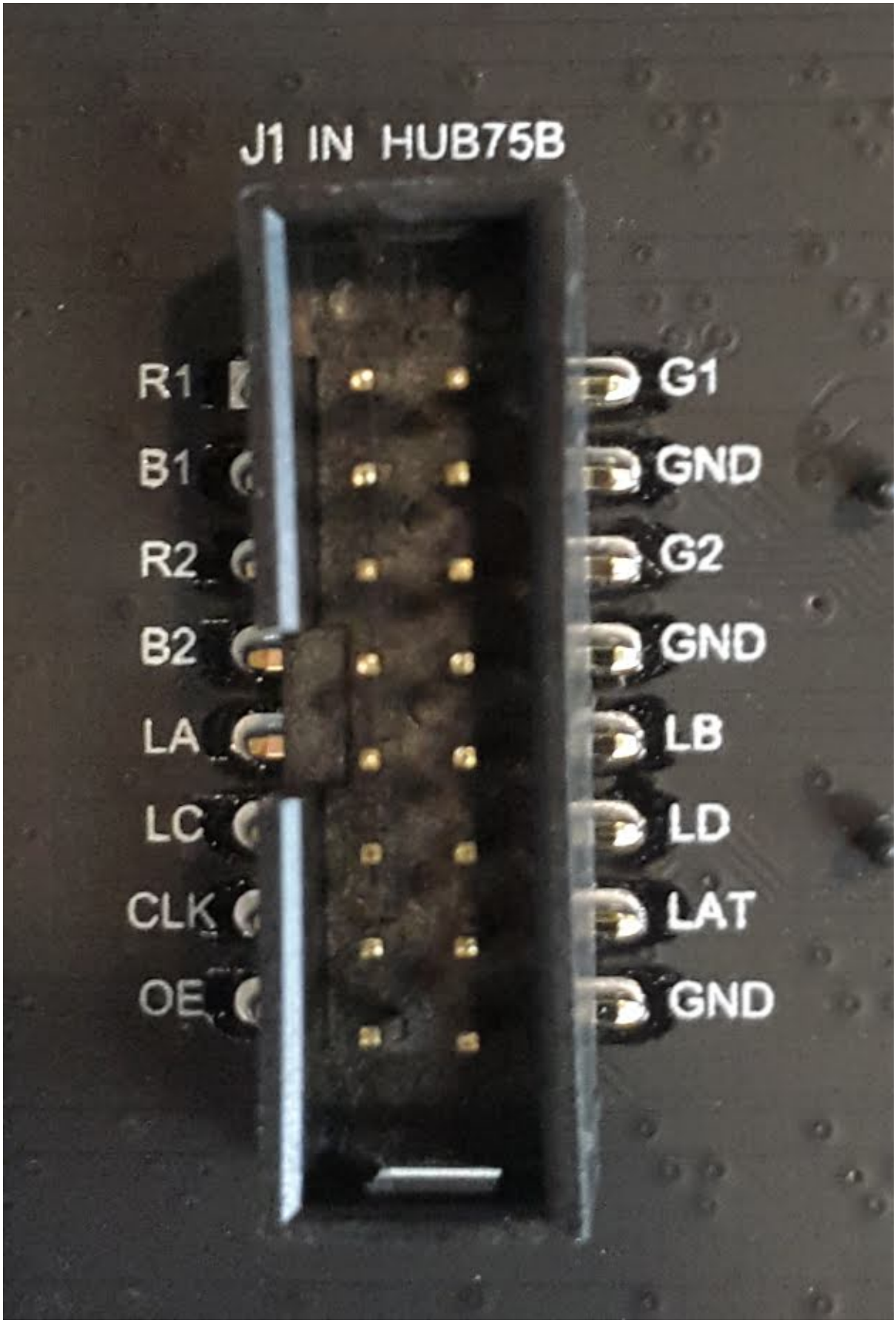


Fig. 15.173: RGB Matrix J1 connector

Todo: Make a mapping table for the Black

https://github.com/FalconChristmas/fpp/blob/master/src/pru/OctoscrollerV2.hp

Table 15.27: PocketScroller pin table

J1 Connector Pin	Pocket Headers	gpio port and bit number	Linux gpio number	PRU R30 bit number
R1	P2_10	1-20	52	
B1	P2_06	1-25	57	
R2	P2_04	1-26	58	
B2	P2_01	1-18	50	
LA	P2_32	3-16	112	PRU0.2
LC	P1_31	3-18	114	PRU0.4
CLK	P1_33	3-15	111	PRU0.1
OE	P1_29	3-21	117	PRU0.7
G1	P2_08	1-28	60	
G2	P2_02	1-27	59	
LB	P2_30	3-17	113	PRU0.3
LD	P2_34	3-19	115	PRU0.5
LAT	P1_36	3-14	110	PRU0.0

The J1 mapping to gpio port and bit number comes from https://github.com/FalconChristmas/fpp/blob/master/capes/pb/panels/PocketScroller.json. The gpio port and bit number mapping to Pocket Headers comes from https://docs.google.com/spreadsheets/d/1FRGvYOyW1RiNSEVprvstfJAVeapnASgDXHtxeDOjgqw/edit#gid=0.

Oscilloscope display of CLK, OE, LAT and R1 shows four of the signal waveforms driving the RGB LED matrix.

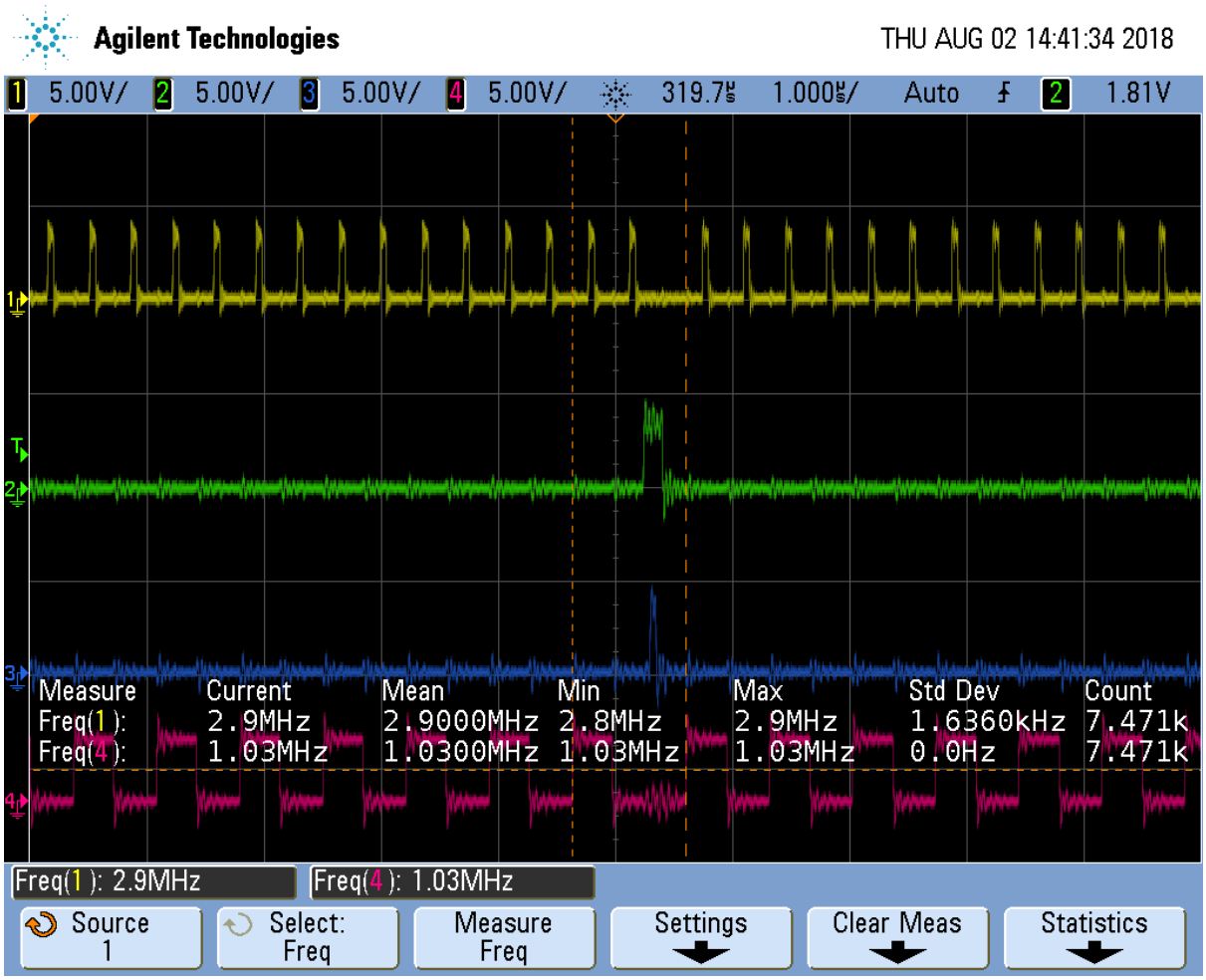


Fig. 15.174: Oscilloscope display of CLK, OE, LAT and R1

The top waveform is the CLK, the next is OE, followed by LAT and finally R1. The OE (output enable) is active low, so most of the time the display is visible. The sequence is:

- Put data on the R1, G1, B1, R2, G2 and B2 lines
- Toggle the clock.
- Repeat the first two steps as one row of data is transferred. There are 384 LEDs (2 rows of 32 RGB LEDs times 3 LED per RGB), but we are clocking in six bits (R1, G1, etc.) at a time, so 384/6=64 values need to be clocked in.
- Once all the values are in, disable the display (OE goes high)
- Then toggle the latch (LAT) to latch the new data.
- Turn the display back on.
- Increment the address lines (LA, LB, LC and LD) to point to the next rows.
- Keep repeating the above to keep the display lit.

Using the PRU we are able to run the clock a about 2.9 MKHz. *FPP waveforms* shows the optimized assembler code used by FPP clocks in at some 6.3 MHz. So the compiler is doing a pretty good job, but you can run some two times faster if you want to use assembly code. In fairness to FPP, it's having to pull it's data out of RAM to display it, so isn't not a good comparison.

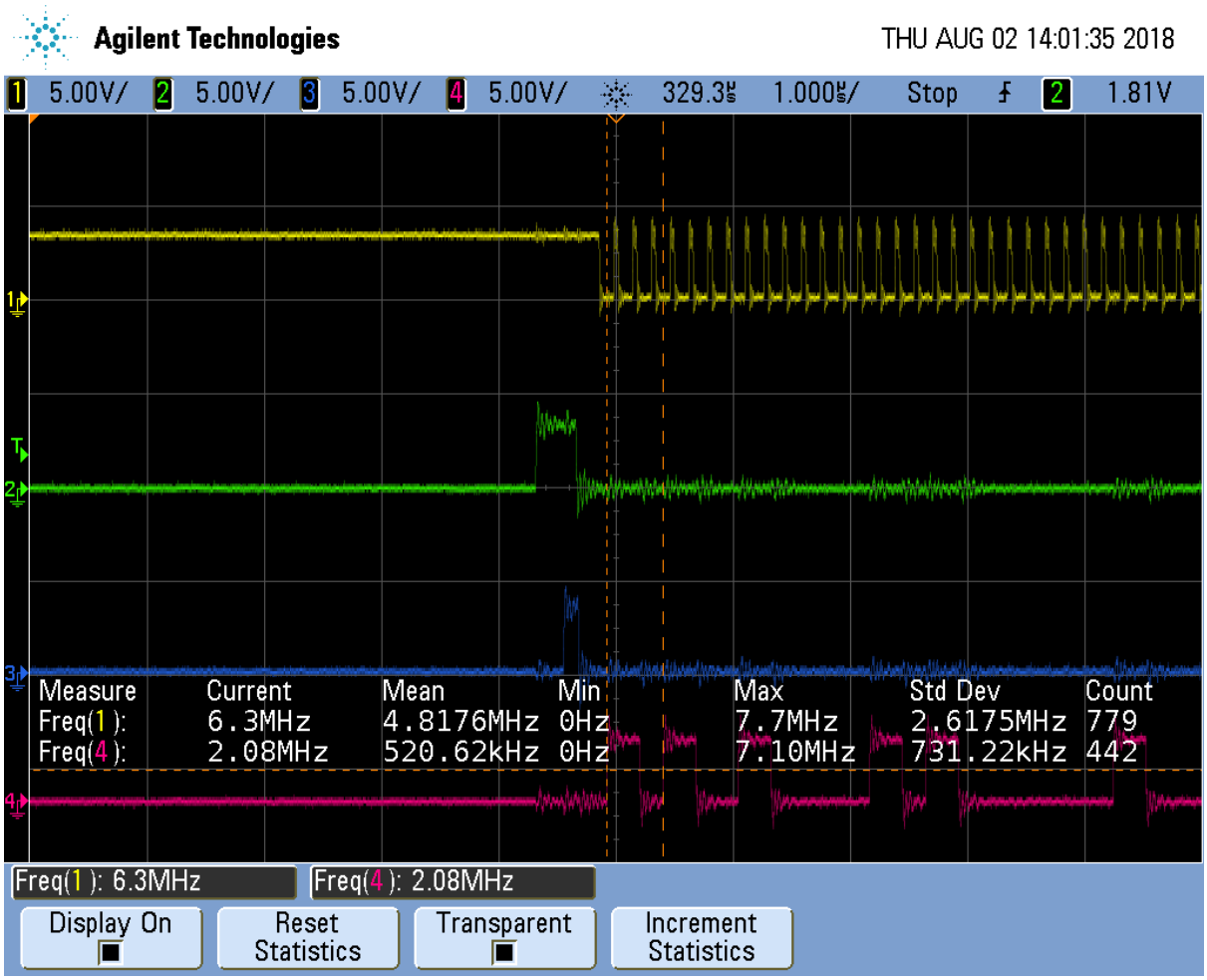


Fig. 15.175: FPP waveforms

Getting More Colors The Adafruit description goes on to say:

information

The only downside of this technique is that despite being very simple and fast, it has no PWM control built-in! The controller can only set the LEDs on or off. So what do you do when you want full color? You actually need to draw the entire matrix over and over again at very high speeds to PWM the matrix manually. For that reason, you need to have a very fast controller (50 MHz is a minimum) if you want to do a lot of colors and motion video and have it look good.

https://cdn-learn.adafruit.com/downloads/pdf/32x16-32x32-rgb-led-matrix.pdf

This is what FPP does, but it's beyond the scope of this project.

Compiling and Inserting rpmsg_pru

Problem Your Beagle doesn't have rpmsg_pru.

Solution Do the following.

```
bone$ *cd code/05blocks/module*
bone$ *sudo apt install linux-headers-\`uname -r`*
bone$ *wget https://github.com/beagleboard/linux/raw/4.9/drivers/rpmsg/rpmsg_
↪pru.c*
bone$ *make*
make -C /lib/modules/4.9.88-ti-r111/build M=$PWD
make[1]: Entering directory '/usr/src/linux-headers-4.9.88-ti-r111'
  LD      /home/debian/PRUCookbook/docs/code/05blocks/module/built-in.o
  CC [M]  /home/debian/PRUCookbook/docs/code/05blocks/module/rpmsg_client_
↪sample.o
  CC [M]  /home/debian/PRUCookbook/docs/code/05blocks/module/rpmsg_pru.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /home/debian/PRUCookbook/docs/code/05blocks/module/rpmsg_client_
↪sample.mod.o
  LD [M]  /home/debian/PRUCookbook/docs/code/05blocks/module/rpmsg_client_
↪sample.ko
  CC      /home/debian/PRUCookbook/docs/code/05blocks/module/rpmsg_pru.mod.o
  LD [M]  /home/debian/PRUCookbook/docs/code/05blocks/module/rpmsg_pru.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.9.88-ti-r111'
bone$ *sudo insmod rpmsg_pru.ko*
bone$ *lsmod | grep rpm*
rpmsg_pru               5799  2
virtio_rpmsg_bus       13620  0
rpmsg_core              8537  2 rpmsg_pru,virtio_rpmsg_bus
```

It's now installed and ready to go.

Copyright

Listing 15.109: copyright.c

```
1 /*
2  * Copyright (C) 2015 Texas Instruments Incorporated - http://www.ti.com/
3  *
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions
7  * are met:
8  *
```

(continues on next page)

(continued from previous page)

```
 *        * Redistributions of source code must retain the above copyright
 *          notice, this list of conditions and the following disclaimer.
 *
 *        * Redistributions in binary form must reproduce the above copyright
 *          notice, this list of conditions and the following disclaimer in␣
↪the
 *          documentation and/or other materials provided with the
 *          distribution.
 *
 *        * Neither the name of Texas Instruments Incorporated nor the names␣
↪of
 *          its contributors may be used to endorse or promote products␣
↪derived
 *          from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

copyright.c

15.2.6 Accessing More I/O

So far the examples have shown how to access the GPIO pins on the BeagleBone Black's P9 header and through the pass:[__]R30 register. Below shows how more GPIO pins can be accessed.

The following are resources used in this chapter.

Note: *Resources*

- P8 Header Table
- P9 Header Table
- AM572x Technical Reference Manual (AI)
- AM335x Technical Reference Manual (All others)
- PRU Assembly Language Tools

Editing /boot/uEnv.txt to Access the P8 Header on the Black

Problem When I try to configure some pins on the *P8* header of the Black I get an error.

```
bone$ *config-pin P8_28 pruout*
ERROR: open() for /sys/devices/platform/ocp/ocp:P8_28_pinmux/state failed,␣
↪No such file or directory
```

Solution On the images for the BeagleBone Black, the HDMI display driver is enabled by default and uses many of the P8 pins. If you are not using HDMI video (or the HDI audio, or even the eMMC) you can disable it by editing /boot/uEnv.txt

Open /boot/uEnv.txt and scroll down always until you see:

Listing 15.110: /boot/uEnv.txt

```
###Disable auto loading of virtual capes (emmc/video/wireless/adc)
#disable_uboot_overlay_emmc=1
disable_uboot_overlay_video=1
#disable_uboot_overlay_audio=1
```

Uncomment the lines that correspond to the devices you want to disable and free up their pins.

Tip: P8 Header Table shows what pins are allocated for what.

Save the file and reboot. You now have access to the P8 pins.

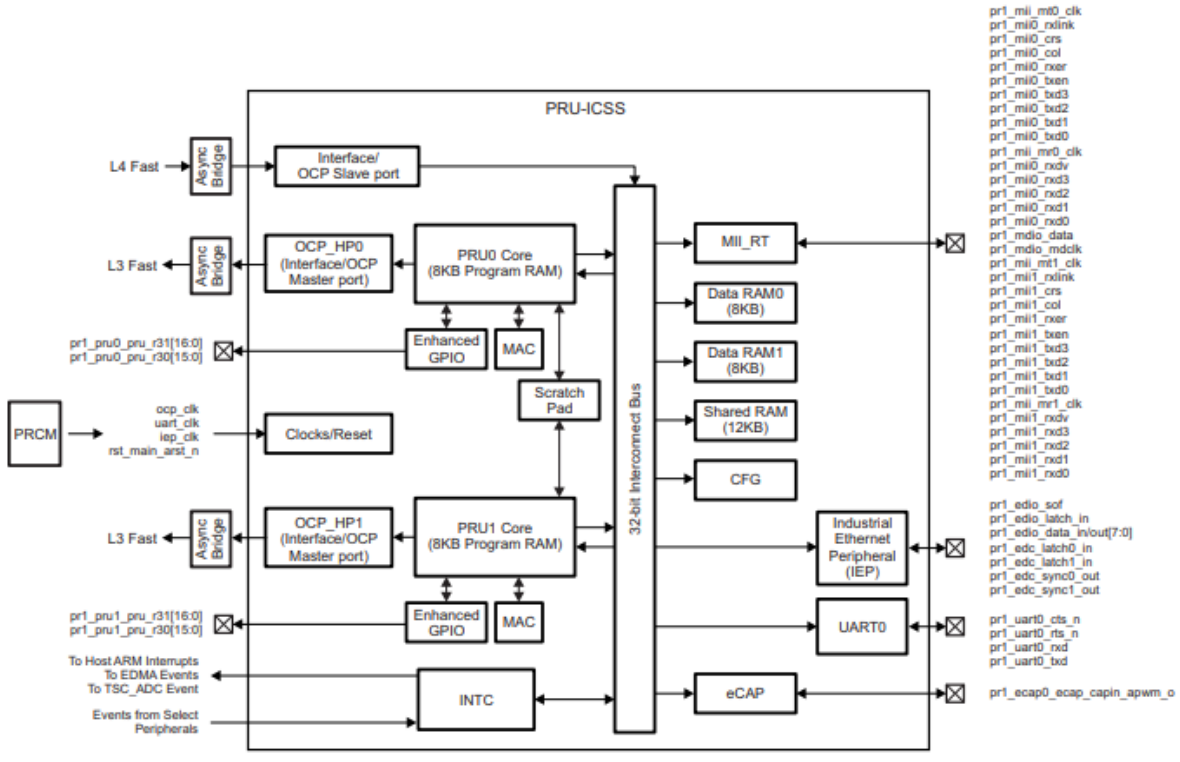
Accessing gpio

Problem I've used up all the GPIO in pass: [___]R30, where can I get more?

Solution So far we have focused on using PRU 0. *Mapping bit positions to pin names* shows that PRU 0 can access ten GPIO pins on the BeagleBone Black. If you use PRU 1 you can get to an additional 14 pins (if they aren't in use for other things.)

What if you need even more GPIO pins? You can access **any** GPIO pin by going through the **O**pen-**C**ore **P**rotocol (OCP) port.

Figure 4-2. PRU-ICSS Integration



For the availability of all features, see the device features in Chapter 1, *Introduction*.

Fig. 15.176: PRU Integration

The figure above shows we've been using the _Enhanced **GPIO** interface when using `pass: [__]R30`, but it also shows you can use the OCP. You get access to many more GPIO pins, but it's a slower access.

Listing 15.111: gpio.pru0.c

```
// This code accesses GPIO without using R30 and R31
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"

#define P9_11         (0x1<<30)                    // Bit position tied to P9_11 on Black
#define P2_05         (0x1<<30)                    // Bit position tied to P2_05 on Pocket

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t *gpio0 = (uint32_t *)GPIO0;

        while(1) {
                gpio0[GPIO_SETDATAOUT]   = P9_11;
                __delay_cycles(100000000);
                gpio0[GPIO_CLEARDATAOUT] = P9_11;
                __delay_cycles(100000000);
        }
}
```

`gpio.pru0.c`

This code will toggle `P9_11` on and off. Here's the setup file.

Listing 15.112: setup.sh

```
#!/bin/bash

export TARGET=gpio.pru0
echo TARGET=$TARGET

# Configure the PRU pins based on which Beagle is running
machine=$(awk '{print $NF}' /proc/device-tree/model)
echo -n $machine
if [ $machine = "Black" ]; then
    echo " Found"
    pins="P9_11"
elif [ $machine = "Blue" ]; then
    echo " Found"
    pins=""
elif [ $machine = "PocketBeagle" ]; then
    echo " Found"
    pins="P2_05"
else
    echo " Not Found"
    pins=""
fi

for pin in $pins
do
    echo $pin
    config-pin $pin gpio
    config-pin -q $pin
```

(continues on next page)

(continued from previous page)

```
done
```

`setup.sh`

Notice in the code `config-pin` set `P9_11` to `gpio`, not `pruout`. This is because are using the OCP interface to the pin, not the usual PRU interface.

Set your exports and make.

```
bone$ *source setup.sh*
TARGET=gpio.pru0
...
bone$ *make*
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=gpio.pru0
-    Stopping PRU 0
-   copying firmware file /tmp/vsx-examples/gpio.pru0.out to /lib/firmware/
↪am335x-pru0-fw
write_init_pins.sh
-    Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR = /sys/class/remoteproc/remoteproc1
```

Discussion When you run the code you see `P9_11` toggling on and off. Let's go through the code line-by-line to see what's happening.

Table 15.28: gpio.pru0.c line-by-line

Line	Explanation
2-5	Standard includes
5	The AM335x has four 32-bit GPIO ports. Lines 55-58 of *prugpio.h* define the addresses for each of the ports. You can find these in Table 2-2 page 180 of the AM335x TRM 180. Look up *P9_11* in the P9 header. Under the _Mode7_ column you see *gpio0[30]*. This means *P9_11* is bit 30 on GPIO port 0. Therefore we will use *GPIO0* in this code. You can also run `gpioinfo` and look for P9_11.
5	Line 103 of *prugpio.h* defines the address offset from *GIO0* that will allow us to _clear_ any (or all) bits in GPIO port 0. Other architectures require you to read a port, then change some bit, then write it out again, three steps. Here we can do the same by writing to one location, just one step.
5	Line 104 of *prugpio.h* is like above, but for _setting_ bits.
5	Using this offset of line 105 of *prugpio.h* lets us just read the bits without changing them.
7,8	This shifts *0x1* to the 30^th^ bit position, which is the one corresponding to *P9_11*.
15	Here we initialize *gpio0* to point to the start of GPIO port 0's control registers.
18	***gpio0[GPIO_SETDATAOUT]* refers to the *SETDATAOUT* register of port 0. Writing** where 1's are written, but leaves alone the bits where 0's are.
19	Wait 100,000,000 cycles, which is 0.5 seconds.
20	This is line 18, but the output bit is set to 0 where 1's are written.

How fast can it go? This approach to GPIO goes through the slower OCP interface. If you set `pass:[__]delay_cycles(0)` you can see how fast it is.

The period is 80ns which is 12.MHz. That's about one forth the speed of the `pass:[__]R30` method, but still not bad.

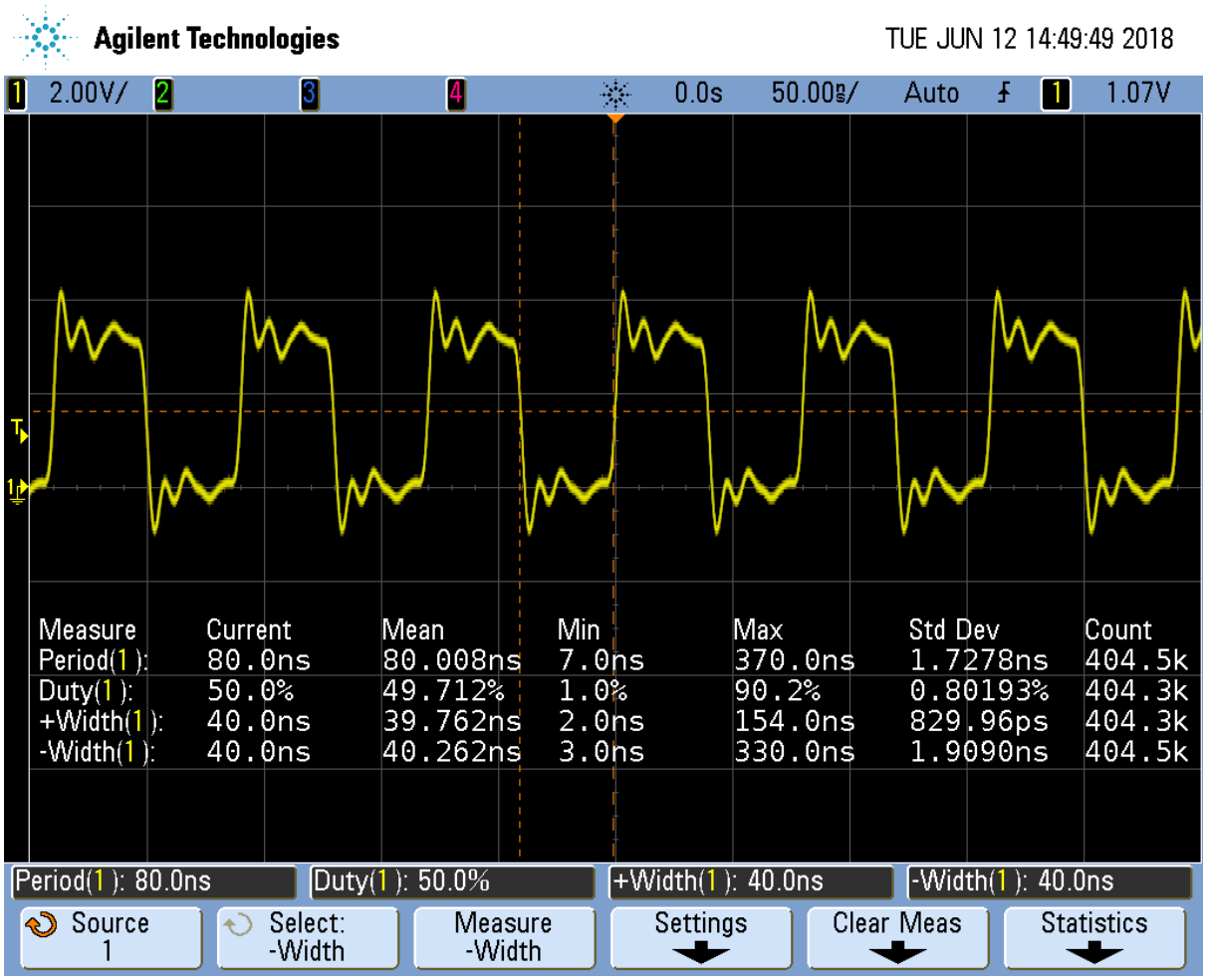


Fig. 15.177: gpio.pru0.c with pass:[__]delay_cycles(0)

If you are using an oscilloscope, look closely and you'll see the following.

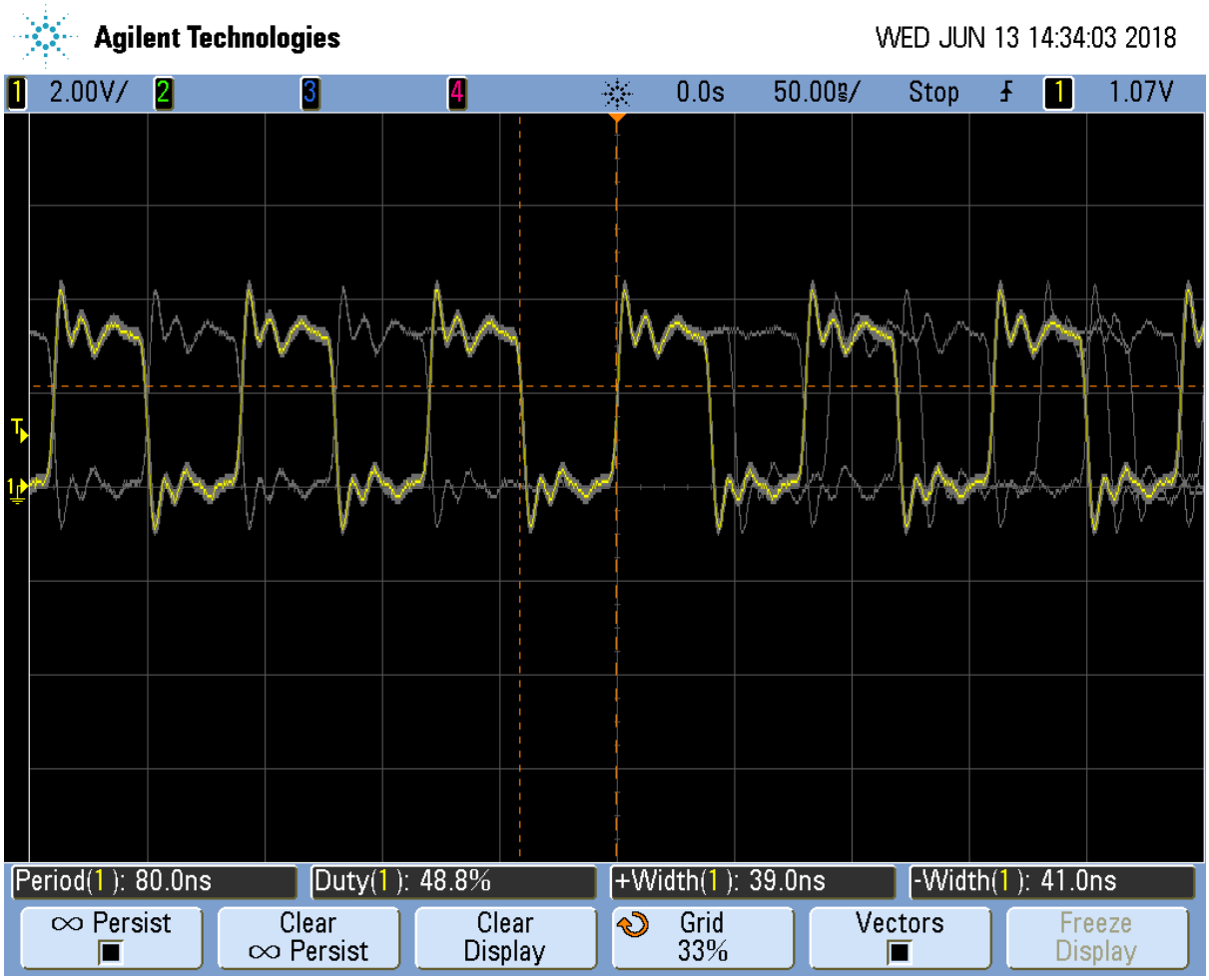


Fig. 15.178: PWM with jitter

The PRU is still as solid as before in its timing, but now it's going through the OCP interface. This interface is shared with other parts of the system, therefore the sometimes the PRU must wait for the other parts to finish. When this happens the pulse width is a bit longer than usual thus adding jitter to the output.

For many applications a few nanoseconds of jitter is unimportant and this GPIO interface can be used. If your application needs better timing, use the `pass: [___]R30` interface.

Configuring for UIO Instead of RemoteProc

Problem You have some legacy PRU code that uses UIO instead of remoteproc and you want to switch to UIO.

Solution Edit `/boot/uEnt.txt` and search for `uio`. I find

```
###pru_uio (4.4.x-ti, 4.9.x-ti, 4.14.x-ti & mainline/bone kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
```

Uncomment the `uboot` line. Look for other lines with `uboot_overlay_pru=` and be sure they are commented out.

Reboot your Bone.

```
bone$ sudo reboot
```

Check that UIO is running.

```
bone$ lsmod | grep uio
uio_pruss              16384  0
uio_pdrv_genirq        16384  0
uio                    20480  2 uio_pruss,uio_pdrv_genirq
```

You are now ready to run the legacy PRU code.

Converting pasm Assembly Code to clpru

Problem You have some legacy assembly code written in pasm and it won't assemble with clpru.

Solution Generally there is a simple mapping from pasm to clpru. pasm vs. clpru notes what needs to be changed. I have a less complete version on my eLinux.org site.

Discussion The clpru assembly can be found in PRU Assembly Language Tools.

15.2.7 More Performance

So far in all our examples we've been able to meet our timing goals by writing our code in the C programming language. The C compiler does a surprisingly good job at generating code, most the time. However there are times when very precise timing is needed and the compiler isn't doing it.

At these times you need to write in assembly language. This chapter introduces the PRU assembler and shows how to call assembly code from C. Detailing on how to program in assembly are beyond the scope of this text.

The following are resources used in this chapter.

Note: *Resources*

- PRU Optimizing C/C++ Compiler, v2.2, User's Guide
- PRU Assembly Language Tools User's Guide
- PRU Assembly Instruction User Guide

Calling Assembly from C

Problem You have some C code and you want to call an assembly language routine from it.

Solution You need to do two things, write the assembler file and modify the `Makefile` to include it. For example, let's write our own `my_delay_cycles` routine in in assembly. The intrinsic `pass:[__]delay_cycles` must be passed a compile time constant. Our new `delay_cycles` can take a runtime delay value.

delay-test.pru0.c is much like our other c code, but on line 10 we declare `my_delay_cycles` and then on lines 24 and 26 we'll call it with an argument of 1.

Listing 15.113: delay-test.pru0.c

```
1 // Shows how to call an assembly routine with one parameter
2 #include <stdint.h>
3 #include <pru_cfg.h>
4 #include "resource_table_empty.h"
5 #include "prugpio.h"
```

(continues on next page)

(continued from previous page)

```

// The function is defined in delay.asm in same dir
// We just need to add a declaration here, the definition can be
// separately linked
extern void my_delay_cycles(uint32_t);

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t gpio = P9_31;         // Select which pin to toggle.;

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        while(1) {
                __R30 |= gpio;                 // Set the GPIO pin to 1
                my_delay_cycles(1);
                __R30 &= ~gpio;                 // Clear the GPIO pin
                my_delay_cycles(1);
        }
}
```

delay-test.pru0.c

delay.pru0.asm is the assembly code.

Listing 15.114: delay.pru0.asm

```
; This is an example of how to call an assembly routine from C.
;        Mark A. Yoder, 9-July-2018
        .global my_delay_cycles
my_delay_cycles:
delay:
        sub                r14,    r14, 1                ; The first argument␣
↪is passed in r14
        qbne        delay, r14, 0

        jmp                r3.w2                         ; r3 contains the␣
↪return address
```

delay.pru0.asm

The Makefile has one addition that needs to be made to compile both *delay-test.pru0.c* and *delay.pru0.asm*. If you look in the local Makefile you'll see:

Listing 15.115: Makefile

```
include /opt/source/pru-cookbook-code/common/Makefile
```

Makefile

This Makefle includes a common Makefile at /opt/source/pru-cookbook-code/common/Makefile, this the Makefile you need to edit. Edit /opt/source/pru-cookbook-code/common/Makefile and go to line 195.

```
$(GEN_DIR)/%.out: $(GEN_DIR)/%.o *$(GEN_DIR)/$(TARGETasm).o*
  @mkdir -p $(GEN_DIR)
  @echo 'LD   $^'
  $(eval $(call target-to-proc,$@))
  $(eval $(call proc-to-build-vars,$@))
  @$(LD) $@ $^ $(LDFLAGS)
```

Add `*(GEN_DIR)/$(TARGETasm).o*` as shown in bold above. You will want to remove this addition once you are done with this example since it will break the other examples.

The following will compile and run everything.

```
bone$ config-pin P9_31 pruout
bone$ make TARGET=delay-test.pru0 TARGETasm=delay.pru0
/opt/source/pru-cookbook-code/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_
↪Black,TARGET=delay-test.pru0
-    Stopping PRU 0
-     copying firmware file /tmp/vsx-examples/delay-test.pru0.out to /lib/
↪firmware/am335x-pru0-fw
write_init_pins.sh
-    Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR = /sys/class/remoteproc/remoteproc1
```

The resulting output is shown in *Output of my_delay_cycles()*.

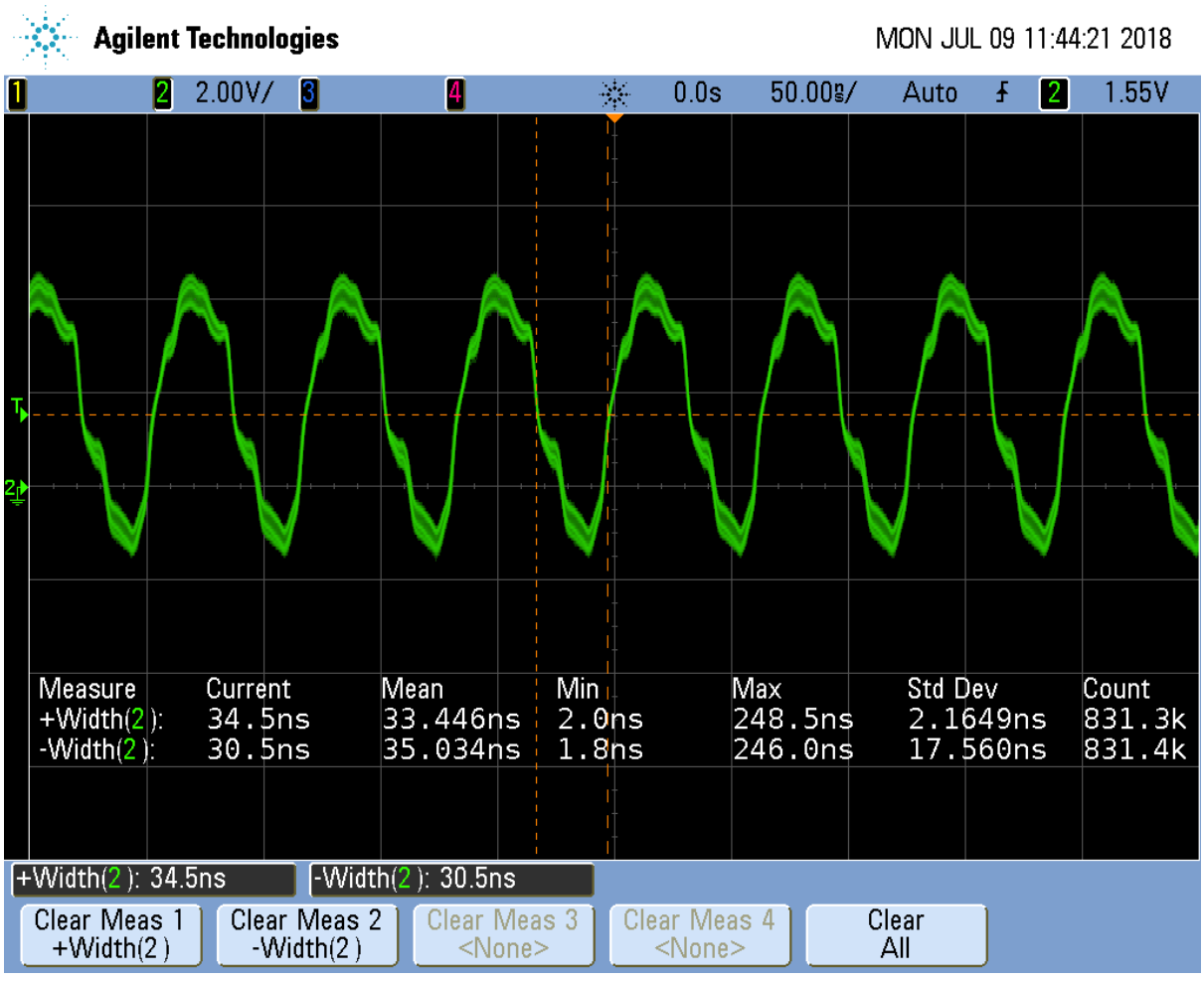


Fig. 15.179: Output of my_delay_cycles()

Notice the on time is about 35ns and the off time is 30ns.

Discussion There is much to explain here. Let's start with *delay.pru0.asm*.

Table 15.29: Line-by-line of delay.pru0.asm

Line	Explanation
3	Declare *my_delay_cycles* to be global so the linker can find it.
4	Label the starting point for *my_delay_cycles*.
5	Label for our delay loop.
6	The first argument is passed in register `r14`. Page 111 of PRU Optimizing C/C++ Compiler, v2.2, User's Guide gives the argument passing convention. Registers `r14` to `r29` are used to pass arguments, if there are more arguments, the argument stack (`r4`) is used. The other register conventions are found on page 108. Here we subtract 1 from `r14` and save it back into `r14`.
7	*qbne* is a quick branch if not equal.
9	Once we've delayed enough we drop through the quick branch and hit the jump. The upper bits of register *r3* has the return address, therefore we return to the c code.

Output of my_delay_cycles() shows the **on** time is 35ns and the off time is 30ns. With 5ns/cycle this gives 7 cycles on and 6 off. These times make sense because each instruction takes a cycle and you have, set `R30`, jump to `my_delay_cycles`, `sub`, `qbne`, `jmp`. Plus the instruction (not seen) that initializes *r14* to the passed value. That's a total of six instructions. The extra instruction is the branch at the bottom of the `while` loop.

Returning a Value from Assembly

Problem Your assembly code needs to return a value.

Solution `R14` is how the return value is passed back. *delay-test2.pru0.c* shows the c code.

Listing 15.116: delay-test2.pru0.c

```
// Shows how to call an assembly routine with a return value
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"

#define         TEST        100

// The function is defined in delay.asm in same dir
// We just need to add a declaration here, the definition can be
// separately linked
extern uint32_t my_delay_cycles(uint32_t);

uint32_t ret;

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t gpio = P9_31;         // Select which pin to toggle.;

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        while(1) {
                __R30 |= gpio;                 // Set the GPIO pin to 1
                ret = my_delay_cycles(1);
                __R30 &= ~gpio;                 // Clear the GPIO pin
                ret = my_delay_cycles(1);
        }
}
```

delay-test2.pru0.c

delay2.pru0.asm is the assembly code.

Listing 15.117: delay2.pru0.asm

```
; This is an example of how to call an assembly routine from C with a return value.
;        Mark A. Yoder, 9-July-2018

        .cdecls "delay-test2.pru0.c"

        .global my_delay_cycles
my_delay_cycles:
delay:
        sub                 r14,    r14, 1                  ; The first argument is passed in r14
        qbne        delay, r14, 0

        ldi                 r14, TEST                   ; TEST is defined in delay-test2.c
                                                            ; r14 is the return register

        jmp                 r3.w2                       ; r3 contains the return address
```

delay2.pru0.asm

An additional feature is shown in line 4 of *delay2.pru0.asm*. The `.cdecls "delay-test2.pru0.c"` says to include any defines from `delay-test2.pru0.c` In this example, line 6 of *delay-test2.pru0.c #defines* TEST and line 12 of *delay2.pru0.asm* reference it.

Using the Built-In Counter for Timing

Problem I want to count how many cycles my routine takes.

Solution Each PRU has a `CYCLE` register which counts the number of cycles since the PRU was enabled. They also have a `STALL` register that counts how many times the PRU stalled fetching an instruction. *cycle.pru0.c - Code to count cycles.* shows they are used.

Listing 15.118: cycle.pru0.c - Code to count cycles.

```
// Access the CYCLE and STALL registers
#include <stdint.h>
#include <pru_cfg.h>
#include <pru_ctrl.h>
#include "resource_table_empty.h"
#include "prugpio.h"

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t gpio = P9_31;          // Select which pin to toggle.;

        // These will be kept in registers and never written to DRAM
        uint32_t cycle, stall;

        // Clear SYSCFG[STANDBY_INIT] to enable OCP master port
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

```

(continues on next page)

(continued from previous page)

```
21          PRU0_CTRL.CTRL_bit.CTR_EN = 1;          // Enable cycle counter
22
23          __R30 |= gpio;                                  // Set the GPIO pin to␣
↪1
24          // Reset cycle counter, cycle is on the right side to force the␣
↪compiler
25          // to put it in it's own register
26          PRU0_CTRL.CYCLE = cycle;
27          __R30 &= ~gpio;                                  // Clear the GPIO pin
28          cycle = PRU0_CTRL.CYCLE;          // Read cycle and store in a register
29          stall = PRU0_CTRL.STALL;          // Ditto for stall
30
31          __halt();
32  }
```

cycle.pru0.c

Discussion The code is mostly the same as other examples. `cycle` and `stall` end up in registers which we can read using prudebug. *Line-by-line for cycle.pru0.c* is the Line-by-line.

Table 15.30: Line-by-line for cycle.pru0.c

Line	Explanation
4	Include needed to reference *CYCLE* and *STALL*.
16	Declaring *cycle* and *stall*. The compiler will optimize these and just keep them in registers. We'll have to look at the *cycle.pru0.lst* file to see where they are stored.
21	Enables *CYCLE*.
26	Reset *CYCLE*. It ignores the value assigned to it and always sets it to 0. *cycle* is on the right hand side to make the compiler give it its own register.
28, 29	Reads the *CYCLE* and *STALL* values into registers.

You can see where `cycle` and `stall` are stored by looking into */tmp/vsx-examples/cycle.pru0.lst Lines 113..119*.

Listing 15.119: /tmp/vsx-examples/cycle.pru0.lst Lines 113..119

```
113     102         .dwpsn  file "cycle.pru0.c",line 23,column 2,is_stmt,isa 0
114     103;----------------------------------------------------------------------
↪--
115     104;  23 | PRU0_CTRL.CTRL_bit.CTR_EN = 1;  // Enable cycle counter      ␣
↪
116     105;----------------------------------------------------------------------
↪--
117     106 0000000c 200080240002C0          LDI32     r0, 0x00022000         ;␣
↪[ALU_PRU] |23| $O$C1
118     107 00000014 000000F1002081          LBBO      &r1, r0, 0, 4          ;␣
↪[ALU_PRU] |23|
119     108 00000018 0000001F03E1E1          SET       r1, r1, 0x00000003     ;␣
↪[ALU_PRU] |23|
```

cycle.pru0.lst

Here the `LDI32` instruction loads the address `0x22000` into `r0`. This is the offset to the `CTRL` registers. Later in the file we see */tmp/vsx-examples/cycle.pru0.lst Lines 146..152*.

Listing 15.120: /tmp/vsx-examples/cycle.pru0.lst Lines 146..152

```
146     129;----------------------------------------------------------------------
↪--
147     130;  30 | cycle = PRU0_CTRL.CYCLE;          // Read cycle and store in a␣
↪register
```

(continues on next page)

(continued from previous page)

```
148     131;------------------------------------------------------------------
    ↪--
149     132 0000002c 000000F10C2081          LBBO      &r1, r0, 12, 4        ;␣
    ↪[ALU_PRU] |30| $O$C1
150     133        .dwpsn  file "cycle.pru0.c",line 31,column 2,is_stmt,isa 0
151     134;------------------------------------------------------------------
    ↪--
152     135;  31 | stall = PRU0_CTRL.STALL;        // Ditto for stall          ␣
    ↪
```

cycle.pru0.lst

The first LBBO takes the contents of r0 and adds the offset 12 to it and copies 4 bytes into r1. This points to CYCLE, so r1 has the contents of CYCLE.

The second LBBO does the same, but with offset 16, which points to STALL, thus STALL is now in r0.

Now fire up **prudebug** and look at those registers.

```
bone$ sudo prudebug
PRU0> r
r
r
Register info for PRU0
    Control register: 0x00000009
      Reset PC:0x0000  STOPPED, FREE_RUN, COUNTER_ENABLED, NOT_SLEEPING,␣
    ↪PROC_DISABLED

    Program counter: 0x0012
      Current instruction: HALT

    R00: *0x00000005*    R08: 0x00000200    R16: 0x000003c6    R24:␣
    ↪0x00110210
    R01: *0x00000003*    R09: 0x00000000    R17: 0x00000000    R25:␣
    ↪0x00000000
    R02: 0x000000fc    R10: 0xfff4ea57    R18: 0x000003e6    R26: 0x6e616843
    R03: 0x0004272c    R11: 0x5fac6373    R19: 0x30203020    R27: 0x206c656e
    R04: 0xffffffff    R12: 0x59bfeafc    R20: 0x0000000a    R28: 0x00003033
    R05: 0x00000007    R13: 0xa4c19eaf    R21: 0x00757270    R29: 0x02100000
    R06: 0xefd30a00    R14: 0x00000005    R22: 0x0000001e    R30: 0xa03f9990
    R07: 0x00020024    R15: 0x00000003    R23: 0x00000000    R31: 0x00000000
```

So cycle is 3 and stall is 5. It must be one cycle to clear the GPIO and 2 cycles to read the CYCLE register and save it in the register. It's interesting there are 5 stall cycles.

If you switch the order of lines 30 and 31 you'll see cycle is 7 and stall is 2. cycle now includes the time needed to read stall and stall no longer includes the time to read cycle.

Xout and Xin - Transferring Between PRUs

Problem I need to transfer data between PRUs quickly.

Solution The pass:[__]xout() and pass:[__]xin() intrinsics are able to transfer up to 30 registers between PRU 0 and PRU 1 quickly. *xout.pru0.c* shows how xout() running on PRU 0 transfers six registers to PRU 1.

Listing 15.121: xout.pru0.c

```
1 // From: http://git.ti.com/pru-software-support-package/pru-software-support-
   ↪package/trees/master/examples/am335x/PRU_Direct_Connect0
2 #include <stdint.h>
```

(continues on next page)

(continued from previous page)

```
#include <pru_intc.h>
#include "resource_table_pru0.h"

volatile register uint32_t __R30;
volatile register uint32_t __R31;

typedef struct {
        uint32_t reg5;
        uint32_t reg6;
        uint32_t reg7;
        uint32_t reg8;
        uint32_t reg9;
        uint32_t reg10;
} bufferData;

bufferData dmemBuf;

/* PRU-to-ARM interrupt */
#define PRU1_PRU0_INTERRUPT (18)
#define PRU0_ARM_INTERRUPT (19+16)

void main(void)
{
        /* Clear the status of all interrupts */
        CT_INTC.SECR0 = 0xFFFFFFFF;
        CT_INTC.SECR1 = 0xFFFFFFFF;

        /* Load the buffer with default values to transfer */
        dmemBuf.reg5 = 0xDEADBEEF;
        dmemBuf.reg6 = 0xAAAAAAAA;
        dmemBuf.reg7 = 0x12345678;
        dmemBuf.reg8 = 0xBBBBBBBB;
        dmemBuf.reg9 = 0x87654321;
        dmemBuf.reg10 = 0xCCCCCCCC;

        /* Poll until R31.30 (PRU0 interrupt) is set
         * This signals PRU1 is initialized */
        while ((__R31 & (1<<30)) == 0) {
        }

        /* XFR registers R5-R10 from PRU0 to PRU1 */
        /* 14 is the device_id that signifies a PRU to PRU transfer */
        __xout(14, 5, 0, dmemBuf);

        /* Clear the status of the interrupt */
        CT_INTC.SICR = PRU1_PRU0_INTERRUPT;

        /* Halt the PRU core */
        __halt();
}
```

xout.pru0.c

PRU 1 waits at line 41 until PRU 0 signals it. *xin.pru1.c* sends an interrupt to PRU 0 and waits for it to send the data.

Listing 15.122: xin.pru1.c

```
// From: http://git.ti.com/pru-software-support-package/pru-software-support-
↪package/trees/master/examples/am335x/PRU_Direct_Connect1
#include <stdint.h>
#include "resource_table_empty.h"
```

(continues on next page)

(continued from previous page)

```

volatile register uint32_t __R30;
volatile register uint32_t __R31;

typedef struct {
        uint32_t reg5;
        uint32_t reg6;
        uint32_t reg7;
        uint32_t reg8;
        uint32_t reg9;
        uint32_t reg10;
} bufferData;

bufferData dmemBuf;

/* PRU-to-ARM interrupt */
#define PRU1_PRU0_INTERRUPT (18)
#define PRU1_ARM_INTERRUPT (20+16)

void main(void)
{
        /* Let PRU0 know that I am awake */
        __R31 = PRU1_PRU0_INTERRUPT+16;

        /* XFR registers R5-R10 from PRU0 to PRU1 */
        /* 14 is the device_id that signifies a PRU to PRU transfer */
        __xin(14, 5, 0, dmemBuf);

        /* Halt the PRU core */
        __halt();
}
```

xin.pru1.c

Use `prudebug` to see registers R5-R10 are transferred from PRU 0 to PRU 1.

```
PRU0> r
Register info for PRU0
    Control register: 0x00000001
      Reset PC:0x0000  STOPPED, FREE_RUN, COUNTER_DISABLED, NOT_SLEEPING,␣
↪PROC_DISABLED

    Program counter: 0x0026
      Current instruction: HALT

    R00: 0x00000012    *R08: 0xbbbbbbbb*    R16: 0x000003c6    R24:␣
↪0x00110210
    R01: 0x00020000    *R09: 0x87654321*    R17: 0x00000000    R25:␣
↪0x00000000
    R02: 0x000000e4    *R10: 0xcccccccc*    R18: 0x000003e6    R26:␣
↪0x6e616843
    R03: 0x0004272c    R11: 0x5fac6373    R19: 0x30203020    R27: 0x206c656e
    R04: 0xffffffff    R12: 0x59bfeafc    R20: 0x0000000a    R28: 0x00003033
    *R05: 0xdeadbeef*    R13: 0xa4c19eaf    R21: 0x00757270    R29:␣
↪0x02100000
    *R06: 0xaaaaaaaa*    R14: 0x00000005    R22: 0x0000001e    R30:␣
↪0xa03f9990
    *R07: 0x12345678*    R15: 0x00000003    R23: 0x00000000    R31:␣
↪0x00000000

PRU0> *pru 1*
pru 1
```

(continues on next page)

(continued from previous page)

```
Active PRU is PRU1.

PRU1> *r*
r
Register info for PRU1
    Control register: 0x00000001
      Reset PC:0x0000  STOPPED, FREE_RUN, COUNTER_DISABLED, NOT_SLEEPING,␣
↪PROC_DISABLED

    Program counter: 0x000b
      Current instruction: HALT

    R00: 0x00000100    *R08: 0xbbbbbbbb*    R16: 0xe9da228b    R24:␣
↪0x28113189
    R01: 0xe48cdb1f    *R09: 0x87654321*    R17: 0x66621777    R25:␣
↪0xddd29ab1
    R02: 0x000000e4    *R10: 0xcccccccc*    R18: 0x661f83ea    R26:␣
↪0xcf1cd4a5
    R03: 0x0004db97    R11: 0xdec387d5    R19: 0xa85adb78    R27: 0x70af2d02
    R04: 0xa90e496f    R12: 0xbeac3878    R20: 0x048fff22    R28: 0x7465f5f0
    *R05: 0xdeadbeef*    R13: 0x5777b488    R21: 0xa32977c7    R29:␣
↪0xae96b530
    *R06: 0xaaaaaaaa*    R14: 0xffa60550    R22: 0x99fb123e    R30:␣
↪0x52c42a0d
    *R07: 0x12345678*    R15: 0xdeb2142d    R23: 0xa353129d    R31:␣
↪0x00000000
```

Discussion *xout.pru0.c Line-by-line* shows the line-by-line for `xout.pru0.c`

Table 15.31: xout.pru0.c Line-by-line

Line	Explanation
4	A different resource so PRU 0 can receive a signal from PRU 1.
9-16	`dmemBuf` holds the data to be sent to PRU 1. Each will be transferred to its corresponding register by `xout()`.
21-22	Define the interrupts we're using.
27-28	Clear the interrupts.
31-36	Initialize dmemBuf with easy to recognize values.
40	Wait for PRU 1 to signal.
45	`pass:[__]xout()` does a direct transfer to PRU 1. Page 92 of PRU Optimizing C/C++ Compiler, v2.2, User's Guide shows how to use *xout()*. The first argument, 14, says to do a direct transfer to PRU 1. If the first argument is 10, 11 or 12, the data is transferred to one of three scratchpad memories that PRU 1 can access later. The second argument, 5, says to start transferring with register `r5` and use as many registers as needed to transfer all of `dmemBuf`. The third argument, 0, says to not use remapping. (See the User's Guide for details.) The final argument is the data to be transferred.
48	Clear the interrupt so it can go again.

xin.pru1.c Line-by-line shows the line-by-line for `xin.pru1.c`.

Table 15.32: xin.pru1.c Line-by-line

Line	Explanation
8-15	Place to put the received data.
26	Signal PRU 0
30	Receive the data. The arguments are the same as *xout()*, 14 says to get the data directly from PRU 0. 5 says to start with register *r5*. *dmemBuf* is where to put the data.

If you really need speed, considering using `pass:[__]xout()` and `pass:[__]xin()` in assembly.

Copyright

Listing 15.123: copyright.c

```
/*
 * Copyright (C) 2015 Texas Instruments Incorporated - http://www.ti.com/
 *
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *        * Redistributions of source code must retain the above copyright
 *          notice, this list of conditions and the following disclaimer.
 *
 *        * Redistributions in binary form must reproduce the above copyright
 *          notice, this list of conditions and the following disclaimer in␣
↪the
 *          documentation and/or other materials provided with the
 *          distribution.
 *
 *        * Neither the name of Texas Instruments Incorporated nor the names␣
↪of
 *          its contributors may be used to endorse or promote products␣
↪derived
 *          from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

```

copyright.c

15.2.8 Moving to the BeagleBone AI

So far all our examples have focussed mostly on the BeagleBone Black and PocketBeagle. These are both based on the am335x chip. The new kid on the block is the BeagleBone AI which is based on the am5729. The new chip brings with it new capabilities one of which is four PRUs. This chapter details what changes when moving from two to four PRUs.

The following are resources used in this chapter.

Note: *Resources*

- AM572x Technical Reference Manual (AI)
- BeagleBone AI PRU pins

Moving from two to four PRUs

Problem You have code that works on the am335x PRUs and you want to move it to the am5729 on the AI.

Solution Things to consider when moving to the AI are:

- Which pins are you going to use
- Which PRU are you going to run on

Knowing which pins to use impacts the PRU you'll use.

Discission The various System Reference Manuals (SRM's) list which pins go to the PRUs. Here the tables are combined into one to make it easier to see what goes where.

Table 15.33: Mapping bit positions to pin names

PRU 0	Bit 0	Black pin P9_31	AI PRU1 pin	AI PRU2 pin P8_44	Pocket pin P1.36
0	1	P9_29		P8_41	P1.33
0	2	P9_30		P8_42/P8_21	P2.32
0	3	P9_28	P8_12	P8_39/P8_20	P2.30
0	4	P9_92	P8_11	P8_40/P8_25	P1.31
0	5	P9_27	P9_15	P8_37/P8_24	P2.34
0	6	P9_91		P8_38/P8_5	P2.28
0	7	P9_25		P8_36/P8_6	P1.29
0	8			P8_34/P8_23	
0	9			P8_35/P8_22	
0	19			P8_33/P8_3	
0	11			P8_31/P8_4	
0	12			P8_32	
0	13			P8_45	
0	14	P8_12(out) P8_16(in)|		P9_11	P2.24
0	15	P8_11(out) P8_15(in)|		P8_17/P9_13	P2.33
0	16	P9_41(in) P9_26(in)|		P8_27	
0	17		P9_26	P8_28	
0	18			P8_29	
0	19			P8_30	
0	20			P8_46/P8_8	
1	0	P8_45		P8_32	
1	1	P8_46	*P9_20*		
1	2	P8_43	*P9_19*		
1	3	P8_44	P9_41		
1	4	P8_41			
1	5	P8_42	*P8_18*	P9_25	
1	6	P8_39	*P8_19*	P8_9	
1	7	P8_40	*P8_13*	P9_31	
1	8	P8_27		P9_18	P2.35
1	9	P8_29	P8_14	P9_17	P2.01
1	10	P8_28	P9_42	P9_31	P1.35
1	11	P8_30	P9_27	P9_29	P1.04
1	12	P8_21		P9_30	
1	13	P8_20		P9_26	

continues on next page

Table 15.33 – continued from previous page

1	14		P9_14	P9_42	P1.32	
1	15		*P9_16*	P8_10	P1.30	
1	16	P9_26(in)	*P8_15*	P8_7		
1	17		*P8_26*	P8_27		
1	18		*P8_16*	P8_45		
1	19			P8_46		
1	19			P8_43		

The pins in *bold* are already configured as pru pins. See *Seeing how pins are configured* to see what's currently configured as what. See *Configuring pins on the AI via device trees* to configure pins.

Seeing how pins are configured

Problem You want to know how the pins are currently configured.

Solution The `show-pins.pl` command does what you want, but you have to set it up first.

```
bone$ cd ~/bin
bone$ ln -s /opt/scripts/device/bone/show-pins.pl .
```

This creates a symbolic link to the `show-pins.pl` command that is rather hidden away. The link is put in the `bin` directory which is in the default command `$PATH`. Now you can run `show-pins.pl` from anywhere.

```
bone$ *show-pins.pl*
P9.19a                    16   R6 7 fast rx  up  i2c4_scl
P9.20a                    17   T9 7 fast rx  up  i2c4_sda
P8.35b                    57  AD9 e fast    down gpio3_0
P8.33b                    58  AF9 e fast    down gpio3_1
...
```

Here you see `P9.19a` and `P9.20a` are configured for i2c with pull up resistors. The `P8` pins are configured as gpio with pull down resistors. They are both on gpio port 3. `P8.35b` is bit 0 while `P8.33b` is bit 1. You can find which direction they are set by using `gpioinfo` and the chip number. Unfortunately you subtract one from the port number to get the chip number. So `P8.35b` is on chip number 2.

```
bone$ *gpioinfo 2*
        line   0:      unnamed       unused   *input*  active-high
        line   1:      unnamed       unused   *input*  active-high
        line   2:      unnamed       unused   input  active-high
        line   3:      unnamed       unused   input  active-high
        line   4:      unnamed       unused   input  active-high
...
```

Here we see both (lines 0 and 1) are set to input.

Adding `-v` gives more details.

```
bone$ *show-pins.pl -v*
...
sysboot 14                14   H2 f fast    down sysboot14
sysboot 15                15   H3 f fast    down sysboot15
P9.19a                    16   R6 7 fast rx  up  i2c4_scl
P9.20a                    17   T9 7 fast rx  up  i2c4_sda
                                                 18   T6 f fast    down␣
↪Driver off
                                                 19   T7 f fast    down␣
↪Driver off
bluetooth in              20   P6 8 fast rx      uart6_rxd       ␣
↪mmc@480d1000 (wifibt_extra_pins_default)
bluetooth out             21   R9 8 fast rx      uart6_txd       ␣
↪mmc@480d1000 (wifibt_extra_pins_default)
...
```

The best way to use `show-pins.pl` is with `grep`. To see all the pru pins try:

```
bone$ *show-pins.pl  | grep -i pru | sort*
P8.13                   100   D3 c fast rx      pr1_pru1_gpi7
P8.15b                  109   A3 d fast    down pr1_pru1_gpo16
P8.16                   111   B4 d fast    down pr1_pru1_gpo18
```

(continues on next page)

(continued from previous page)

```
P8.18                    98   F5 c fast rx      pr1_pru1_gpi5
P8.19                    99   E6 c fast rx      pr1_pru1_gpi6
P8.26                   110   B3 d fast    down pr1_pru1_gpo17
P9.16                   108   C5 d fast    down pr1_pru1_gpo15
P9.19b                   95   F4 c fast rx  up   pr1_pru1_gpi2
P9.20b                   94   D2 c fast rx  up   pr1_pru1_gpi1
```

Here we have nine pins configured for the PRU registers `R30` and `R31`. Five are input pins and four are out.

Configuring pins on the AI via device trees

Problem I want to configure another pin for the PRU, but I get an error.

```
bone$ *config-pin P9_31 pruout*
ERROR: open() for /sys/devices/platform/ocp/ocp:P9_31_pinmux/state failed,␣
↪No such file or directory
```

Solution The pins on the AI must be configure at boot time and therefor cannot be configured with `config-pin`. Instead you must edit the device tree.

Discission Suppose you want to make `P9_31` a PRU output pin. First go to the am5729 System Reference Manual and look up `P9_31`.

Tip: The BeagleBone AI PRU pins table may be easier to use.

`P9_31` appears twice, as `P9_31a` and `P9_31b`. Either should work, let's pick `P9_31a`.

Warning: When you have two internal pins attached to the same header (either P8 or P9) make sure only one is configured as an output. If both are outputs, you could damage the AI.

We see that when `P9_31a` is set to `MODE13` it will be a PRU **out** pin. `MODE12` makes it a PRU **in** pin. It appears at bit 10 on PRU2_1.

Next, find which kernel you are running.

```
bone$ uname -a
Linux ai 4.14.108-ti-r131 #1buster SMP PREEMPT Tue Mar 24 19:18:36 UTC 2020␣
↪armv7l GNU/Linux
```

I'm running the 4.14 version. Now look in `/opt/source` for your kernel.

```
bone$ cd /opt/source/
bone$ ls
adafruit-beaglebone-io-python  dtb-5.4-ti       rcpy
BBIOConfig                     librobotcontrol  u-boot_v2019.04
bb.org-overlays                list.txt         u-boot_v2019.07-rc4
*dtb-4.14-ti*                    pyctrl
dtb-4.19-ti                    py-uio
```

`am5729-beagleboneai.dts` is the file we need to edit. Search for `P9_31`. You'll see:

```
1 DRA7XX_CORE_IOPAD(0x36DC, MUX_MODE14) // B13: P9.30: mcasp1_axr10.off //
2 DRA7XX_CORE_IOPAD(0x36D4, *MUX_MODE13*) // B12: *P9.31a*: mcasp1_axr8.off //
3 DRA7XX_CORE_IOPAD(0x36A4, MUX_MODE14) // C14: P9.31b: mcasp1_aclkx.off //
```

Change the MUX_MODE14 to MUX_MODE13 for output, or MUX_MODE12 for input.

Compile and install. The first time will take a while since it recompiles all the dts files.

```
bone$ make
...
DTC     src/arm/am335x-sl50.dtb
DTC     src/arm/am5729-beagleboneai.dtb
DTC     src/arm/am335x-nano.dtb
...
bone$ sudo make install
...
'src/arm/am5729-beagleboneai.dtb' -> '/boot/dtbs/4.14.108-ti-r131/am5729-
↪beagleboneai.dtb'
...
bone$ reboot
...
bone$ *show-pins.pl -v | sort | grep -i pru*
P8.13                   100   D3 c fast rx      pr1_pru1_gpi7
P8.15b                  109   A3 d fast    down pr1_pru1_gpo16
P8.16                   111   B4 d fast    down pr1_pru1_gpo18
P8.18                    98   F5 c fast rx      pr1_pru1_gpi5
P8.19                    99   E6 c fast rx      pr1_pru1_gpi6
P8.26                   110   B3 d fast    down pr1_pru1_gpo17
P9.16                   108   C5 d fast    down pr1_pru1_gpo15
P9.19b                   95   F4 c fast rx  up  pr1_pru1_gpi2
P9.20b                   94   D2 c fast rx  up  pr1_pru1_gpi1
P9.31a                  181  B12 d fast    down pr2_pru1_gpo10
```

There it is. *P9_31* is now a PRU output pin on PRU1_0, bit 3.

Using the PRU pins

Problem Once I have the PRU pins configured on the AI how do I use them?

Solution In *Configuring pins on the AI via device trees* we configured P9_31a to be a PRU pin. show-pins.pl showed that it appears at pr2_pru1_gpo10, which means pru2_1 accesses it using bit 10 of register R30.

Discission It's easy to modify the pwm example from *PWM Generator* to use this pin. First copy the example you want to modify to pwm1.pru2_1.c. The pru2_1 in the file name tells the Makefile to run the code on pru2_1. *pwm1.pru2_1.c* shows the adapted code.

Listing 15.124: pwm1.pru2_1.c

```
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"

#define P9_31 (0x1<<10)

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
        uint32_t gpio = P9_31;          // Select which pin to toggle.;

        /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
```

(continues on next page)

(continued from previous page)

```
        CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

        while(1) {
                __R30 |= gpio;                  // Set the GPIO pin to 1
                __delay_cycles(100000000);
                __R30 &= ~gpio;                  // Clear the GPIO pin
                __delay_cycles(100000000);
        }
}
```

`pwm1.pru2_1.c`

One line 6 `P9_31` is defined as `(0x1:ref:`10)`, which means shift `1` over by 10 bits. That's the only change needed. Copy the local Makefile to the same directory and compile and run.

```
bone$ make TARGET=pwm1.pru2_1
```

Attach an LED to `P9_31` and it should be blinking.

15.2.9 PRU Projects

Users of TI processors with PRU-ICSS have created application for many different uses. A list of a few are shared below. For additional support resources, software and documentation visit the PRU-ICSS wiki.

LEDscape

Description: BeagleBone Black cape and firmware for driving a large number of WS281x LED strips.

Type: Code Library Documentation and example projects.

References:

- https://github.com/osresearch/LEDscape http://trmm.net/LEDscape

LDGraphy

Description: Laser direct lithography for printing PCBs.

Type: Code Library and example project.

References:

- https://github.com/hzeller/ldgraphy/blob/master/README.md

PRdUino

Description: This is a port of the Energia platform based on the Arduino framework allowing you to use Arduino software libraries on PRU.

Type: Code Library

References:

- https://github.com/lucas-ti/PRdUino

DMX Lighting

Description: Controlling professional lighting systems

Type: Project Tutorial Code Library

References:

- https://beagleboard.org/CapeContest/entries/BeagleBone+DMX+Cape/
- https://web.archive.org/web/20130921033304/blog.boxysean.com/2012/08/12/first-steps-with-the-beaglebone-pru/
- https://github.com/boxysean/beaglebone-DMX

Interacto

Description: A cape making BeagleBone interactive with a triple-axis accelerometer, gyroscope and magnetometer plus a 640 x 480/30 fps camera. All sensors are digital and communicate via I²C to the BeagleBone. The camera frames are captured using the PRU-ICSS. The sensors on this cape give hobbyists and students a starting point to easily build robots and flying drones.

Type: Project 1 Project 2 Code Library

References:

- https://beagleboard.org/CapeContest/entries/Interacto/
- https://web.archive.org/web/20130507141634/http://www.hitchhikeree.org:80/beaglebone_capes/interacto/
- https://github.com/cclark2/interacto_bbone_cape

Replicape: 3D Printer

Description: Replicape is a high end 3D-printer electronics package in the form of a Cape that can be placed on a BeagleBone Black. It has five high power stepper motors with cool running MosFets and it has been designed to fit in small spaces without active cooling. For a Replicape Daemon that processes G-code, see the Redeem Project

Type: Project Code Library

References:

- http://www.thing-printer.com/product/replicape/
- https://bitbucket.org/intelligentagent/replicape/

PyPRUSS: Python Library

Description: PyPRUSS is a Python library for programming the PRUs on BeagleBone (Black)

Type: Code Library

References: https://github.com/MuneebMohammed/pypruss

Geiger

Description: The Geiger Cape, created by Matt Ranostay, is a design that measures radiation counts from background and test sources by utilising multiple Geiger tubes. The cape can be used to detect low-level radiation, which is needed in certain industries such as security and medical.

Type: Project 1 Project 2 Code Library

References:

- http://beagleboard.org/CapeContest/entries/Geiger+Cape/
- http://elinux.org/BeagleBone/GeigerCapePrototype

Note: #TODO#: the git repo was taken down

Servo Controller Foosball Table

Description: Used for ball tracking and motor control

Type: Project Tutorial Code Library

References:

- http://www.element14.com/community/community/knode/single-board_computers/next-gen_beaglebone/blog/2013/07/17/hackerspace-challenge–leeds-only-pru-can-make-the-leds-bright
- https://docs.google.com/spreadsheet/pub?key=0AmI_ryMKXUGJdDQ3LXB4X3VBWlpxQTFWbGh6RGJHUEE&output=html
- https://github.com/pbrook/pypruss

Imaging with connected camera

Description: Low resolution imaging ideal for machine vision use-cases, robotics and movement detection

Type: Project Code Library

References:

- http://www.element14.com/community/community/knode/single-board_computers/next-gen_beaglebone/blog/2013/08/18/bbb–imaging-with-a-pru-connected-camera

Computer Numerical Control (CNC) Translator

Description: Smooth stepper motor control; real embedded version of LinuxCNC

Type: Tutorial Tutorial

References:

- http://www.buildlog.net/blog/2013/09/cnc-translator-for-beaglebone/ http://bb-lcnc.blogspot.com/p/machinekit_16.html

Robotic Control

Description: Chubby SpiderBot

Type: Project Code Library Project Reference

References:

- http://www.youtube.com/watch?v=dEes9k7-DYY
- http://www.youtube.com/watch?v=JXyewd98e9Q
- http://www.ti.com/lit/wp/spry235/spry235.pdf

Note: #TODO#: The Chubby1_v1 repo on github.com for user cagdasc was taken down.

Software UART

Description: Soft-UART implementation on the PRU of AM335x

Type: Code Library Reference

References:

- https://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/Foundational_Components/PRU-ICSS/Linux_Drivers/pru-sw-uart.html

Deviant LCD

Description: PRU bit-banged LCD interface @ 240x320

Type: Project Code Library

References:

- http://www.beagleboard.org/CapeContest/entries/DeviantLCD/
- https://github.com/cclark2/deviantlcd_bbone_cape

Nixie tube interface

Description:

Type: Code Library

References:

- https://github.com/mranostay/beagle-nixie

Thermal imaging camera

Description: Thermal camera using BeagleBone Black, a small LCD, and a thermal array sensor

Type: Project Code Library

References:

- https://element14.com/community/community/knode/single-board_computers/next-gen_beaglebone/blog/2013/06/07/bbb–building-a-thermal-imaging-camera

Sine wave generator using PWMs

Description: Simulation of a pulse width modulation

Type: Project Reference Code Library

References:

- http://elinux.org/ECE497_BeagleBone_PRU
- https://github.com/millerap/AM335x_PRU_BeagleBone

Emulated memory interface

Description: ABX loads amovie into the BeagleBone's memory and then launches the memory emulator on the PRU sub-processor of the BeagleBone's ARM AM335x

Type: Project

References:

- https://github.com/lybrown/abx

6502 memory interface

Description: System permitting communication between Linux and 6502 processor

Type: Project Code Library

References:

- http://elinux.org/images/a/ac/What's_Old_Is_New-_A_6502-based_Remote_Processor.pdf
- https://github.com/lybrown/abx

JTAG/Debug

Description: Investigating the fastest way to program using JTAG and provide for debugging facilities built into the BeagleBone.

Type: Project

References:

- http://beagleboard.org/project/PRUJTAG/

High Speed Data Acquistion

Description: Reading data at high speeds

Type: Reference

References:

- http://www.element14.com/community/community/knode/single-board_computers/next-gen_beaglebone/blog/2013/08/04/bbb–high-speed-data-acquisition-and-web-based-ui

Prufh (PRU Forth)

Description: Forth Programming Language and Compiler. It consists of a compiler, the forth system itself, and anoptional program for loading and communicating with the forth code proper.

Type: Compiler

References:

- https://github.com/biocode3D/prufh

VisualPRU

Description: VisualPRU is a minimal browser-based editor and debugger for the BeagleBone PRUs. The app runs from a local server on the BeagleBone.

Type: Editor and Debugger

References:

- https://github.com/mmcdan/visualpru

libpruio

Description: Library for easy configuration and data handling at high speeds. This library can configure and control the devices from single source (no need for further overlays or the device tree compiler)

Type: Documentation

References:

- http://users.freebasic-portal.de/tjf/Projekte/libpruio/doc/html/index.html
- Library http://www.freebasic-portal.de/downloads/fb-on-arm/libpruio-325.html{[}(German)]

BeagleLogic

Description: 100MHz 14channel logic analyzer using both PRUs (one to capture and one to transfer the data)

Type: Project

References:

- http://beaglelogic.net

BeaglePilot

Description: Uses PRUs as part of code for a BeagleBone based autopilot

Type: Code Library

References:

- https://github.com/BeaglePilot/beaglepilot

PRU Speak

Description: Implements BotSpeak, a platform independent interpreter for tools like Labview, on the PRUs

Type: Code Library

References:

- https://github.com/deepakkarki/pruspeak

Chapter 16

Accessories

This section includes tested accessories for BeagleBoard.org hardware and kits.

Note: This documentation is actively evolving and we are always looking for volunteers to test and add new hardware accessories that they have tested. General feedback and contribution is also appreciated. You can checkout these pages to contribuite to the BeagleBoard.org docs project,

1. Docs contribution guide
2. Docs project issue tracker.

Danger: The accessories section only contains a subset of 3rd party products that have been manually used by community members with BeagleBoard.org products. BeagleBoard.org should not be held liable for the functionality of BeagleBoard.org products in association with these 3rd party products in any way possible. This is just a place for people to report their experiences and not a statement of compatibility. BeagleBoard.org approve that these items have at least some aspect of testing by foundation members, though only specific versions and it is up to the manufacturers of those items to maintain compatibility.

Power Supplies



Power source for all your BeagleBoard.org hardware.

Displays



Dedicated, portable, and TV monitors.

Peripherals



Keyboard, mice, and other peripherals.

Cables



USB, debug, HDMI, and other cables.

Cameras





USB and CSI cameras

16.1 Power supplies

Different BeagleBoard products require different power supplies. While BeagleBone Black and other AM335X based boards will be fine with a 5V @ 1A, others, such as BeagleBone AI-64 require aleast 5V @ 3A. You have to either supply the power via USB jack or a matching (center positive) barrel jack as shown in the table below.

Note: The power supply is not supplied with the board.

Table 16.1: BeagleBoard power supplies

Board	Connector	Power	Tested accessories
BeagleBone Black	2.1mm Barrel Jack	5V @ 2A	• Adafruit • Sparkfun • Logic Supply
BeagleBone Black Wireless			
BeagleV-Ahead			
Beaglebone xM			
PocketBeagle	microUSB	5V @ 2A	• AA10A-050A(M)-R • AA10E-050A(M)-R
BeagleBone AI-64	Type-C	5V @ 3A	• AA65M-59FKA-R
BeaglePlay			
BeagleBone AI			
BeagleV-Fire			
BeagleBone Green Gateway	2.1mm Barrel Jack	12V @ 5A	• PSAC60M-120-R
Beaglebone Blue			
BeagleBone X15	2.5mm Barrel Jack	12V @ 5A	• TRG70A120-12E01-Level-V

Tip: Most modern day mobile phone chargers are capable of delivering enough current to power any BeagleBone. You may try using that with a suitable cable before buying any standalone power source for your board. Please ensure the expected output voltage and current capabilities are listed on your power supply before attempting to power your board.

If you plan to use capes or add your own circuitry, a power supply capable of higher current may be required.

Note: USB-C supplies will auto-negotiate the highest power mode that both power supply and BeagleBoard support. In most cases, this will be 5V @ 3A. It is OK to use a higher Wattage USB-C PD supply with a board, but it is recommended to use supplies from well-known manufacturers to avoid supplies that may break the USB-C PD specification.

16.2 Displays

16.2.1 Monitors and Resolutions

Verified Desktop Monitors

The following monitors and resolutions have been tested for operation with the BeagleBone Black. Check here often as we will update the list as we confirm the operation. Let Support know the model and resolution at which it works and we will add it to the list. Only the highest resolution is listed below.

Note: All the monitors most likely will work with AI-64 also but, make sure that you are using active miniDP to HDMI converter becuase passive convertor will not work.

Links are not provided as they change frequently. So, search on the model number to find a source for these.

- Sony Model LMD-2450W 1280 x 1024 @60Hz
- Hitachi Model LD9000T 1280 x 720 @60Hz
- Samsung UN32EH500F 1280 x 1024 @60Hz

- Samsung P2770HD Syncmaster 1280 x 1024 @60Hz
- MAG GML 2226 2200M 1280 x 1024 @60Hz
- ASUS VW266H 1280 x 1024 @60Hz
- Asus VE278H 720 x 480 @60Hz
- HP TSS-23x11 1280 x 1024 @60Hz
- Acer S230HL 1280 x 1024 @60Hz
- Acer S231HL 1280 x 720 @60Hz
- ASUS VH238H 1280 x 720 @50Hz
- Sharp Aquos TV Dell S244OL 1920 x 1080 @60HZ

Unsupported Monitors

LG 37LH30 - did not seem to work, the display didn't even recognize that anything was plugged in at all.

Verified Televisions

The following TVs and resolutions have been tested for operation with the BeagleBone Black. Check here often as we will update the list as we confirm the operation. If you have a TV that works fine, let us know the model and resolution at which it works and we will add it to the list. Only the highest resolution is listed below.

Links are not provided as they change frequently. So, search on the model number to find a source for these.

Vizio E371VL 1280x720 @60Hz. Vizio E322VL 1920x1080 @24Hz. Panasonic TX-L19X10BW 1280x720 @60Hz.

Unsupported Televisions

Tip: If you don't have a monitor/TV you can use a Video Captiure Card like this from PiBox with OBS or any camera application to see the video coming through.

16.3 Peripherals

Note: Most Keyboards, Mouse, and USB Hubs are plug-and-play devices and are supported out of the box in Linux. The list below only shows a subset that has been tested by contributors. Many other options will work without any additional tinkering required on behalf of the user.

16.3.1 Keyboard & Mouse Combo

With limited ports availability on BeagleBones it is recommended to use wireless Keyboard & Mouse combos.

- Adafruit keyboard & Mouse w/batteries
- Logitech K400 Plus Keyboard/Touchpad Combo
- Portronics Key2-A Combo of Multimedia Wireless Keyboard & Mouse

16.3.2 Keyboards

Make sure that you plug the keyboard into the USB Host connector before powering on the board.

- Logitech K400 Plus Keyboard/Touchpad Combo
- Logitech MK345 Keyboard/Mouse Combo
- Logitech MK270 Keyboard/Mouse Combo
- Inland Keyboard and Mouse Combo

16.3.3 Mice

Make sure that you plug the mouse into the USB Host connector before powering on the board.

- Amazon Basics Wireless Computer Mouse
- Logitech M310

16.3.4 USB HUBS

Make sure that you plug the HUB into the USB Host connector before powering on the board.

- Inland 4 Port
- Manhattan 10-port HUB
- 4-Port USB Cable HUB
- D-LINK DUB-H7
- Trust HU-5770 7-Port Powered Hub

Tip: Make sure you are powering BeagleBone with decent power supply with enough current before attaching any additional peripherals. If you require many USB peripheral USB devices to be attached, consider using a "Powered USB Hub" See *Power supplies* for more information on power requirements.

16.4 Cables

16.4.1 USB Data/Power Cables

For all Beagles, there is a USB client, also called gadget, capable connection that will enable you to create network, serial and data storage connections from a host computer.

In most cases, you can also provide power over this same cable.

In most cases, you can also use the port in a host mode, also sometimes called on-the-go to refer to when a device that is typically a client can also act as a host.

Cable included?

A USB (High-speed A to Mini-B) cable will normally be supplied with BeagleBone Black. For other boards, you'll have to procure your own USB cable.

What cable is needed?

The type of cable you have to procure is listed in the table below:

Table 16.2: USB client capable data/power ports on Beagles

Board	USB	Host capable?	Power required[2]
BeaglePlay	High-speed USB-C	Yes[1]	500mA
BeagleV-Fire	High-speed USB-C	Yes	750mA
BeagleV-Ahead	Super-speed Micro-AB	Yes	900mA
BeagleBone AI-64	Super-speed USB-C	Yes	3000mA
BeagleBone AI	Super-speed USB-C	Yes	900mA
BeagleBone Black	High-speed Mini-AB	Yes	500mA
BeagleBone Blue	High-speed Micro-AB	Yes	500mA
BeagleBone Black Wireless	High-speed Micro-AB	Yes	500mA
BeagleBone (original)	High-speed Mini-B	No	500mA
BeagleBoard-xM	High-speed Micro-AB	Yes	500mA
BeagleBoard-X15	High-speed Micro-B	No	N/A
PocketBeagle	High-speed Micro-AB	Yes	500mA

Important: BeagleBoard-X15 cannot be powered over the USB port.

16.4.2 Serial Debug Cables

The default serial port settings for Beagles are:

Table 16.3: UART settings

Setting	Value
Baud	115,200
Bits	8
Parity	N
Stop Bits	1
Handshake	None

JST-SH serial cables

These cables are not active (only wries and connector) and provide an interface between USB to Serial converter cables such as the ones listed below and serial debug ports on Beagles such as BeagleBone AI and AI-64. You can purchase these cables from different sources including:

1. Farnell
2. DigiKey

Standard FTDI Cable

The debug cable is a standard FTDI to TTL cable. **Make sure you get the 3.3V version**! It can purchased from several different sources including but not limited to:

- FTDI serial cable direct
- FTDI serial cable at DigiKey
- FTDI serial cable at Newark

[2] Power requirement is for the base board in typical operation. Peripherals will add to the power requirement.
[1] Requires USB client device that does not require specification dictated type-C PD handshake.

- FTDI serial cable at Sparkfun
- FTDI serial cable at Adafruit

Other options with different USB to Serial ICs exist and will work as well, such as CP2102, CH340G etc but may require additional drivers depending on your operating system.



Pin 1 on the cable is the black wire and connects to pin 1 on the board. (the pin with the white dot next to it)

Adafruit 4 Pin Cable (CP2102)

Adafruit 4-pin serial cable (SiLabs CP2102 based, boards older than 2017 use a Prolific chipset instead)



Table 16.4: Adafruit 4 pin serial cable connection to BeagleBone Black

Board	Wire
Pin 1 (GND)	Black (GND)
Pin 4 (RX)	Green (TX)
Pin 5 (TX)	White (RX)

Note: The naming of the signals reflect those of the cable. The swapping of TX and RX takes place on the board.

You will also find an extra RED wire on this cable that supplies 5V @ 500mA which could power the board if connected to one of the VDD_5V pins. It's recommended that you leave it unconnected.

16.4.3 JTAG debug Cables

TagConnect (JTAG)

Boards like *BeagleConnect Freedom* and *BeaglePlay* use the TagConnect interface which allows you to perform firmware updates and JTAG hardware debugging. To use the interface, the the parts below from tag-connect are required.

1. 10pin TagConnect (no legs) ribbon cable.
2. TagConnect retaining clip.

16.4.4 HDMI Cables

Working HDMI Cables

The BeagleBone Black uses a microHDMI cable.



microHDMI to VGA

Cable Matters Micro HDMI to VGA Adapter

16.4.5 miniDP to HDMI

Working miniDP to HDMI Adapters

Note: BeagleBone-AI64 requires an **ACTIVE** Mini DisplayPort to HDMI cable or adaptor to work, a passive miniDP to HDMI setup will not work at all.

- IVANKY 4K Active Mini DisplayPort to HDMI Adapter
- CableCreation Mini DP (Thunderbolt 2 Compatible) to HDMI

Examples of "Bad" MiniDP to HDMI Adapters

- UGREEN Mini DP Male to HDMI
- AGARO Mini Displayport (Mini Dp) To Hdmi
- AmazonBasics Mini Display Port to HDMI

16.5 Cameras

16.5.1 USB Cameras

Camera	BeagleBone Black	BeagleBone AI-64
Logitech C270		Tested
Logitech C920		Tested
Logitech C922		Tested

16.5.2 CSI Cameras

Note: Using any CSI camera will require you to load an additional overlay.

BeagleBone AI-64

Tip: Additionally a 15 Pin to 22 Pin camera flex cable will be required for the camera to be used on BeagleBone AI-64 if your camera module has a 15 pin connector.

- IMX219 from Arducam
- Raspberry Pi v2 (IMX219)

Chapter 17

Terms & Conditions

17.1 Design

These design materials referred to in this document are NOT SUPPORTED and DO NOT constitute a reference design. Only “community” support is allowed via resources at forum.beagleboard.org.

THERE IS NO WARRANTY FOR THE DESIGN MATERIALS, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE DESIGN MATERIALS “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING,BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND THE PERFORMANCE OF THE DESIGN MATERIALS IS WITH YOU. SHOULD THE DESIGN MATERIALS PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIRING, OR CORRECTION.

This board was designed as an evaluation and development tool. It was not designed with any other application in mind. As such, the design materials that are provided which include schematic, BOM, and PCB files, may or may not be suitable for any other purposes. If used, the design material becomes your responsibility as to whether or not it meets your specific needs or your specific applications and may require changes to meet your requirements.

See the LICENSE file regarding the copyright of these materials.

This LICENSE does not apply to BeagleBoard.org Foundation trademarks. Express written permission is required for use of BeagleBoard.org Foundation trademarks, including, but not limited to BeagleBoard.org, BeagleBone, BeagleBoard, PocketBeagle, BeagleV, BeaglePlay, BeagleConnect, BeagleBoard Compatible, BeagleBoard Embedded, and BeagleBoard Approved. Please visit https://www.beagleboard.org/brand-use and https://www.beagleboard.org/partner-programs for additional details.

17.2 Additional terms

BeagleBoard.org Foundation and logo-licensed manufacturers provide the board under the following conditions:

The user assumes all responsibility and liability for proper and safe handling of the goods. Further, the user indemnifies the Supplier from all claims arising from the handling or use of the goods.

Should the board not meet the specifications indicated in the System Reference Manual, the board may be returned within 90 days from the date of delivery to the distributor of purchase for a full refund. THE FOREGOING LIMITED WARRANTY IS THE EXCLUSIVE WARRANTY MADE BY SELLER TO BUYER AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. EXCEPT TO THE EXTENT OF THE INDEMNITY SET FORTH ABOVE, NEITHER PARTY SHALL BE LIABLE TO THE OTHER FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

Please read the System Reference Manual and, specifically, the Warnings and Restrictions notice in the Systems Reference Manual prior to handling the product. This notice contains important safety information about temperatures and voltages.

No license is granted under any patent right or other intellectual property right of Supplier covering or relating to any machine, process, or combination in which such Supplier products or services might be or are used. The Supplier currently deals with a variety of customers for products, and therefore our arrangement with the user is not exclusive. The Supplier assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein.

17.3 United States FCC and Canada IC regulatory compliance information

The board is annotated to comply with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation. Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This Class A or B digital apparatus complies with Canadian ICES-003. Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

17.4 Board warnings, restrictions and disclaimers

For Feasibility Evaluation Only, in Laboratory/Development Environments. The board is not a complete product. It is intended solely for use for preliminary feasibility evaluation in laboratory/development environments by technically qualified electronics experts who are familiar with the dangers and application risks associated with handling electrical mechanical components, systems and subsystems. It should not be used as all or part of a finished end product.

Your Sole Responsibility and Risk. You acknowledge, represent, and agree that:

You have unique knowledge concerning Federal, State, and local regulatory requirements (including but not limited to Food and Drug Administration regulations, if applicable) which relate to your products and which relate to your use (and/or that of your employees, affiliates, contractors or designees) of the board for evaluation, testing and other purposes.

You have full and exclusive responsibility to assure the safety and compliance of your products with all such laws and other applicable regulatory requirements, and also to assure the safety of any activities to be conducted by you and/or your employees, affiliates, contractors or designees, using the board. Further, you are responsible to assure that any interfaces (electronic and/or mechanical) between the board and any human body are designed with suitable isolation and means to safely limit accessible leakage currents to minimize the risk of electrical shock hazard.

Since the board is not a completed product, it may not meet all applicable regulatory and safety compliance standards which may normally be associated with similar items. You assume full responsibility to determine and/or assure compliance with any such standards and related certifications as may be applicable. You will employ reasonable safeguards to ensure that your use of the board will not result in any property damage, injury, or death, even if the board should fail to perform as described or expected.

Certain Instructions. It is important to operate the board within Supplier's recommended specifications and environmental considerations per the user guidelines. Exceeding the specified the board ratings (including but not limited to input and output voltage, current, power, and environmental ranges) may cause property damage, personal injury, or death. If there are questions concerning these ratings please contact the Supplier representative before connecting interface electronics including input power and intended loads. Any loads applied outside of the specified output range may result in unintended and/or inaccurate operation and/or possible permanent damage to the board and/or interface electronics. Please consult the System Reference Manual before connecting any load to the board output. If there is uncertainty as to the load specification,

please contact the Supplier representative. During normal operation, some circuit components may have case temperatures greater than 60 C as long as the input and output are maintained at a normal ambient operating temperature. These components include but are not limited to linear regulators, switching transistors, pass transistors, and current sense _resistors which can be identified using the board schematic located at the link in the board System Reference Manual. When placing measurement probes near these devices during normal operation, please be aware that these devices may be very warm to the touch. As with all electronic evaluation tools, only qualified personnel knowledgeable in electronic measurement and diagnostics normally found in development environments should use the board.

Agreement to Defend, Indemnify and Hold Harmless. You agree to defend, indemnify and hold the Suppliers, their licensors and their representatives harmless from and against any and all claims, damages, losses, expenses, costs and liabilities (collectively, "Claims") arising out of or in connection with any use of the board that is not in accordance with the terms of the agreement. This obligation shall apply whether Claims arise under the law of tort or contract or any other legal theory, and even if the board _fails to perform as described or expected.

Safety-Critical or Life-Critical Applications. If you intend to evaluate the components for possible use in safety critical applications (such as life support) where a failure of the Supplier's product would reasonably be expected to cause severe personal injury or death, such as devices which are classified as FDA Class III or similar classification, then you must specifically notify Suppliers of such intent and enter into a separate Assurance and Indemnity Agreement.

Mailing Address:

BeagleBoard.org Foundation 4467 Ascot Ct Oakland Twp, MI 48306 U.S.A.

WARRANTY: If purchased from an authorized distributor, as listed on the board page at https://www.beagleboard.org/boards, then the board assembly as purchased is warranted against defects in materials and workmanship for a period of 90 days from purchase. This warranty does not cover any problems occurring as a result of improper use, modifications, exposure to water, excessive voltages, abuse, or accidents. No boards should be sent to back to a distributor without contacting rma/support.

Note: Repairs and replacements only provided on unmodified boards purchased via an authorized distributor **within the first 90 days**. All repaired board will have their flash reset to factory contents. For repairs and replacements, please contact 'support' at BeagleBoard.org using the RMA form:

RMA request

Before making any attempt to return your defective board to a distributor you should visit support page and reach out to Jason for possible solutions.

Additional terms:

- Your repaired/replacement boards will not be sent by priority shipment, please be patient.
- You are responsible for all the expenses if there isn't really an issue with the board.
- If no issue is found or express return is needed, the customer will pay all shipping costs.

For up to date SW images and technical information refer to https://www.beagleboard.org/distros

All support for is provided via community support at https://forum.beagleboard.org

To return a defective board for repair, please request a return materials authorization (RMA) at https://www.beagleboard.org/rma

---

Important: Please **DO NOT** return the board without approval from the RMA team first.
