# Table of contents

1 Introduction  
   1.1 Support ................................................. 1  
      1.1.1 Getting started ........................................ 1  
      1.1.2 Getting support ..................................... 6  
      1.1.3 Understanding Your Beagle ............................. 7  
      1.1.4 Working with Cape Add-on Boards ................... 7  
   1.2 Bone101 ............................................. 7  
      1.2.1 QWIIC, STEMMA and Grove Add-ons in Linux ........... 7  
   1.3 Contribution ........................................... 10  
      1.3.1 Code of Conduct ..................................... 10  
      1.3.2 Frequently Asked Questions ........................... 10  
      1.3.3 What should I know before I get started? ............ 10  
      1.3.4 How can I contribute? ............................... 13  
      1.3.5 Style and usage guidelines ......................... 13  

2 Boards  
   2.1 BeagleBone (all) ....................................... 21  
   2.2 BeagleBone Black ....................................... 22  
      2.2.1 Introduction .......................................... 23  
      2.2.2 Change History ....................................... 23  
      2.2.3 Connecting Up Your BeagleBone Black ................. 26  
      2.2.4 BeagleBone Black Overview ........................... 34  
      2.2.5 BeagleBone Black High Level Specification ............ 38  
      2.2.6 Detailed Hardware Design ............................. 43  
      2.2.7 Connectors ........................................... 72  
      2.2.8 Cape Board Support .................................. 83  
      2.2.9 BeagleBone Black Mechanical ......................... 96  
      2.2.10 Pictures ............................................ 96  
      2.2.11 Support Information ................................. 96  
   2.3 BeagleBone Blue ....................................... 104  
      2.3.1 BeagleBone Blue Pinouts .............................. 105  
      2.3.2 SSH ................................................ 106  
      2.3.3 WiFi Setup .......................................... 106  
      2.3.4 IP settings .......................................... 107  
      2.3.5 Flashing Firmware .................................... 107  
      2.3.6 Play with the code .................................... 109  
      2.3.7 BeagleBone Blue tests ............................... 109  
      2.3.8 Accessories ......................................... 110  
      2.3.9 Frequently Asked Questions (FAQs) .................... 113  
   2.4 BeagleBone AI ......................................... 116  
      2.4.1 Introduction .......................................... 116  
      2.4.2 Change History ....................................... 116  
      2.4.3 Connecting Up Your BeagleBone AI .................... 117  
      2.4.4 BeagleBone AI Overview ............................. 125  
      2.4.5 BeagleBone AI High Level Specification ............... 127  
      2.4.6 Detailed Hardware Design ............................ 133
2.4.7 Connectors ......................................................... 145
2.4.8 Cape Board Support .............................................. 162
2.4.9 Mechanical Information ........................................ 164
2.4.10 Pictures ............................................................. 165
2.4.11 Support Information ............................................. 166
2.4.12 Terms and Conditions ......................................... 166
2.5 BeagleBone AI-64 .................................................. 168
  2.5.1 Introduction ..................................................... 169
  2.5.2 Change History ................................................ 169
  2.5.3 Connecting up your BeagleBone AI-64 ..................... 170
  2.5.4 BeagleBone AI-64 Overview ................................ 177
  2.5.5 BeagleBone AI-64 High Level Specification ............... 180
  2.5.6 Connectors ....................................................... 187
  2.5.7 BeagleBone AI-64 Mechanical ............................... 203
  2.5.8 Pictures ......................................................... 203
  2.5.9 Support Information ......................................... 203
  2.5.10 Update software on BeagleBone AI-64 .................. 208
  2.5.11 Edge AI ......................................................... 209
2.6 PocketBeagle ...................................................... 250
  2.6.1 Introduction .................................................... 251
  2.6.2 Change History ................................................ 251
  2.6.3 Connecting Up PocketBeagle ................................. 253
  2.6.4 PocketBeagle Overview ..................................... 265
  2.6.5 PocketBeagle High Level Specification .................... 267
  2.6.6 Detailed Hardware Design .................................. 272
  2.6.7 Connectors ....................................................... 282
  2.6.8 PocketBeagle Cape Support ................................. 290
  2.6.9 PocketBeagle Mechanical ................................. 290
  2.6.10 Additional Pictures ....................................... 290
  2.6.11 Support Information ...................................... 290
2.7 Capes ............................................................... 292
  2.7.1 BeagleBone cape interface spec ............................ 293
  2.7.2 BeagleBoard/org BeagleBone Relay Cape ................ 314
2.8 BeagleConnect ................................................... 316
  2.8.1 BeagleConnect Technology .................................. 317
  2.8.2 BeagleConnect™ Greybus demo using BeagleConnect™ Freedom 320
  2.8.3 BeagleConnect™ Story ....................................... 334
  2.8.4 BeagleConnect Experience .................................. 334
  2.8.5 BeagleConnect Boards ...................................... 335
2.9 BeagleBoard (all) ................................................ 344
3 Projects .................................................................. 345
  3.1 simpPRU ............................................................... 345
    3.1.1 simpPRU Basics ............................................. 345
    3.1.2 Build from source ......................................... 346
    3.1.3 Install .......................................................... 346
    3.1.4 Language Syntax ........................................... 347
    3.1.5 IO Functions ................................................ 356
    3.1.6 Usage(simprru) ............................................ 360
    3.1.7 Usage(simprru-console) .................................. 361
    3.1.8 simpPRU Examples ....................................... 364
  3.2 BB-Config ........................................................ 378
    3.2.1 BB-Config Detail ........................................... 378
    3.2.2 Build from Source ......................................... 380
    3.2.3 Features ...................................................... 380
    3.2.4 Version ....................................................... 390
4 Books ................................................................ 393
4.1 BeagleBone Cookbook ................................................................. 393
  4.1.1 Basics .................................................................................. 393
  4.1.2 Sensors ............................................................................... 403
  4.1.3 Displays and Other Outputs .................................................. 430
  4.1.4 Motors ............................................................................... 442
  4.1.5 Beyond the Basics ................................................................. 456
  4.1.6 Internet of Things ................................................................. 480
  4.1.7 The Kernel .......................................................................... 517
  4.1.8 Real-Time I/O ..................................................................... 530
  4.1.9 Capes ............................................................................... 542
  4.1.10 Parts and Suppliers ............................................................ 576
4.2 PRU Cookbook ........................................................................... 579
  4.2.1 Case Studies - Introduction .................................................. 579
  4.2.2 Getting Started ................................................................. 611
  4.2.3 Running a Program; Configuring Pins ................................. 620
  4.2.4 Debugging and Benchmarking .............................................. 630
  4.2.5 Building Blocks - Applications ............................................. 646
  4.2.6 Accessing More I/O ............................................................ 716
  4.2.7 More Performance ............................................................ 722
  4.2.8 Moving to the BeagleBone AI .............................................. 733
  4.2.9 PRU Projects ...................................................................... 737
Chapter 1

Introduction

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

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)

Please check out our Support page to find out how to get started, resolve issues, and engage 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 on-going on our Contribution page.

1.1 Support

Note: #TODO# all the links need updating and content moved into this repo, especially bone101.

1.1.1 Getting started

The starting experience for all Beagles has been made to be as consistent as is possible. For any of the Beagle Linux-based open hardware computers, visit Getting Started Guide.

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 Debian image from beagleboard.org/latest-images. 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.
The Debian 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 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

1.1. Support
at http://elinux.org/Beagleboard:BeagleBoneBlack_Debian#Flashing_eMMC. When the flashing is complete, all 4 USRx LEDs will be steady on or off. The latest Debian flasher images automatically 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.

Note:  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 the Quick Start Guide (QSG) or System Reference Manual (SRM) for your board 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 (BeagleBone Blue and BeagleBone AI only).

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 IDE should be running on your Beagle. Point your browser to it 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.
## 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.

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

**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: support.microsoft.com/kb/827218.
3. On systems without the latest service release, you may get an error (0xc000007b). In that case, please install the following and retry: https://www.microsoft.com/en-us/download/confirmation.aspx?id=13523
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 https://www.linux-usb.org/gadget/ and https://joshuawise.com/horndis

Visit https://beagleboard.org/support 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.

**Bad to the Bone**

Perfect for high-school seniors or freshman university level text, consider using “Bad to the Bone”
BeagleBone Cookbook

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

Exploring BeagleBone and Embedded Linux Primer

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

1.1.2 Getting support

BeagleBoard.org products and open hardware designs are 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.

**Note:** #TODO#: Need a reference to how to run beagle-version.

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

Community resources

Please execute the board diagnostics, review the hardware documentation, and consult the mailing list and IRC channel for support. BeagleBoard.org is a “community” project with free support only given to those who are willing to discussing their issues openly for the benefit of the entire community.

- Frequently Asked Questions
- Mailing List
- Live Chat
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

1.1.3 Understanding Your Beagle

- BeagleBone Introduction
- Hardware
- Software
- Books
  - Exploring BeagleBone
  - BeagleBone Cookbook
  - Bad to the Bone

1.1.4 Working with Cape Add-on Boards

- Capes
  - BeagleBone cape interface spec

1.2 Bone101

Note: This page is under construction. Most of the information here is drastically out of date.

This is a collection of articles to aide in quickly understanding how to make use of BeagleBone boards and other Beagles running Linux. Most of the useful information has moved to BeagleBone Cookbook, but some articles are being built here from a different perspective.

Articles under construction or to be imported and updated:

- QWIIC, STEMMA and Grove Add-ons in Linux

1.2.1 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.
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.

```bash
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.

```bash
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 the name and address used to load the appropriate driver for your add-on**
2. **Ensure the driver is included in your kernel build**
3. **Identify the location of the I2C signals on the board and the controller link in Linux**
4. **Ensure the board pinmux is set properly to expose the I2C peripheral**
5. **Ensure the board to add-on connection is good**
6. Issue the Linux command to load the driver

7. Identify and utilize the interface provided by the driver

**Driver name**  One resource that is very helpful is the list that Vaishnav put together for supporting Mikroelektronika Click add-ons. His 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 an `i2c_device_id` table:

```c
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 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.

1.3  Contribution

Note:  This section is under development 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.
Upstream Kernel Contributions

Note: For detailed information on Kernel Development 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 busses (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 inorder 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:
This will generate one file that is generally referred to as the patch file. This is what you will now be sending upstream inorder 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 patch file 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 BeagleBone Black it’s probably not ideal for you to actually clone the entire kernel on BBB 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 busses (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?

**Why “device” drivers?**

TODO

**Why do we need drivers?**

TODO

**What do drivers look like?**

TODO

**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

### 1.3.4 How can I contribute?

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

#### Reporting bugs

#### Suggesting enhancements

#### Submitting merge requests

### 1.3.5 Style and usage guidelines

- *Git Usage*
- *Git commit messages*
- *Documentation Style Guide*

**Git Usage**

**Note:** For detailed information on Git and Gitlab checkout the official [Git and GitLab help page](https://git.beagleboard.org). Also, for good GitLab workflow you can checkout the [Introduction to GitLab Flow (FREE)](https://git.beagleboard.org) page.

These are (draft) general guidelines taken from [BioPython project](https://github.com/biopython/biopython) 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.
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/](http://code.google.com/p/git-osx-installer/)

**Windows**  Download the official installers from [Windows installers](http://beagleboard.org/downloads/git)

**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](http://beagleboard.org/). 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 a GitLab account is very simple:

```bash
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):

```bash
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):

```bash
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,

```bash
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:

```
1.3. Contribution 15
```
git branch demo-branch checkout demo-branch

To check which branch you are on, use:

```bash
git branch
```

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

```bash
git status
```

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

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

and now you commit:

```bash
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:

```bash
git push origin demo
```

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 on GitLab). 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):

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

Then all you need to do is:

```bash
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:

```bash
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 --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:
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

[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 to GitLab:

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://docs.scipy.org/doc/numpy-1.15.1/dev/gitwash/development_workflow.html Numpy is also evaluating git
- https://lab.github.com/courses
- 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 docs is mostly written with ReStructuredText (r)

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. First heading (H2) use =.
4. First heading (H2) use -.
5. First heading (H2) 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 length of the heading text, for example:

<table>
<thead>
<tr>
<th>Incorrect H1</th>
</tr>
</thead>
<tbody>
<tr>
<td>##### ①</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Correct H1</th>
</tr>
</thead>
<tbody>
<tr>
<td>############ ②</td>
</tr>
</tbody>
</table>

① Length of heading symbol # is smaller than the content above.
② Shows the correct way of setting the document title (H1) with #.
Chapter 2

Boards

*BeagleBone* is a family of ARM-based, Linux-capable boards intended to be bare-bones, with a balance of features to enable rapid prototyping and provide a solid reference for building end products.

*PocketBeagle* boards are ultra-tiny ARM-based, Linux-capable boards intended to be very low cost, with minimal features suitable for beginners and attractive to professionals looking for a more minimal starting point.

BeagleBone and PocketBeagle *Capses* are add-on boards for BeagleBone and PocketBeagle boards.

*BeagleConnect* boards are ARM microcontroller-based, Zephyr-capable boards meant to act as ultra low cost smart peripherals to their Linux-capable counterparts, with connectivity options that enable almost endless sensing and actuation expansion.

*BeagleBoard* is a family of ARM-based, Linux-capable boards where this project started.

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 boards-terms-and-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 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.

Contributors
All boards received without RMA approval will not be worked on.

- BeagleBone (original)
- BeagleBone Black
- BeagleBone Blue
- BeagleBone AI
- BeagleBone AI-64

2.2 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.

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 boards-terms-and-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.
2.2.1 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 latest docs rendering for the most up to date information.

https://docs.beagleboard.io/latest/boards/beaglebone/black/

2.2.2 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.

Document Change History

<table>
<thead>
<tr>
<th>Rev</th>
<th>Changes</th>
<th>Date</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>A4</td>
<td>Preliminary</td>
<td>January 4, 2013</td>
<td>GC</td>
</tr>
<tr>
<td>A5</td>
<td>Production release</td>
<td>January 8, 2013</td>
<td>GC</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Rev</th>
<th>Changes</th>
</tr>
</thead>
</table>
| A5.1 | 1. Added information on Power button and the battery access points.  
2. Final production released version. |
| A5.2 | 1. Edited version.  
2. Added numerous pictures of the Rev A5A board. |
| 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. |
| 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. |
| 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. |
| A5.6 | 1. Added information on Rev A5C  
2. Added PRU/ICSS option to tables for P8 and P9.  
3. Added section on USB Host Correct modes on <<table-15>>.  
4. Fixed a few typos |
| 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. |
| A6 | 1. Added Rev A6 changes. |
| A6A | 1. Added Rev A6A changes |
| B | 1. Changed the processor to the AM3358BZCZ |

continues on next page
<table>
<thead>
<tr>
<th>Rev</th>
<th>Changes</th>
<th>Date</th>
<th>By</th>
</tr>
</thead>
</table>
| 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 |

**Board Changes**

**Rev C3**  PCB revision C.
- Updated microSD card cage due to availability. See [https://github.com/beagleboard/beaglebone-black/issues/6](https://github.com/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://github.com/beagleboard/beaglebone-black/issues/4](https://github.com/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/74914bd01efeb61376ec3dda4b9f9143ad2bb635c](https://github.com/beagleboard/beaglebone-black/commit/74914bd01efeb61376ec3dda4b9f9143ad2bb635c).
  - 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](https://github.com/beagleboard/beaglebone-black/commit/5787736d816832cc8cc9629d19f334b6a12e67f9).
  - DDR3:
    - Micron MT41K256M16TW-107:P
  - eMMC:
    - Micron MTFC4GACAJCN-1M WT
    - Kingston EMMC04G-S100-A08U

2.2. BeagleBone Black
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 $1920 \times 1080 \text{@} 24\text{Hz}$. 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.

2.2.3 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.
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. 2.1: In the Box](image)

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
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.

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:

1. Open the USB Drive folder.

2. Click on the file named start.htm
2.2. BeagleBone Black

Fig. 2.4: Board Power LED

Fig. 2.5: Board Boot Status
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 <<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. 2.6: Desktop Configuration](image)

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*. 

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.

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.

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 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.

2.2. BeagleBone Black
Fig. 2.10: Connect Keyboard and Mouse Receiver to the Board

Fig. 2.11: Keyboard and Mouse Hubs

Fig. 2.12: Ethernet Cable Connection
2.2. BeagleBone Black

Fig. 2.13: External DC Power

Fig. 2.14: Connect microHDMI Cable to the Board
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.

![Fig. 2.15: Board Boot Status](image)

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.
   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.

9. Powering Down
   A. Press the power button momentarily.
   B. The system will power down automatically.
   C. Remove the power jack.

2.2.4 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.
Ithasbeenequippedwithaminimumsetoffeaturestoallowtheusertoexperiencethepoweroftheprocessorandis notintendedasafulldevelopmentplatformasmanyofthefeaturesandinterfacessuppliedbytheprocessorarenott accessiblefromtheBeagleBoneBlackviaonboardsupportofsomointerfaces. Itisnotacompleteproductdesigned to donyparticularfunction. It isa 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.

**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

• 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.

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 2.2: BeagleBone Black Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Sitara AM3358BZCZ100 1GHz, 2000 MIPS</td>
</tr>
<tr>
<td>Graphics Engine</td>
<td>SGX530 3D, 20M Polygons/S</td>
</tr>
<tr>
<td>SDRAM Memory</td>
<td>512MB DDR3L 800MHz</td>
</tr>
<tr>
<td>Onboard Flash</td>
<td>4GB, 8-bit Embedded MMC</td>
</tr>
<tr>
<td>PMIC</td>
<td>TPS65217C PMIC regulator and one additional LDO.</td>
</tr>
<tr>
<td>Debug Support</td>
<td>Optional Onboard 20-pin CTI JTAG, Serial Header</td>
</tr>
<tr>
<td>Power Source</td>
<td>miniUSB USB or DC Jack</td>
</tr>
<tr>
<td>PCB</td>
<td>3.4” x 2.1”</td>
</tr>
<tr>
<td>Indicators</td>
<td>1-Power, 2-Ethernet, 4-User Controllable LEDs</td>
</tr>
<tr>
<td>HS USB 2.0 Client Port</td>
<td>Access to USB0, Client mode via miniUSB</td>
</tr>
<tr>
<td>HS USB 2.0 Host Port</td>
<td>Access to USB1, Type A Socket, 500mA LS/FS/HS</td>
</tr>
<tr>
<td>Serial Port</td>
<td>UART0 access via 6 pin 3.3V TTL Header. Header is populated</td>
</tr>
<tr>
<td>Ethernet</td>
<td>10/100, RJ45</td>
</tr>
<tr>
<td>SD/MMC Connector</td>
<td>microSD , 3.3V</td>
</tr>
</tbody>
</table>
| 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                          |

### 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.
Fig. 2.17: Connectors, LEDs and Switches

- **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.

**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.

### 2.2.5 BeagleBone Black High Level Specification

This section provides the high level specification of the BeagleBone Black.

**Block Diagram**

**Processor**

The revision B and later boards have moved to the Sitara AM3358BZCZ100 device.
Memory

Described in the following sections are the three memory devices found on the board.

512MB DDR3L  A single 256Mb x16 DDR3L 4Gb (512MB) memory device is used. The memory used is 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. 2.19: 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 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 *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.

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.

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.

**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.

**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 +/-0.25V.

**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.

**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.
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).

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/](http://www.digikey.com/)

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

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.

2.2.6 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. 2.20: BeagleBone Black Block Diagram
Power Section

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](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](http://en.wikipedia.org/wiki/DC-to-DC_converter).

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.


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.
Fig. 2.22: TPS65217C Block Diagram
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.

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.

2.2. BeagleBone Black 47
Fig. 2.24: 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>
<thead>
<tr>
<th>PIN</th>
<th>DESIGNATION</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAT</td>
<td>TP5</td>
<td>Battery connection point</td>
</tr>
<tr>
<td>SENSE</td>
<td>TP6</td>
<td>Battery voltage sense input, connect to BAT directly at the battery terminal.</td>
</tr>
<tr>
<td>TS</td>
<td>TP7</td>
<td>Temperature sense input. Connect to NTC thermistor to sense battery temperature.</td>
</tr>
<tr>
<td>GND</td>
<td>TP8</td>
<td>System ground.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>MODE</th>
<th>USB</th>
<th>DC</th>
<th>DC+USB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>Idling @ UBoot</td>
<td>210</td>
<td>210</td>
<td>210</td>
</tr>
<tr>
<td>Kernel Booting (Peak)</td>
<td>460</td>
<td>460</td>
<td>460</td>
</tr>
<tr>
<td>Kernel Idling</td>
<td>350</td>
<td>350</td>
<td>350</td>
</tr>
<tr>
<td>Kernel Idling Display Blank</td>
<td>280</td>
<td>280</td>
<td>280</td>
</tr>
<tr>
<td>Loading a Webpage</td>
<td>430</td>
<td>430</td>
<td>430</td>
</tr>
</tbody>
</table>

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**  
**VRTX 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.

---

Fig. 2.25: Power Rails
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
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 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.

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.

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.

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
Fig. 2.26: Power Rail Power Up Sequencing

Fig. 2.27: TPS65217C Power Sequencing Timing

Fig. 2.28: Power Processor Interfaces
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 I2C 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 I2C bus is used to control the voltage scaling function in the TPS65217C.

**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](http://www.ti.com/product/am3358)

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>Linux, Android, Windows Embedded CE, QNX, ThreadX</th>
<th>MMC/SD</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standby Power</td>
<td>7 mW</td>
<td>CAN</td>
<td>2</td>
</tr>
<tr>
<td>ARM CPU</td>
<td>1 ARM Cortex-A8</td>
<td>UART (SCI)</td>
<td>6</td>
</tr>
<tr>
<td>ARM MHz (Max.)</td>
<td>275,500,600,800,1000</td>
<td>ADC</td>
<td>8-ch 12-bit</td>
</tr>
<tr>
<td>ARM MIPS (Max.)</td>
<td>1000,1200,2000</td>
<td>PWM (Ch)</td>
<td>3</td>
</tr>
<tr>
<td>Graphics Acceleration</td>
<td>1 3D</td>
<td>eCAP</td>
<td>3</td>
</tr>
</tbody>
</table>

continues on next page

---

2.2. BeagleBone Black 53
Table 2.5 – continued from previous page

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>Linux, Android, Windows Embedded CE, QNX, ThreadX</th>
<th>MMC/SD</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Hardware Acceleration</td>
<td>2 PRU-ICSS, Crypto Accelerator</td>
<td>eQEP</td>
<td>3</td>
</tr>
<tr>
<td>On-Chip L1 Cache</td>
<td>64 KB (ARM Cortex-A8)</td>
<td>RTC</td>
<td>1</td>
</tr>
<tr>
<td>On-Chip L2 Cache</td>
<td>256 KB (ARM Cortex-A8)</td>
<td>I2C</td>
<td>3</td>
</tr>
<tr>
<td>Other On-Chip Memory</td>
<td>128 KB</td>
<td>McASP</td>
<td>2</td>
</tr>
<tr>
<td>Display Options</td>
<td>LCD</td>
<td>SPI</td>
<td>2</td>
</tr>
<tr>
<td>General Purpose Memory</td>
<td>1 16-bit (GPMC, NAND flash, NOR Flash, SRAM)</td>
<td>DMA (Ch)</td>
<td>64-Ch EDMA</td>
</tr>
<tr>
<td>DRAM</td>
<td>1 16-bit (LPDDR-400, DDR2-532, DDR3-400)</td>
<td>IO Supply (V)</td>
<td>1.8V(ADC), 3.3V</td>
</tr>
<tr>
<td>USB Ports</td>
<td>2</td>
<td>Operating Temperature Range (C)</td>
<td>40 to 90</td>
</tr>
</tbody>
</table>

High Level Features

Documentation Full documentation for the processor can be found on the TI website at [http://www.ti.com/product/am3358](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.
2.2. BeagleBone Black

Fig. 2.29: Sitara AM3358BZCZ Block Diagram

Fig. 2.30: Processor Crystals
Fig. 2.31: Board Reset Circuitry

**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.

**Address Lines:** Provide the row address for ACTIVATE commands, and the column address and auto pre-charge 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
Fig. 2.32: DDR3L Memory Design
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.

**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. 2.33: DDR3L VREF Design](image)

**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. 2.34: eMMC Memory Design](image)

The pins used by the eMMC1 in the boot mode are listed below in Table 6.

![Fig. 2.35: eMMC Boot Pins](image)

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.

### 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>
<thead>
<tr>
<th>Name</th>
<th>Size (bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>4</td>
<td>0xAA, 0x55, 0x33, EE</td>
</tr>
<tr>
<td>Board Name</td>
<td>8</td>
<td>Name for board in ASCII: A335BNLT</td>
</tr>
</tbody>
</table>

continues on next page
### Table 2.6 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>Serial Number</td>
<td>12</td>
<td>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</td>
</tr>
<tr>
<td>Configuration Option</td>
<td>32</td>
<td>Codes to show the configuration setup on this board. All FF</td>
</tr>
<tr>
<td>RSVD</td>
<td>6</td>
<td>FF FF FF FF FF FF FF FF FF FF FF FF</td>
</tr>
<tr>
<td>RSVD</td>
<td>6</td>
<td>FF FF FF FF FF FF FF FF FF FF FF FF FF</td>
</tr>
<tr>
<td>RSVD</td>
<td>6</td>
<td>FF FF FF FF FF FF FF FF FF FF FF FF FF</td>
</tr>
<tr>
<td>Available</td>
<td>4018</td>
<td>Available space for other non-volatile codes/data</td>
</tr>
</tbody>
</table>

![Board ID](image)

Fig. 2.36: 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.

**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.

The MMC0_CMD signal indicates that a command versus data is being sent.

There is no separate card detect pin in the microSD specification. It uses MMC0_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.
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.

Resistors R71-R74 were changed to 4.75K on the revision A5B and later boards.

<table>
<thead>
<tr>
<th>LED</th>
<th>GPIO SIGNAL</th>
<th>PROC PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>USR0</td>
<td>GPIO1_21</td>
<td>V15</td>
</tr>
<tr>
<td>USR1</td>
<td>GPIO1_22</td>
<td>U15</td>
</tr>
<tr>
<td>USR2</td>
<td>GPIO1_23</td>
<td>T15</td>
</tr>
<tr>
<td>USR3</td>
<td>GPIO1_24</td>
<td>V16</td>
</tr>
</tbody>
</table>

A logic level of “1” will cause the LEDs to turn on.

**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 to the SPL, the secondary stage (primary program loader—"SPL" or "MLO") takes over. The SPL stage initializes 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 U-Boot code. In the case of the BeagleBone Black, that is either the MMC or microSD based on the position of the boot switch.
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.

**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.

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 SPI00 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.

**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.

**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.

**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
Fig. 2.42: Ethernet Connector Interface

Fig. 2.43: Ethernet PHY, Power, Reset, and Clocks
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.

**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.

**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>
<thead>
<tr>
<th>RESOLUTION</th>
<th>AUDIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>800 x 600 @60Hz</td>
<td></td>
</tr>
<tr>
<td>800 x 600 @56Hz</td>
<td></td>
</tr>
<tr>
<td>640 x 480 @75Hz</td>
<td></td>
</tr>
<tr>
<td>640 x 480 @60Hz</td>
<td>YES</td>
</tr>
<tr>
<td>720 x 400 @70Hz</td>
<td></td>
</tr>
<tr>
<td>1280 x 1024 @75Hz</td>
<td></td>
</tr>
<tr>
<td>1024 x 768 @75Hz</td>
<td></td>
</tr>
<tr>
<td>1024 x 768 @70Hz</td>
<td></td>
</tr>
<tr>
<td>1024 x 768 @60Hz</td>
<td></td>
</tr>
<tr>
<td>800 x 600 @75Hz</td>
<td></td>
</tr>
<tr>
<td>800 x 600 @72Hz</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 2.8 – continued from previous page

<table>
<thead>
<tr>
<th>RESOLUTION</th>
<th>AUDIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>720 x 480 @60Hz</td>
<td>YES</td>
</tr>
<tr>
<td>1280 x 720 @60Hz</td>
<td>YES</td>
</tr>
<tr>
<td>1920 x 1080 @24Hz</td>
<td>YES</td>
</tr>
</tbody>
</table>

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 eliminates the need of any additional device to handle this feature. While this feature is supported in this device, as of this point, the SW to support this feature has not been implemented and is not a feature that is considered critical. It can be switched to very low power Standby or Sleep modes to save power when HDMI is not used. TDA19988 embeds 1–2–C-bus master interface for DDC-bus communication to read EDID. This device can be controlled or configured via I–2–C-bus interface.

**HDMI Video Processor Interface**

The Figure 44 shows the connections between the processor and the HDMI framer device. There are 16 bits of display data, 5-6-5 that is used to drive the framer. The reason for 16 bits is that allows for compatibility with display and LCD capes already available on the original BeagleBone. The unused bits on the TDA19988 are tied low. In addition to the data signals are the VSYNC, HSYNC, DE, and PCLK signals that round out the video interface from the processor.

**HDMI Control Processor Interface**

In order to use the TDA19988, the processor needs to setup the device. This is done via the I2C interface between the processor and the TDA19988. There are two signals on the TDA19988 that could be used to set 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.

**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 TDA19988. 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.

**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.
2.2. BeagleBone Black

Fig. 2.45: HDMI Framer Processor Interface

Fig. 2.46: TDA19988 I2C Address

Fig. 2.47: 24.576MHZ Oscillator
HDMI Connector Interface

Figure 2.48 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.

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.

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.
2.2. BeagleBone Black

**Fig. 2.49: Connector Interface Circuitry**

**Figure 48. USB Host Circuitry**

Fig. 2.50: USB Host circuit
**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).*

---

**Fig. 2.51: PRU-ICSS Block Diagram**

**PRU-ICSS Block Diagram**

**PRU-ICSS Pin Access**  Both PRU 0 and PRU1 are accessible from the expansion headers. Some may not be useable 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 2.9: P8 PRU0 and PRU1 Access**

<table>
<thead>
<tr>
<th>PIN</th>
<th>PROC</th>
<th>NAME</th>
<th>PROC NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>R12</td>
<td>GPIO1_13</td>
<td>pr1_pru0_pru_r30_15 (Output)</td>
</tr>
<tr>
<td>12</td>
<td>T12</td>
<td>GPIO1_12</td>
<td>pr1_pru0_pru_r30_14 (Output)</td>
</tr>
</tbody>
</table>

continues on next page
## Table 2.9 – continued from previous page

<table>
<thead>
<tr>
<th>PIN</th>
<th>PROC</th>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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_r31_13  
(Output) |
| 21  | U9   | GPIO1_30 | pr1_pru1_pru_r31_12  
(Output) |
| 27  | U5   | GPIO2_22 | pr1_pru1_pru_r31_8  
(Output) |
| 28  | V5   | GPIO2_24 | pr1_pru1_pru_r31_10  
(Output) |
| 29  | R5   | GPIO2_23 | pr1_pru1_pru_r31_9  
(Output) |
| 39  | T3   | GPIO2_12 | pr1_pru1_pru_r31_6  
(Output) |
| 40  | T4   | GPIO2_13 | pr1_pru1_pru_r31_7  
(Output) |
| 41  | T1   | GPIO2_10 | pr1_pru1_pru_r31_4  
(Output) |
| 42  | T2   | GPIO2_11 | pr1_pru1_pru_r31_5  
(Output) |
| 43  | R3   | GPIO2_8  | pr1_pru1_pru_r31_2  
(Output) |
| 44  | R4   | GPIO2_9  | pr1_pru1_pru_r31_3  
(Output) |
| 45  | R1   | GPIO2_6  | pr1_pru1_pru_r31_0  
(Output) |
| 46  | R2   | GPIO2_7  | pr1_pru1_pru_r31_1  
(Output) |

## Table 2.10: P9 PRU0 and PRU1 Access

<table>
<thead>
<tr>
<th>PIN</th>
<th>PROC</th>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>A16</td>
<td>I2C1_SCL</td>
<td>pr1_uart0_txd</td>
</tr>
<tr>
<td>18</td>
<td>B16</td>
<td>I2C1_SDA</td>
<td>pr1_uart0_rxd</td>
</tr>
<tr>
<td>19</td>
<td>D17</td>
<td>I2C2_SCL</td>
<td>pr1_uart0 rte_s_n</td>
</tr>
<tr>
<td>20</td>
<td>D18</td>
<td>I2C2_SDA</td>
<td>pr1_uart0 cts_s_n</td>
</tr>
<tr>
<td>21</td>
<td>B17</td>
<td>UART2_TXD</td>
<td>pr1_uart0 cts_s_n</td>
</tr>
<tr>
<td>22</td>
<td>A17</td>
<td>UART2_RXD</td>
<td>pr1_uart0 cts_s_n</td>
</tr>
</tbody>
</table>
| 24  | D15  | UART1_TXD | pr1_uart0_txd  
(Input) |
| 25  | A14  | GPIO3_21 | pr1_pru0_pru_r31_5  
(Output) |
| 26  | D16  | UART1_RXD | pr1_pru0_pru_r31_6  
(Output) |
| 27  | C13  | GPIO3_19 | pr1_pru0_pru_r31_7  
(Output) |
| 28  | C12  | SPI1_CS0 | pr1_pru0_pru_r31_3  
(Output) |
| 29  | B13  | SPI1_D0  | pr1_pru0_pru_r31_1  
(Output) |
| 30  | D12  | SPI1_D1  | pr1_pru0_pru_r31_2  
(Output) |

continues on next page
## 2.10 Connectors

### 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.

**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>
<thead>
<tr>
<th>PIN</th>
<th>PROC</th>
<th>NAME</th>
<th>pr1_pru0_pru_r30_0 (Output)</th>
<th>pr1_pru0_pru_r31_0 (Input)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>A13</td>
<td>SPI1_SCLK</td>
<td>pr1_pru0_pru_r30_0</td>
<td>pr1_pru0_pru_r31_0</td>
</tr>
</tbody>
</table>

**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.
Fig. 2.52: Expansion Connector Location
### Table 2.11: ExpansionHeaderP8Pinout

| PIN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 10 | 11 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 21 | 24 | 25 | 26 | 27 | 28 | 30 | 31 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 44 | 46 |

---

BeagleBoard Docs, Release 0.0.20230323

---

Chapter 2. Boards
**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>
<thead>
<tr>
<th>NAME</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
<th>MODULE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B26</td>
<td>B27</td>
<td>B28</td>
<td>B29</td>
<td>B30</td>
<td>B31</td>
<td>B32</td>
<td>B33</td>
<td>B34</td>
<td>B35</td>
<td>B36</td>
<td>B37</td>
<td>B38</td>
<td>B39</td>
<td>B40</td>
<td>B41</td>
</tr>
<tr>
<td></td>
<td>B42</td>
<td>B43</td>
<td>B44</td>
<td>B45</td>
<td>B46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Chapter 2. Boards**

76
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).

The board requires a regulated 5VDC +/-0.25V 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.

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.

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.

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. 2.54: USB Client

Fig. 2.55: USB Host Connector
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.

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

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.

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 http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232R_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 2.13: J1 Serial Header Pins

<table>
<thead>
<tr>
<th>PIN NUMBER</th>
<th>SIGNAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ground</td>
</tr>
<tr>
<td>4</td>
<td>Receive</td>
</tr>
<tr>
<td>5</td>
<td>Transmit</td>
</tr>
</tbody>
</table>

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.

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.

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.
2.2. BeagleBone Black

Fig. 2.59: HDMI Connector

Fig. 2.60: HDMI Cable
Fig. 2.61: microSD Connector
Ethernet

The board comes with a single 10/100 Ethernet interface located next to the power jack as shown in Figure below.

![Ethernet Connector](image)

Fig. 2.62: Ethernet Connector

The PHY supports AutoMDX which means either a straight or a swap cable can be used.

**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/](http://www.digikey.com/)

### 2.2.8 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.**

**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>
<thead>
<tr>
<th>PIN</th>
<th>PROC</th>
<th>NAME</th>
<th>MODE0</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>U5</td>
<td>GPIO2_22</td>
<td>lcd_vsync</td>
</tr>
<tr>
<td>28</td>
<td>V5</td>
<td>GPIO2_24</td>
<td>lcd_pclk</td>
</tr>
<tr>
<td>29</td>
<td>R5</td>
<td>GPIO2_23</td>
<td>lcd_hsync</td>
</tr>
<tr>
<td>30</td>
<td>R6</td>
<td>GPIO2_25</td>
<td>lcd_ac_bias_en</td>
</tr>
<tr>
<td>31</td>
<td>V4</td>
<td>UART5_CTSN</td>
<td>lcd_data14</td>
</tr>
<tr>
<td>32</td>
<td>T5</td>
<td>UART5_DTSN</td>
<td>lcd_data15</td>
</tr>
<tr>
<td>33</td>
<td>V3</td>
<td>UART4_DTSN</td>
<td>lcd_data13</td>
</tr>
<tr>
<td>34</td>
<td>U4</td>
<td>UART3_DTSN</td>
<td>lcd_data11</td>
</tr>
<tr>
<td>35</td>
<td>V2</td>
<td>UART4_CTSN</td>
<td>lcd_data12</td>
</tr>
<tr>
<td>36</td>
<td>U3</td>
<td>UART3_CTSN</td>
<td>lcd_data10</td>
</tr>
<tr>
<td>37</td>
<td>U1</td>
<td>UART5_TXD</td>
<td>lcd_data8</td>
</tr>
<tr>
<td>38</td>
<td>U2</td>
<td>UART5_RXD</td>
<td>lcd_data9</td>
</tr>
<tr>
<td>39</td>
<td>T3</td>
<td>GPIO2_12</td>
<td>lcd_data6</td>
</tr>
<tr>
<td>40</td>
<td>T4</td>
<td>GPIO2_13</td>
<td>lcd_data7</td>
</tr>
<tr>
<td>41</td>
<td>T1</td>
<td>GPIO2_10</td>
<td>lcd_data4</td>
</tr>
<tr>
<td>42</td>
<td>T2</td>
<td>GPIO2_11</td>
<td>lcd_data5</td>
</tr>
<tr>
<td>43</td>
<td>R3</td>
<td>GPIO2_8</td>
<td>lcd_data2</td>
</tr>
<tr>
<td>44</td>
<td>R4</td>
<td>GPIO2_9</td>
<td>lcd_data3</td>
</tr>
<tr>
<td>45</td>
<td>R1</td>
<td>GPIO2_6</td>
<td>lcd_data0</td>
</tr>
<tr>
<td>46</td>
<td>R2</td>
<td>GPIO2_7</td>
<td>lcd_data1</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>PIN</th>
<th>PROC</th>
<th>SIGNAL</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>V8</td>
<td>MMC1_DAT5</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>U8</td>
<td>MMC1_DAT4</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>V7</td>
<td>MMC1_DAT1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>R8</td>
<td>MMC1_DAT2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>T9</td>
<td>MMC1_DAT7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>R9</td>
<td>MMC1_DAT6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>T8</td>
<td>MMC1_DAT3</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>U7</td>
<td>MMC1_DAT0</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>V9</td>
<td>MMC1_CMD</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>U9</td>
<td>MMC1_CLK</td>
<td>2</td>
</tr>
</tbody>
</table>

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.

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.

![Figure 61: Expansion Board EEPROM Without Write Protect](image)

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.

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**

![Figure 2.64: Expansion Board EEPROM Write Protect](image)

**EEPROM Data Format**  
Table 17 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Size (bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>0</td>
<td>4</td>
<td>0xAA, 0x55, 0x33, 0xEE</td>
</tr>
<tr>
<td>EEPROM Revision</td>
<td>4</td>
<td>2</td>
<td>Revision number of the overall format of this EEPROM in ASCII=A1</td>
</tr>
<tr>
<td>Board Name</td>
<td>6</td>
<td>32</td>
<td>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.</td>
</tr>
<tr>
<td>Version</td>
<td>38</td>
<td>4</td>
<td>Hardware version code for board in ASCII. Version format is up to the developer. i.e. 02.1…00A1….10A0</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>42</td>
<td>16</td>
<td>ASCII name of the manufacturer. Company or individual’s name.</td>
</tr>
<tr>
<td>Part Number</td>
<td>58</td>
<td>16</td>
<td>ASCII Characters for the part number. Up to maker of the board.</td>
</tr>
<tr>
<td>Number of Pins</td>
<td>74</td>
<td>2</td>
<td>Number of pins used by the daughter board including the power pins used. Decimal value of total pins 92 max, stored in HEX.</td>
</tr>
<tr>
<td>Serial Number</td>
<td>76</td>
<td>12</td>
<td>Serial number of the board. This is a 12 character string which is: WWYY&amp;&amp;&amp;&amp;nnnn</td>
</tr>
<tr>
<td>Pin Usage</td>
<td>88</td>
<td>148</td>
<td>Two bytes for each configurable pins of the 74 pins on the expansion connectors. MSB-LSB Bit order: 15..14….8…7…6…5…4….3….2….1….0 Enabled Bit 4….PullUp/Dn Select….0=Pulldown1=PullUp Bit 3….PullUp/DN enabled…0=Enabled1=Disabled Bits 2-0….Mux Mode Selection….Mode 0-7</td>
</tr>
<tr>
<td>VDD_3V3B Current</td>
<td>236</td>
<td>2</td>
<td>Maximum current in milliams. This is HEX value of the current in decimal 1500mA=0x050xDC325mA=0x010x45</td>
</tr>
<tr>
<td>VDD_5V Current</td>
<td>238</td>
<td>2</td>
<td>Maximum current in milliams. This is HEX value of the current in decimal 1500mA=0x050xDC325mA=0x010x45</td>
</tr>
<tr>
<td>SYS_5V Current</td>
<td>240</td>
<td>2</td>
<td>Maximum current in milliams. This is HEX value of the current in decimal 1500mA=0x050xDC325mA=0x010x45</td>
</tr>
<tr>
<td>DC Supplied</td>
<td>242</td>
<td>2</td>
<td>Indicates whether or not the board is supplying voltage on the VDD_5V rail and the current rating.</td>
</tr>
<tr>
<td>Available</td>
<td>244</td>
<td>32543</td>
<td>Available space for other non-volatile codes/data to be used as needed by the manufacturer.</td>
</tr>
</tbody>
</table>

**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

2.2. BeagleBone Black
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 2.17: EEPROM Pin Usage

<table>
<thead>
<tr>
<th>Offset</th>
<th>Conn</th>
<th>Name</th>
<th>Pin Usage</th>
<th>Type</th>
<th>Reserved</th>
<th>SLEW</th>
<th>RX</th>
<th>PU-PD</th>
<th>PU/DE</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>P9-22</td>
<td>UART2_RXD</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>P9-21</td>
<td>UART2_TXD</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>P9-18</td>
<td>I2C1_SDA</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>P9-17</td>
<td>I2C1_SCL</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>P9-42</td>
<td>GPIO0_7</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>P8-35</td>
<td>UART4_CTSN</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>P8-33</td>
<td>UART4_RTSN</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>P8-31</td>
<td>UART5_CTSN</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>P8-32</td>
<td>UART5_RTSN</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>P9-19</td>
<td>I2C2_SCL</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>P9-20</td>
<td>I2C2_SDA</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>P9-26</td>
<td>UART*T1_RXD</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>P9-24</td>
<td>UART*T1_TXD</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>P9-41</td>
<td>CLKO1Out2</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>P8-19</td>
<td>EHRPWM2A</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>P8-13</td>
<td>EHRPWM2B</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>P8-14</td>
<td>GPIO0_26</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>P8-17</td>
<td>GPIO0_27</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>P9-11</td>
<td>UART4_RXD</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>P9-13</td>
<td>UART4_TXD</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>P8-25</td>
<td>GPIO1_0</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>P8-24</td>
<td>GPIO1_1</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>P8-5</td>
<td>GPIO1_2</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>P8-6</td>
<td>GPIO1_3</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>P8-23</td>
<td>GPIO1_4</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>P8-22</td>
<td>GPIO1_5</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>P8-3</td>
<td>GPIO1_6</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>P8-4</td>
<td>GPIO1_7</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>P8-12</td>
<td>GPIO1_12</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>P8-11</td>
<td>GPIO1_13</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>P8-16</td>
<td>GPIO1_14</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>P8-15</td>
<td>GPIO1_15</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>P9-15</td>
<td>GPIO1_16</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
</tbody>
</table>

Continues...
BeagleBoard Docs, Release 0.0.20230323

Table 2.18 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Pin Usage</th>
<th>Type</th>
<th>Reserve</th>
<th>S L E W</th>
<th>R X</th>
<th>P U - P D</th>
<th>P U / DE M</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8-20</td>
<td>GPIO1_31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8-18</td>
<td>GPIO2_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8-7</td>
<td>TIMER4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8-9</td>
<td>TIMER5</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-10</td>
<td>TIMER6</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-8</td>
<td>TIMER7</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-45</td>
<td>GPIO2_6</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-46</td>
<td>GPIO2_7</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-43</td>
<td>GPIO2_8</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-44</td>
<td>GPIO2_9</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-41</td>
<td>GPIO2_10</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-42</td>
<td>GPIO2_11</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-39</td>
<td>GPIO2_12</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-40</td>
<td>GPIO2_13</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-37</td>
<td>UART5_TX+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-38</td>
<td>UART5_RX+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-36</td>
<td>UART3_CTSN</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-34</td>
<td>UART3_RTSN</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-27</td>
<td>GPIO2_22</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-29</td>
<td>GPIO2_23</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-28</td>
<td>GPIO2_24</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P8-30</td>
<td>GPIO2_25</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-29</td>
<td>SPI1_D0</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-30</td>
<td>SPI1_D1</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-28</td>
<td>SPI1_CS0</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-27</td>
<td>GPIO3_19</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-31</td>
<td>SPI1_SCLK</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-25</td>
<td>GPIO3_21</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-39</td>
<td>AIN0</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-40</td>
<td>AIN1</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-37</td>
<td>AIN2</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-38</td>
<td>AIN3</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-33</td>
<td>AIN4</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-36</td>
<td>AIN5</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>P9-35</td>
<td>AIN6</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

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:

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
Figure 63. Expansion Boot Pins

Fig. 2.65: Boot signals
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 VILmax voltage of 0.8 volts and AM3358 maximum input leakage current of 18uA plus any other current leakage paths on these signals.

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. 2.66: Single Expansion Connector](image)

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. 2.67: Single Cape Expansion Connector](image)

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>
<thead>
<tr>
<th>SUPPLIER</th>
<th>PARTNUMBER</th>
<th>LENGTH(in)</th>
<th>OVERHANG(in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major League</td>
<td>TSHC-123-D-03-145-G-LF</td>
<td>.145</td>
<td>.004</td>
</tr>
<tr>
<td>Major League</td>
<td>TSHC-123-D-03-240-G-LF</td>
<td>.240</td>
<td>.099</td>
</tr>
<tr>
<td>Major League</td>
<td>TSHC-123-D-03-255-G-LF</td>
<td>.255</td>
<td>.114</td>
</tr>
</tbody>
</table>
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.

![Figure 66: Expansion Connector](image)

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.

![Figure 67: Stacked Cape Expansion Connector](image)

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>
<thead>
<tr>
<th>SUPPLIER</th>
<th>PART NUMBER</th>
<th>TAIL LENGTH([in])</th>
<th>OVERHANG([in])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major League</td>
<td>SSHQ-123-D-06-G-LF</td>
<td>.190</td>
<td>0.049</td>
</tr>
<tr>
<td>Major League</td>
<td>SSHQ-123-D-08-G-LF</td>
<td>.390</td>
<td>0.249</td>
</tr>
<tr>
<td>Major League</td>
<td>SSHQ-123-D-10-G-LF</td>
<td>.560</td>
<td>0.419</td>
</tr>
</tbody>
</table>

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 Stealing** *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.

**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.

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.

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.**

### 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>
<thead>
<tr>
<th>Current</th>
<th>Name</th>
<th>P9</th>
<th>P9</th>
<th>Name</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>250mA</td>
<td>VDD_3V3B</td>
<td>3</td>
<td>4</td>
<td>VDD_3V3B</td>
<td>250mA</td>
</tr>
<tr>
<td>1000mA</td>
<td>VDD_5V</td>
<td>5</td>
<td>6</td>
<td>VDD_5V</td>
<td>1000mA</td>
</tr>
<tr>
<td>250mA</td>
<td>SYS_5V</td>
<td>7</td>
<td>8</td>
<td>SYS_5V</td>
<td>250mA</td>
</tr>
</tbody>
</table>

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.*

### 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.

![Cape Board Dimensions](image)

**Fig. 2.72: Cape Board Dimensions**

**Standard Cape Size** 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.

2.2. BeagleBone Black
2.2.9 BeagleBone Black Mechanical

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

Silkscreen and Component Locations

2.2.10 Pictures

2.2.11 Support Information

All support for this design is through the BeagleBoard.org community at: beagleboard@googlegroups.com or http://beagleboard.org/discuss

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).

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://beagleboard.org/latest-images

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

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
2.2. BeagleBone Black

Fig. 2.73: Board Dimensions
Fig. 2.74: Component Side Silkscreen
Fig. 2.75: Circuit Side Silkscreen
Fig. 2.76: Top Side
Fig. 2.77: Bottom Side
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.

![Initial Serial Number and Revision Locations](image1)

Fig. 2.78: Initial Serial Number and Revision Locations

![Second Phase Serial Number and Revision Location](image2)

Fig. 2.79: Second Phase Serial Number and Revision Location

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 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 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](http://elinux.org/Beagleboard:BeagleBoneBlack_HDMI)

### 2.3 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.

**Contributors**

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-sa/4.0/)

**Note:** Make sure to read and accept all the terms & condition provided in the boards-terms-and-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.
2.3.1 BeagleBone Blue Pinouts

- Connector pinout details from schematic(s)
• Pin Table with some Blue : Black correlation.

**UT1**

UART (/dev/ttyS1)

| config-pin P9.24 uart | config-pin P9.26 uart |

**GPS**

UART (/dev/ttyS2)

| config-pin P9.21 uart | config-pin P9.22 uart |

### 2.3.2 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 passwd 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.

```sh
sudo -s
(become superuser/root)
enter your password
passwd debian
   (put your simple password **in**)
exit
   (exit from superuser/root)
```

When you are running as root, passwd is more compliant and will accept simple password

### 2.3.3 WiFi Setup

On my network, I'm set up as ip 192.168.1.*. To turn your wifi on, do the following.

```sh
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
```

(continues on next page)
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

2.3.4 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)

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

2.3.5 Flashing Firmware

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.

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.
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

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:

```bash
mkdir sd_tmp
sudo mount /dev/mmcblk0p1 sd_tmp
sudo su
```

```bash
echo "cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh" >> sd_tmp/boot/
```
```
..uEnv.txt
```

```bash
exit
```n
```
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.
2. Use the Etcher utility to burn the Beaglebone image you want to use on the micro sd card you plan on using.
3. 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.

### 2.3.6 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`

<table>
<thead>
<tr>
<th>rc_balance</th>
<th>rc_dsm_passthrough</th>
<th>rc_test_encoders</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc_battery_monitor</td>
<td>rc_kill</td>
<td>rc_test_filters</td>
</tr>
<tr>
<td>rc_benchmark_algebra</td>
<td>rc_spi_loopback</td>
<td>rc_test_imu</td>
</tr>
<tr>
<td>rc_bind_dsm</td>
<td>rc_startup_routine</td>
<td>rc_test_motors</td>
</tr>
<tr>
<td>rc Blink</td>
<td>rc_test_adc</td>
<td>rc_test_polynomial</td>
</tr>
<tr>
<td>rc_calibrate_dsm</td>
<td>rc_test_algebra</td>
<td>rc_test_servos</td>
</tr>
<tr>
<td>rc_calibrate_escs</td>
<td>rc_test_barometer</td>
<td>rc_test_time</td>
</tr>
<tr>
<td>rc_calibrate_gyro</td>
<td>rc_test_buttons</td>
<td>rc_test_vector</td>
</tr>
<tr>
<td>rc_calibrate_mag</td>
<td>rc_test_cape</td>
<td>rc_uart_loopback</td>
</tr>
<tr>
<td>rc_check_battery</td>
<td>rc_test_dmp</td>
<td>rc_version</td>
</tr>
<tr>
<td>rc_check_model</td>
<td>rc_test_drivers</td>
<td></td>
</tr>
<tr>
<td>rc_cpu_freq</td>
<td>rc_test_dsm</td>
<td></td>
</tr>
</tbody>
</table>

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

### 2.3.7 BeagleBone Blue tests

#### ADC

- Grove Rotary Angle Sensor See output on adc_1 source

```
rc_test_adc
```

#### 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
```

#### 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

```bash
cd /sys/class/gpio; echo 97 >export; cd gpio97; echo in >direction; watch -n0 cat value
```

**UT1**

• Grove GPS

```bash
tio /dev/ttyO1 -b 9600
```

**GPS**

• GPS Receiver - EM-506

```bash
tio /dev/ttyO2 -b 4800
```

**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

```bash
cd /sys/bus/i2c/devices/i2c-1; echo tsl2561 0x29 >new_device; watch -n0 cat 1-0029/\n:iio\:device0/in_illuminance0_input
```

• Temperature & Humidity Sensor

```bash
cd /sys/bus/i2c/devices/i2c-1; echo th02 0x40 >new_device; watch -n0 cat 1-0040/iio\n:iio\:device0/in_temp_raw
```

**Motors**

```bash
rc_test_motors
```

## 2.3.8 Accessories

**Chassis and kits**

• EduMIP

• Pololu Romi Chassis with geared motors
  – Wheel encoders
  – Chassis - Black

• Sprout Runt Rover

**Cases**

**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)

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

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)

Motors
2-wire JST-ZH (1.5mm pitch)
- Digi-Key wires
- Digi-Key receptacle
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

Power supplies

12V with 5.5mm/2.1mm center positive

- Jameco: supply and power cord
- Hobby King 12V 3A supply

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

Radio remotes

- Hobby King OrangeRX satellite receiver
- Spektrum DSM2 Remote Receiver

GPS

- Sparkfun GPS Receiver - EM-506 (48 Channel)
- Adafruit Ultimate GPS breakout
- Ublox Neo-M8N GPS with Compass
- SeeedStudio Grove - GPS
Replacement antennas

- LSR PIFA
- LSR Dipole: antenna and cable
- Anaren U.FL 2.4GHz 6MM Antenna
- TI approved antennas

USB devices

USB cameras

- Logitech C270
- Logitech C920

SPI devices

SPI TFT displays

- Adafruit 2.4” LCD breakout

I2C devices

- See One-Liner-Module-Tests#Grove_I2C_modules

UART devices

Computer serial adapters

- Sparkfun FTDI Cable 5V VCC-3.3V I/O
- Adafruit FTDI Serial TTL-232 USB Cable

Bluetooth devices

- WowWee Groove Cube Speaker

2.3.9 Frequently Asked Questions (FAQs)

Are there any books to help me get started?


For more general books on BeagleBone, Linux and other related topics, see https://beagleboard.org/books.

What system firmware should I use for starting to explore my BeagleBone Blue?


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.
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

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 Cloud9 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.

How do I connect BeagleBone Blue to my own WiFi network?

From the bash command prompt in Linux:

```bash
sudo -s (become superuser/root)
connmanctl
cconnmanctl> tether wifi off (not really necessary on latest images)
cconnmanctl> enable wifi (not really necessary)
cconnmanctl> scan wifi
cconnmanctl> 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")
cconnmanctl> agent on
cconnmanctl> connect wifi_f45eab2f1ee1_6372797774616c_managed_psk
connmanctl> quit
```

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

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

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://github.com/beagleboard/beaglebone-blue/wiki/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.

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.

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.

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

Are there available mechanical models?

A community contributed model is available at https://grabcad.com/library/beaglebone-blue-1

What is the operating temperature range?

`0..70` due to processor, else `-20..70`

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
2.4 BeagleBone AI

Contributors

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
All derivative works are to be attributed to Jason Kridner of BeagleBoard.org.

Note: Make sure to read and accept all the terms & condition provided in the boards-terms-and-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.

2.4.1 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.

2.4.2 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

Rev A2

Proposed changes.
- Add footprint for pull-down resistor on serial debug header RX line.
- Move microSD card cage closer to microHDMI to fit cases better.
- Connect AM5729 ball AB10 to to P9.13 to provide a GPIO.
- HDMI hot-plug detection fixes.
- Add additional CAN port to the expansion headers.
- Fix JTAG connector to not require wire mods.
- Add I2C EEPROM for board identifier.

2.4.3 Connecting Up Your BeagleBone AI

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
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 web sites:

- Antenna
- Heat Sink

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 70°C 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.

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
**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

![USB Type-C connector on BeagleBone® AI](image1)

2. Connect a USB type-C cable to BeagleBone® AI USB type-C port.

![Connecting USB cable to BeagleBone® AI](image2)

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 Cloud9 IDE.

10. Open the directories in the left navigation of Cloud9.
Standalone w/Display and Keyboard/Mouse

**Note:** This configuration requires loading the latest debian 9 image from [https://elinux.org/Beagleboard:Latest-images-testing](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

Presently, the “Cloud 9” application is broken in debian 10 only for this configuration. We are working on a better
solution.

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

**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.
6. Follow the instructions in the browser window.

**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.


Locate the 3 PIN debug header on BeagleBone® AI, near the USB C connection.

![USB 3 Type-C Serial Debug](image)

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)
2.4.4 BeagleBone AI Overview

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
Board Component Locations

2.4.5 BeagleBone AI High Level Specification

This section provides the high level specification of BeagleBone® AI

Block Diagram

The figure below is the high level block diagram of BeagleBone® AI. For detailed layout information please check the schematics.
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*
MPU Subsystem

The Dual Cortex-A15 MPU subsystem integrates the following submodules:

- **ARM Cortex-A15 CPU Core**
  - 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 sub-modules:

- 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](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 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)

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 boot source. The connector will support larger capacity microSD cards. The microSD card is not provided with the board.

Boot Modes

Power Management

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.
## On-board I2C Devices

<table>
<thead>
<tr>
<th>Address</th>
<th>Identiﬁer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12</td>
<td>U3</td>
<td>TPS6590379 PMIC DVS</td>
</tr>
<tr>
<td>0x41</td>
<td>U78</td>
<td>STMPE811Q ADC and GPIO expander</td>
</tr>
<tr>
<td>0x47</td>
<td>U13</td>
<td>HD3SS3220 USB Type-C DRP port controller</td>
</tr>
<tr>
<td>0x50</td>
<td>U9</td>
<td>24LC32 board ID EEPROM</td>
</tr>
<tr>
<td>0x58</td>
<td>U3</td>
<td>TPS6590379 PMIC power registers</td>
</tr>
<tr>
<td>0x5a</td>
<td>U3</td>
<td>TPS6590379 PMIC interfaces and auxiliaries</td>
</tr>
<tr>
<td>0x5c</td>
<td>U3</td>
<td>TPS6590379 PMIC trimming and test</td>
</tr>
<tr>
<td>0x5e</td>
<td>U3</td>
<td>TPS6590379 PMIC OTP</td>
</tr>
</tbody>
</table>

## 2.4.6 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.

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.

![Block Diagram for Power]

### 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.

6.1.2 USB-C Power  Figure 23 below shows how the USB-C power input is connected to the TPS6590379.

TODO: (Schematic screenshot)
Power Button

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>
<thead>
<tr>
<th>Name</th>
<th>Size (bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>4</td>
<td>MSB 0xEE3355AA LSB (stored LSB first)</td>
</tr>
<tr>
<td>Board Name</td>
<td>8</td>
<td>Name for board in ASCII “BBONE-AI” = BeagleBone AI</td>
</tr>
<tr>
<td>Version</td>
<td>4</td>
<td>Hardware version code for board in ASCII “00A1” = rev. A1</td>
</tr>
</tbody>
</table>
| 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
|                  |              | • nnnnn = incrementing board number                                      |

Example:
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

6.9.2 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

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.
PRU-ICSS

The Texas Instruments AM5729 Sitara™ provides 2 Programmable Real-Time Unit Subsystem and Industrial Communication 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 GPs (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
PRU-ICSS Resources and FAQ’s

Resources

- Great resources for PRU and BeagleBone® has been compiled here [https://beagleboard.org/pru](https://beagleboard.org/pru)
- The PRU Cookbook provides examples and getting started information [https://github.com/MarkAYoder/PRUCookbook](https://github.com/MarkAYoder/PRUCookbook)

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>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
<th>PROC</th>
<th>HEADER_PIN</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr1_pru0_gpo0</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A H 6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo1</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A H 3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo2</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A H 5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo3</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A G 6</td>
<td>P 8,12</td>
<td>MODE 13</td>
</tr>
<tr>
<td>pr1_pru0_gpo4</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A H 4</td>
<td>P 8,11</td>
<td>MODE 13</td>
</tr>
<tr>
<td>pr1_pru0_gpo5</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A G 4</td>
<td>P 9,15</td>
<td>MODE 13</td>
</tr>
<tr>
<td>pr1_pru0_gpo6</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A G 2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo7</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A G 3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo8</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A G 5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo9</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A F 2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo10</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A F 6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo11</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A F 3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo12</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A F 4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>SIGNAL NAME</td>
<td>DESCRIPTION</td>
<td>TYPE</td>
<td>PROC</td>
<td>HEADER_PIN</td>
<td>MODE</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------</td>
<td>------</td>
<td>------</td>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>pr1_pru0_gpo13</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>AF1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo14</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>AE3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo15</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>AE5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo16</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>AE1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo17</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>AE2</td>
<td>P9_26</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru0_gpo18</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>AE6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo19</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>AD2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpo20</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>AD3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi0</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AH6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi1</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AH3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi2</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AH5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi3</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AG6</td>
<td>P8_12</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru0_gpi4</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AH4</td>
<td>P8_11</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru0_gpi5</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AG4</td>
<td>P9_15</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru0_gpi6</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AG2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi7</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AG3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi8</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AG5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi9</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AF2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi10</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AF6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi11</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AF3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi12</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AF4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi13</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AF1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi14</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AE3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi15</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AE5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi16</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AE1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi17</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AE2</td>
<td>P9_26</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru0_gpi18</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AE6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi19</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AD2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru0_gpi20</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>AD3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpo0</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>E2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpo1</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>D2</td>
<td>P9_20</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo2</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>F4</td>
<td>P9_19</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo3</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>C1</td>
<td>P9_41</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo4</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>E4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpo5</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>F5</td>
<td>P8_18</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo6</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>E6</td>
<td>P8_19</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo7</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>D3</td>
<td>P8_13</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo8</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>F6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpo9</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>D5</td>
<td>P8_14</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo10</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>C2</td>
<td>P9_42</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo11</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>C3</td>
<td>P9_27</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo12</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>C4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpo13</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>B2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpo14</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>D6</td>
<td>P9_14</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo15</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>C5</td>
<td>P9_16</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo16</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>A3</td>
<td>P8_15</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo17</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>B3</td>
<td>P8_26</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo18</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>B4</td>
<td>P8_16</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr1_pru1_gpo19</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>B5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpo20</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>A4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpi0</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>E2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpi1</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>D2</td>
<td>P9_20</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi2</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>F4</td>
<td>P9_19</td>
<td>MODE12</td>
</tr>
</tbody>
</table>

2.4. BeagleBone AI

139
<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
<th>PROC</th>
<th>HEADER_PIN</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr1_pru1_gpi3</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>C4</td>
<td>P9_41</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi4</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>E4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpi5</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>F5</td>
<td>P8_18</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi6</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>E6</td>
<td>P8_19</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi7</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>D3</td>
<td>P8_13</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi8</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>F6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpi9</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>D5</td>
<td>P8_14</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi10</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>C2</td>
<td>P9_42</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi11</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>C3</td>
<td>P9_27</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi12</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>C4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpi13</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>B2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpi14</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>D6</td>
<td>P8_14</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi15</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>C5</td>
<td>P9_16</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi16</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>A3</td>
<td>P8_15</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi17</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>B3</td>
<td>P8_26</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi18</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>B4</td>
<td>P8_16</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr1_pru1_gpi19</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>B5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_pru1_gpi20</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>A4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_clk</td>
<td>MIIO Transmit Clock</td>
<td>I</td>
<td>U5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_txen</td>
<td>MIIO Transmit Enable</td>
<td>O</td>
<td>V3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_txd3</td>
<td>MIIO Transmit Data</td>
<td>O</td>
<td>V5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_txd2</td>
<td>MIIO Transmit Data</td>
<td>O</td>
<td>V4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_txd1</td>
<td>MIIO Transmit Data</td>
<td>O</td>
<td>Y2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_txd0</td>
<td>MIIO Transmit Data</td>
<td>O</td>
<td>W2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_rxdv</td>
<td>MIIO Data Valid</td>
<td>I</td>
<td>V2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_rxd2</td>
<td>MIIO Receive Data</td>
<td>I</td>
<td>V9</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_crs</td>
<td>MIIO Carrier Sense</td>
<td>I</td>
<td>V7</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_rxer</td>
<td>MIIO Receive Error</td>
<td>I</td>
<td>U7</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_rx1</td>
<td>MIIO Receive Data</td>
<td>I</td>
<td>V6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_rx0</td>
<td>MIIO Receive Data</td>
<td>I</td>
<td>U6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_col</td>
<td>MIIO Collision Detect</td>
<td>I</td>
<td>V1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii0_rxlink</td>
<td>MIIO Receive Link</td>
<td>I</td>
<td>U4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii1_clk</td>
<td>MIII Transmit Clock</td>
<td>I</td>
<td>C1</td>
<td>P9_41</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_txen</td>
<td>MIII Transmit Enable</td>
<td>O</td>
<td>E4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii1_txd3</td>
<td>MIII Transmit Data</td>
<td>O</td>
<td>F5</td>
<td>P8_18</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_txd2</td>
<td>MIII Transmit Data</td>
<td>O</td>
<td>E6</td>
<td>P8_19</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_txd1</td>
<td>MIII Transmit Data</td>
<td>O</td>
<td>D5</td>
<td>P8_14</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_txd0</td>
<td>MIII Transmit Data</td>
<td>O</td>
<td>C2</td>
<td>P9_42</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_rxd0</td>
<td>MIII Receive Data</td>
<td>I</td>
<td>C3</td>
<td>P9_27</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_rxd1</td>
<td>MIII Receive Data</td>
<td>I</td>
<td>C4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii1_rxd3</td>
<td>MIII Receive Data</td>
<td>I</td>
<td>D6</td>
<td>P9_14</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_rxd2</td>
<td>MIII Receive Data</td>
<td>I</td>
<td>D5</td>
<td>P9_16</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_rxer</td>
<td>MIII Receive Error</td>
<td>I</td>
<td>A3</td>
<td>P8_15</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_rxlink</td>
<td>MIII Receive Link</td>
<td>I</td>
<td>B4</td>
<td>P8_16</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mii1_col</td>
<td>MIII Collision Detect</td>
<td>I</td>
<td>B5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mii1_crs</td>
<td>MIII Carrier Sense</td>
<td>I</td>
<td>A4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_mdio_mdclock</td>
<td>MDIO Clock</td>
<td>O</td>
<td>D3</td>
<td>P8_13</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr1_mdio_data</td>
<td>MDIO Data</td>
<td>I</td>
<td>F6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edc_latch0_in</td>
<td>Latch Input 0</td>
<td>I</td>
<td>A3/E2</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>
BeagleBone AI

2.4. BeagleBone AI

Names reveal which PRU-ICSSSubsystem is being addressed. pr1 is PRU-ICSS1 and pr2 is PRU-ICSS2

<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
<th>PROC</th>
<th>HEADER_PIN</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr1_edc_latch1_in</td>
<td>Latch Input 1</td>
<td>I</td>
<td>A G 5</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edc_sync0_out</td>
<td>SYNC0 Output</td>
<td>O</td>
<td>A F 2/D 2</td>
<td>P 9_20</td>
<td>MODE 11</td>
</tr>
<tr>
<td>pr1_edc_sync1_out</td>
<td>SYNC1 Output</td>
<td>O</td>
<td>A F 6</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_latch_in</td>
<td>Latch Input</td>
<td>I</td>
<td>A F 3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_sof</td>
<td>Start Of Frame</td>
<td>O</td>
<td>A F 4/F 4</td>
<td>P 9_19</td>
<td>MODE 11</td>
</tr>
<tr>
<td>pr1_edio_data_in0</td>
<td>Ethernet Digital Input</td>
<td>I</td>
<td>A F 1/E 1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_in1</td>
<td>Ethernet Digital Input</td>
<td>I</td>
<td>A E 3/G 2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_in2</td>
<td>Ethernet Digital Input</td>
<td>I</td>
<td>A E 5/H 7</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_in3</td>
<td>Ethernet Digital Input</td>
<td>I</td>
<td>A E 1/G 1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_in4</td>
<td>Ethernet Digital Input</td>
<td>I</td>
<td>A E 2/G 6</td>
<td>P 9_26</td>
<td>MODE 11</td>
</tr>
<tr>
<td>pr1_edio_data_in5</td>
<td>Ethernet Digital Input</td>
<td>I</td>
<td>A E 6/F 2</td>
<td>P 8_36</td>
<td>MODE 12</td>
</tr>
<tr>
<td>pr1_edio_data_in6</td>
<td>Ethernet Digital Input</td>
<td>I</td>
<td>A D 2/F 3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_out0</td>
<td>Ethernet Digital Output</td>
<td>O</td>
<td>A F 1/E 1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_out1</td>
<td>Ethernet Digital Output</td>
<td>O</td>
<td>A E 3/G 2</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_out2</td>
<td>Ethernet Digital Output</td>
<td>O</td>
<td>A E 5/H 7</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_out3</td>
<td>Ethernet Digital Output</td>
<td>O</td>
<td>A E 1/G 1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_out4</td>
<td>Ethernet Digital Output</td>
<td>O</td>
<td>A E 2/G 6</td>
<td>P 9_26</td>
<td>MODE 11</td>
</tr>
<tr>
<td>pr1_edio_data_out5</td>
<td>Ethernet Digital Output</td>
<td>O</td>
<td>A E 6/F 2</td>
<td>P 8_36</td>
<td>MODE 13</td>
</tr>
<tr>
<td>pr1_edio_data_out6</td>
<td>Ethernet Digital Output</td>
<td>O</td>
<td>A D 2/F 3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr1_edio_data_out7</td>
<td>Ethernet Digital Output</td>
<td>O</td>
<td>A D 3/D 1</td>
<td>P 8_15</td>
<td>MODE 13</td>
</tr>
<tr>
<td>pr1_uart0_cts_n</td>
<td>UART Clear-To-Send</td>
<td>I</td>
<td>G 1/F 11</td>
<td>P 8_45</td>
<td>MODE 10</td>
</tr>
<tr>
<td>pr1_uart0_rts_n</td>
<td>UART Ready-To-Send</td>
<td>O</td>
<td>G 6/G 10</td>
<td>P 8_34</td>
<td>MODE 11</td>
</tr>
<tr>
<td>pr1_uart0_rxd</td>
<td>UART Receive Data</td>
<td>I</td>
<td>F 2/F 10</td>
<td>P 8_36</td>
<td>MODE 11</td>
</tr>
<tr>
<td>pr1_uart0_rxd</td>
<td>UART Transmit Data</td>
<td>O</td>
<td>F 3/G 11</td>
<td>P 8_44</td>
<td>MODE 10</td>
</tr>
<tr>
<td>pr1_ecap0_ecap_capin_apwm_o</td>
<td>Capture Input/PWM Output</td>
<td>TO</td>
<td>D 1/E 9</td>
<td>P 8_15</td>
<td>MODE 11</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
<th>PROC</th>
<th>HEADER_PIN</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr2_pru0_gpo0</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>G 11/AC5</td>
<td>P 8_44</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo1</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>E9/AB4</td>
<td>P 8_41</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo2</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>F9/AD4</td>
<td>P 8_42</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo3</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>F8/AC4</td>
<td>P 8_39</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo4</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>E7/AC7</td>
<td>P 8_40</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo5</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>E8/AC6</td>
<td>P 8_37</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo6</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>D9/AC9</td>
<td>P 8_38</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo7</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>D7/AC3</td>
<td>P 8_36</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo8</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>D8/AC8</td>
<td>P 8_34</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo9</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>A5/AD6</td>
<td>P 8_35</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo10</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>C6/AB8</td>
<td>P 8_33</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo11</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>C8/AB5</td>
<td>P 8_31</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo12</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>C7/B18</td>
<td>P 8_32</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo13</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>B7/F15</td>
<td>P 8_45</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo14</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>B8/B19</td>
<td>P 9_11</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo15</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>A7/C17</td>
<td>P 8_17</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo16</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>A8/C15</td>
<td>P 8_27</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo17</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>C9/A16</td>
<td>P 8_28</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo18</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>A9/A19</td>
<td>P 8_29</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr2_pru0_gpo19</td>
<td>PRU0 Gen-ral-P urpose Output</td>
<td>O</td>
<td>B9/A18</td>
<td>P 8_30</td>
<td>MODE13</td>
</tr>
<tr>
<td>SIGNAL NAME</td>
<td>DESCRIPTION</td>
<td>TYPE</td>
<td>PROC</td>
<td>HEAD ER_PIN</td>
<td>MODE</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>pr_2_pru0_gpo20</td>
<td>PRU0 General-Purpose Output</td>
<td>O</td>
<td>A10/F14</td>
<td>P8_46</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru0_gpi0</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>G11/AC5</td>
<td>P8_44</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi1</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>E9/AB4</td>
<td>P8_41</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi2</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>F9/AD4</td>
<td>P8_42</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi3</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>F8/AC4</td>
<td>P8_39</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi4</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>E7/AC7</td>
<td>P8_40</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi5</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>E8/AC6</td>
<td>P8_37</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi6</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>D9/AC9</td>
<td>P8_38</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi7</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>D7/AC3</td>
<td>P8_36</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi8</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>D8/AC8</td>
<td>P8_34</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi9</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>A5/AD6</td>
<td>P8_35</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi10</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>C6/AB8</td>
<td>P8_33</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi11</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>C8/AB5</td>
<td>P8_31</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi12</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>C7/BI8</td>
<td>P8_32</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi13</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>B7/F15</td>
<td>P8_45</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi14</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>B8/B19</td>
<td>P9_11</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi15</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>A7/CI7</td>
<td>P8_17</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi16</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>A8/CI5</td>
<td>P8_27</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi17</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>C9/AI6</td>
<td>P8_28</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi18</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>A9/AI9</td>
<td>P8_29</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi19</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>B9/AI8</td>
<td>P8_30</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru0_gpi20</td>
<td>PRU0 General-Purpose Input</td>
<td>I</td>
<td>A10/F14</td>
<td>P8_46</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru1_gpo0</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>V1/D17</td>
<td>P8_32</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo1</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>U4/AA3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr_2_pru1_gpo2</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>U3/AB9</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr_2_pru1_gpo3</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>V2/AB3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr_2_pru1_gpo4</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>Y1/AA4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr_2_pru1_gpo5</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>W9/D18</td>
<td>P9_25</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo6</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>V9/E17</td>
<td>P8_9</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo7</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>V7/C14</td>
<td>P9_31</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo8</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>U7/G12</td>
<td>P9_18</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo9</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>V6/F12</td>
<td>P9_17</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo10</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>U6/B12</td>
<td>P9_31</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo11</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>U5/A11</td>
<td>P9_29</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo12</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>V5/B13</td>
<td>P9_30</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo13</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>V4/A12</td>
<td>P9_26</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo14</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>V3/E14</td>
<td>P9_42</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo15</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>Y2/A13</td>
<td>P8_10</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo16</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>W2/G14</td>
<td>P8_7</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo17</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>E11</td>
<td>P8_27</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo18</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>F11</td>
<td>P8_45</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo19</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>G10</td>
<td>P8_46</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpo20</td>
<td>PRU1 General-Purpose Output</td>
<td>O</td>
<td>F10</td>
<td>P8_43</td>
<td>MODE13</td>
</tr>
<tr>
<td>pr_2_pru1_gpi0</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>V1/D17</td>
<td>P8_32</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru1_gpi1</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>U4/AA3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr_2_pru1_gpi2</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>U3/AB9</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr_2_pru1_gpi3</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>V2/AB3</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr_2_pru1_gpi4</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>Y1/AA4</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>pr_2_pru1_gpi5</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>W9/D18</td>
<td>P9_25</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru1_gpi6</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>V9/E17</td>
<td>P8_9</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru1_gpi7</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>V7/C14</td>
<td>P9_31</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru1_gpi8</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>U7/G12</td>
<td>P9_18</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr_2_pru1_gpi9</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>V6/F12</td>
<td>P9_17</td>
<td>MODE12</td>
</tr>
<tr>
<td>SIGNAL NAME</td>
<td>DESCRIPTION</td>
<td>TYPE</td>
<td>PROCESSING</td>
<td>HEAD ER_PIN</td>
<td>MODE</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------</td>
<td>------------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>pr2_pru1_gpi10</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>U6/B12</td>
<td>P9_31</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi11</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>U5/A11</td>
<td>P9_29</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi12</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>V5/B13</td>
<td>P9_30</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi13</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>V4/A12</td>
<td>P9_28</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi14</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>V3/E14</td>
<td>P9_42</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi15</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>Y2/A13</td>
<td>P8_10</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi16</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>W2/G14</td>
<td>P8_7</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi17</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>E11</td>
<td>P8_27</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi18</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>F11</td>
<td>P8_45</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi19</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>G10</td>
<td>P8_46</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_pru1_gpi20</td>
<td>PRU1 General-Purpose Input</td>
<td>I</td>
<td>F10</td>
<td>P8_43</td>
<td>MODE12</td>
</tr>
<tr>
<td>pr2_edc_latch0_in</td>
<td>Latch Input 0</td>
<td>I</td>
<td>F9</td>
<td>P8_42</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_edc_latch1_in</td>
<td>Latch Input 1</td>
<td>I</td>
<td>F8</td>
<td>P8_39</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_edc_sync0_out</td>
<td>SYNC0 Output</td>
<td>O</td>
<td>E7</td>
<td>P8_40</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_edc_sync1_out</td>
<td>SYNC1 Output</td>
<td>O</td>
<td>E8</td>
<td>P8_37</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_edio_latch_in</td>
<td>Latch Input</td>
<td>I</td>
<td>D9</td>
<td>P8_38</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_edio_sof</td>
<td>Start Of Frame</td>
<td>O</td>
<td>D7</td>
<td>P8_36</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_uart0_cts_n</td>
<td>UART Clear-To-Send</td>
<td>I</td>
<td>D8</td>
<td>P8_34</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_uart0_rts_n</td>
<td>UART Ready-To-Send</td>
<td>O</td>
<td>A5</td>
<td>P8_35</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_uart0_rxd</td>
<td>UART Receive Data</td>
<td>I</td>
<td>C6</td>
<td>P8_33</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_uart0_txd</td>
<td>UART Transmit Data</td>
<td>O</td>
<td>C8</td>
<td>P8_30</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_ecap0_ecap_apwm_o</td>
<td>Capture Input/PWM output</td>
<td>IO</td>
<td>C7</td>
<td>P8_32</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_dio_da_tda0</td>
<td>Ethernet Input/Output</td>
<td>I</td>
<td>B7</td>
<td>P8_45</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_dio_da_tda1</td>
<td>Ethernet Input/Output</td>
<td>I</td>
<td>B8</td>
<td>P9_11</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_dio_da_tda2</td>
<td>Ethernet Input/Output</td>
<td>I</td>
<td>A7</td>
<td>P8_17</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_dio_da_tda3</td>
<td>Ethernet Input/Output</td>
<td>I</td>
<td>A8</td>
<td>P8_27</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_dio_da_tda4</td>
<td>Ethernet Input/Output</td>
<td>I</td>
<td>C9</td>
<td>P8_28</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_dio_da_tda5</td>
<td>Ethernet Input/Output</td>
<td>I</td>
<td>A9</td>
<td>P8_29</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_dio_da_tda6</td>
<td>Ethernet Input/Output</td>
<td>I</td>
<td>B9</td>
<td>P8_30</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_dio_da_tda7</td>
<td>Ethernet Input/Output</td>
<td>I</td>
<td>A10</td>
<td>P8_46</td>
<td>MODE10</td>
</tr>
<tr>
<td>pr2_mdio_clock</td>
<td>MDIO Clock</td>
<td>O</td>
<td>C14/AB3</td>
<td>P9_31</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mdio_data</td>
<td>MDIO Data</td>
<td>IO</td>
<td>D14/AA4</td>
<td>P9_29</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_rxer</td>
<td>MIIO Receive Error</td>
<td>I</td>
<td>G12</td>
<td>P9_18</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_tclk</td>
<td>MIIO Transmit Clock</td>
<td>I</td>
<td>F12</td>
<td>P9_17</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_ttxen</td>
<td>MIIO Transmit Enable</td>
<td>O</td>
<td>B12</td>
<td>P9_31</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_txd3</td>
<td>MIIO Transmit Data</td>
<td>O</td>
<td>A11</td>
<td>P9_29</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_txd2</td>
<td>MIIO Transmit Data</td>
<td>O</td>
<td>B13</td>
<td>P9_30</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_txd1</td>
<td>MIIO Transmit Data</td>
<td>O</td>
<td>A12</td>
<td>P9_28</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_txd0</td>
<td>MIIO Transmit Data</td>
<td>O</td>
<td>E14</td>
<td>P9_42</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_clk</td>
<td>MIIO Receive Clock</td>
<td>I</td>
<td>A13</td>
<td>P8_10</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_rxdv</td>
<td>MIIO Data Valid</td>
<td>I</td>
<td>G14</td>
<td>P8_7</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_rxd3</td>
<td>MIIO Receive Data</td>
<td>I</td>
<td>F14</td>
<td>P8_8</td>
<td>MODE11</td>
</tr>
<tr>
<td>pr2_mii0_rxd2</td>
<td>MIIO Receive Data</td>
<td>I</td>
<td>A19</td>
<td>NA</td>
<td>MODE11</td>
</tr>
</tbody>
</table>
### Table 2.23 – continued from previous page

<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
<th>PROC</th>
<th>HEAD ER_PIN</th>
<th>MODE</th>
<th>PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr2_mii0_rxd1</td>
<td>MII0 Receive Data</td>
<td>I</td>
<td>A18</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr2_mii0_rxd0</td>
<td>MII0 Receive Data</td>
<td>I</td>
<td>C15</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr2_mii0_rxlink</td>
<td>MII0 Receive Link</td>
<td>I</td>
<td>A16</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr2_mii0_crs</td>
<td>MII0 Carrier Sense</td>
<td>I</td>
<td>B18</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr2_mii0_col</td>
<td>MII0 Collision Detect</td>
<td>I</td>
<td>F15</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_rxer</td>
<td>MII1 Receive Error</td>
<td>I</td>
<td>B19</td>
<td>P9_11</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_rxlink</td>
<td>MII1 Receive Link</td>
<td>I</td>
<td>C17</td>
<td>P9_13</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_m11_clk</td>
<td>MII1 Transmit Clock</td>
<td>I</td>
<td>AC5</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_txden</td>
<td>MII1 Transmit Enable</td>
<td>O</td>
<td>AB4</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_txd3</td>
<td>MII1 Transmit Data</td>
<td>O</td>
<td>AD4</td>
<td>P8_21</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_txd2</td>
<td>MII1 Transmit Data</td>
<td>O</td>
<td>AC4</td>
<td>P8_20</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_txd1</td>
<td>MII1 Transmit Data</td>
<td>O</td>
<td>AC7</td>
<td>P8_25</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_txd0</td>
<td>MII1 Transmit Data</td>
<td>O</td>
<td>AC6</td>
<td>P8_24</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii_m11_clk</td>
<td>MII1 Receive Clock</td>
<td>I</td>
<td>AC9</td>
<td>P8_5</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_rxdv</td>
<td>MII1 Data Valid</td>
<td>I</td>
<td>AC3</td>
<td>P8_6</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_rxd3</td>
<td>MII1 Receive Data</td>
<td>I</td>
<td>AC8</td>
<td>P8_23</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_rxd2</td>
<td>MII1 Receive Data</td>
<td>I</td>
<td>AD6</td>
<td>P8_22</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_rxd1</td>
<td>MII1 Receive Data</td>
<td>I</td>
<td>AB8</td>
<td>P8_3</td>
<td>MODE11</td>
<td></td>
</tr>
<tr>
<td>pr2_mii1_rxd0</td>
<td>MII1 Receive Data</td>
<td>I</td>
<td>AB5</td>
<td>P8_4</td>
<td>MODE11</td>
<td></td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>LED</th>
<th>GPIO SIGNAL</th>
<th>DEFAULT FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2</td>
<td>GPIO3_17</td>
<td>Heartbeat When Linux is Running</td>
</tr>
<tr>
<td>D3</td>
<td>GPIO5_5</td>
<td>microSD Activity</td>
</tr>
<tr>
<td>D4</td>
<td>GPIO3_15</td>
<td>CPU Activity</td>
</tr>
<tr>
<td>D5</td>
<td>GPIO3_14</td>
<td>eMMC Activity</td>
</tr>
<tr>
<td>D8</td>
<td>GPIO3_7</td>
<td>WiFi/Bluetooth Activity</td>
</tr>
</tbody>
</table>
2.4.7 Connectors

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.

**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 and P8.02 are ground pins.
### P8.03-P8.05

<table>
<thead>
<tr>
<th>P8.03</th>
<th>P8.04</th>
<th>P8.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>BALL</td>
<td>AB8</td>
<td>AB5</td>
</tr>
<tr>
<td>REG</td>
<td>0x179C</td>
<td>0x17A0</td>
</tr>
<tr>
<td>MODE 0</td>
<td>mmc3_dat6</td>
<td>mmc3_dat7</td>
</tr>
<tr>
<td>1</td>
<td>spi4_d0</td>
<td>spi4_cs0</td>
</tr>
<tr>
<td>2</td>
<td>uart10_ctsn</td>
<td>uart10_rtsn</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>vin2b_de1</td>
<td>vin2b_clk1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>vin5a_hsync0</td>
<td>vin5a_vsync0</td>
</tr>
<tr>
<td>10</td>
<td>ehrpwm3_tripzone_input</td>
<td>eCAP3_in_PWM3_out</td>
</tr>
<tr>
<td>11</td>
<td>pr2_mii1_rxd1</td>
<td>pr2_mii1_rxd0</td>
</tr>
<tr>
<td>12</td>
<td>pr2_pru0_gpi10</td>
<td>pr2_pru0_gpi11</td>
</tr>
<tr>
<td>13</td>
<td>pr2_pru0_gpo10</td>
<td>pr2_pru0_gpo11</td>
</tr>
<tr>
<td>14</td>
<td>gpio1_24</td>
<td>gpio1_25</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P8.06-P8.09

<table>
<thead>
<tr>
<th>P8.06</th>
<th>P8.07</th>
<th>P8.08</th>
<th>P8.09</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>194</td>
<td>165</td>
<td>166</td>
</tr>
<tr>
<td>BALL</td>
<td>AC3</td>
<td>G14</td>
<td>F14</td>
</tr>
<tr>
<td>REG</td>
<td>0x1790</td>
<td>0x16EC</td>
<td>0x16F0</td>
</tr>
<tr>
<td>MODE 0</td>
<td>mmc3_dat3</td>
<td>mcasp1_axr14</td>
<td>mcasp1_axr15</td>
</tr>
<tr>
<td>1</td>
<td>spi3_cs1</td>
<td>mcasp7_axr14</td>
<td>mcasp7_axr15</td>
</tr>
<tr>
<td>2</td>
<td>uart5_rtsn</td>
<td>mcasp7_axrkr</td>
<td>mcasp7_fsr</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>mcasp2_axclkx</td>
</tr>
<tr>
<td>4</td>
<td>vin2b_d2</td>
<td></td>
<td>mcasp6_axclkx</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>vin6a_d9</td>
<td>vin6a_d8</td>
<td>vin6a_clk0</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>vin5a_d2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>cQEP3_strobe</td>
<td>timer11</td>
<td>timer12</td>
</tr>
<tr>
<td>11</td>
<td>pr2_mii1_rxdv</td>
<td>pr2_mii0_rxdv</td>
<td>pr2_mii0_rxd3</td>
</tr>
<tr>
<td>12</td>
<td>pr2_pru0_gpi17</td>
<td>pr2_pru1_gpi16</td>
<td>pr2_pru0_gpi20</td>
</tr>
<tr>
<td>13</td>
<td>pr2_pru0_gpo7</td>
<td>pr2_pru1_gpo16</td>
<td>pr2_pru0_gpo20</td>
</tr>
<tr>
<td>14</td>
<td>gpio7_2</td>
<td>gpio6_5</td>
<td>gpio6_6</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

---

### 2.4. BeagleBone AI

147
<table>
<thead>
<tr>
<th>P8.10-P8.13</th>
<th>P8.10</th>
<th>P8.11</th>
<th>P8.12</th>
<th>P8.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>164</td>
<td>75</td>
<td>74</td>
<td>107</td>
</tr>
<tr>
<td>BALL</td>
<td>A13</td>
<td>A14</td>
<td>AG6</td>
<td>D3</td>
</tr>
<tr>
<td>REG</td>
<td>0x16E8</td>
<td>0x1510</td>
<td>0x150C</td>
<td>0x1590</td>
</tr>
<tr>
<td>MODE 0</td>
<td>mcsap1_axr13</td>
<td>vin1a_d7</td>
<td>vin1a_d6</td>
<td>vin2a_d10</td>
</tr>
<tr>
<td>1</td>
<td>mcsap7_axr1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>vout3_d0</td>
<td>vout3_d1</td>
<td>mdio_mclk</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>vout3_d16</td>
<td>vout3_d17</td>
<td>vout2_d13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>vin6a_d10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>timer10</td>
<td>eQEP2B_in</td>
<td>eQEP2A_in</td>
<td>ehrpw2B</td>
</tr>
<tr>
<td>11</td>
<td>pr2_mii_mr0_clk</td>
<td>pr1_mdio_mclk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>pr2_prul_gpi15</td>
<td>prl_prul0_gpi4</td>
<td>prl_prul0_gpi3</td>
<td>prl_prul1_gpi7</td>
</tr>
<tr>
<td>13</td>
<td>pr2_prul_gpo15</td>
<td>prl_prul0_gpo4</td>
<td>prl_prul0_gpo3</td>
<td>prl_prul1_gpo7</td>
</tr>
<tr>
<td>14</td>
<td>gpio6_4</td>
<td>gpio3_11</td>
<td>gpio3_10</td>
<td>gpio4_11</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P8.14-P8.16</th>
<th>P8.14</th>
<th>P8.15</th>
<th>P8.16</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>109</td>
<td>99</td>
<td>125</td>
</tr>
<tr>
<td>BALL</td>
<td>D5</td>
<td>D1</td>
<td>B4</td>
</tr>
<tr>
<td>REG</td>
<td>0x1598</td>
<td>0x1570</td>
<td>0x15BC</td>
</tr>
<tr>
<td>MODE 0</td>
<td>vin2a_d12</td>
<td>vin2a_d2</td>
<td>vin2a_d21</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>rgmii1_txc</td>
<td>rgmii1_rxd2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>vout2_d11</td>
<td>vout2_d21</td>
<td>vout2_d2</td>
</tr>
<tr>
<td>5</td>
<td>emu12</td>
<td>vin3a_fld0</td>
<td>vin3a_d13</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>miil1_rxclk</td>
<td>uart10_rxd</td>
<td>miil1_col</td>
</tr>
<tr>
<td>9</td>
<td>kdb_col8</td>
<td>kbd_row6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>eCAP2_in_PWM2_out</td>
<td>eCAP1_in_PWM1_out</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>prl_mii1_txd1</td>
<td>prl_ecap0_ecap_capin_apwm_o</td>
<td>prl_mii1_rxlk</td>
</tr>
<tr>
<td>12</td>
<td>prl_prul_gpi9</td>
<td>prl_edio_data_in7</td>
<td>prl_prul_gpi18</td>
</tr>
<tr>
<td>13</td>
<td>prl_prul_gpo9</td>
<td>prl_edio_data_out7</td>
<td>prl_prul_gpo18</td>
</tr>
<tr>
<td>14</td>
<td>gpio4_13</td>
<td>gpio4_3</td>
<td>gpio4_29</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2nd BALL</td>
<td>A3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd REG</td>
<td>0x15B4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd MODE 0</td>
<td>vin2a_d19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 3</td>
<td>rgmii1_rxctl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 4</td>
<td>vout2_d4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 8</td>
<td>miil1_txer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
### Table 2.24 – continued from previous page

<table>
<thead>
<tr>
<th>2nd 10</th>
<th>2nd 11</th>
<th>2nd 12</th>
<th>2nd 13</th>
<th>2nd 14</th>
<th>2nd 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.14</td>
<td>ehrpwm3_tripzone_input</td>
<td>pr1_mii1_rxd0</td>
<td>pr1_prul_gpi16</td>
<td>pr1_prul_gpo16</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P8.14-P8.16

<table>
<thead>
<tr>
<th>2nd 17</th>
<th>2nd 18</th>
<th>2nd 19</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.17</td>
<td>P8.18</td>
<td>P8.19</td>
</tr>
<tr>
<td>GPIO</td>
<td>242</td>
<td>105</td>
</tr>
<tr>
<td>BALL</td>
<td>A7</td>
<td>F5</td>
</tr>
<tr>
<td>REG</td>
<td>0x1624</td>
<td>0x1588</td>
</tr>
<tr>
<td>MODE 0</td>
<td>vout1_d18</td>
<td>vin2a_d8</td>
</tr>
<tr>
<td>1</td>
<td>emu4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>vin4a_d2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>vout2_d15</td>
<td>vout2_d14</td>
</tr>
<tr>
<td>4</td>
<td>obs11</td>
<td>emu18</td>
</tr>
<tr>
<td>5</td>
<td>obs27</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>mii1_rxd3</td>
<td>mii1_rxd0</td>
</tr>
<tr>
<td>9</td>
<td>kbd_col5</td>
<td>kbd_col6</td>
</tr>
<tr>
<td>10</td>
<td>pr2_edio_data_in2</td>
<td>eQEP2_strobe</td>
</tr>
<tr>
<td>11</td>
<td>pr2_edio_data_out2</td>
<td>pr1_mii1_txd3</td>
</tr>
<tr>
<td>12</td>
<td>pr2_prul_gpi15</td>
<td>pr1_prul_gpi5</td>
</tr>
<tr>
<td>13</td>
<td>pr2_prul_gpo15</td>
<td>pr1_prul_gpo5</td>
</tr>
<tr>
<td>14</td>
<td>gpio8_18</td>
<td>gpio4_9</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P8.17-P8.19

<table>
<thead>
<tr>
<th>2nd 20</th>
<th>2nd 21</th>
<th>2nd 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.20</td>
<td>P8.21</td>
<td>P8.22</td>
</tr>
<tr>
<td>GPIO</td>
<td>190</td>
<td>189</td>
</tr>
<tr>
<td>BALL</td>
<td>AC4</td>
<td>AD4</td>
</tr>
<tr>
<td>REG</td>
<td>0x1780</td>
<td>0x177C</td>
</tr>
<tr>
<td>MODE 0</td>
<td>mmc3_cmd</td>
<td>mmc3_clk</td>
</tr>
<tr>
<td>1</td>
<td>spi3_sclk</td>
<td>spi4_d1</td>
</tr>
<tr>
<td>2</td>
<td>uart10_txd</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>vin2b_d6</td>
<td>vin2b_d7</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>vin5a_d6</td>
<td>vin5a_d7</td>
</tr>
<tr>
<td>10</td>
<td>eCAP2_in_PWM2_out</td>
<td>ehrpwm2_tripzone_input</td>
</tr>
<tr>
<td>11</td>
<td>pr2_mii1_txd2</td>
<td>pr2_mii1_txd3</td>
</tr>
<tr>
<td>12</td>
<td>pr2_prul_gpi3</td>
<td>pr2_prul_gpi2</td>
</tr>
<tr>
<td>13</td>
<td>pr2_prul_gpo3</td>
<td>pr2_prul_gpo2</td>
</tr>
<tr>
<td>14</td>
<td>gpio6_30</td>
<td>gpio6_29</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>
### Table 8.23-P8.26

<table>
<thead>
<tr>
<th></th>
<th>P8.23</th>
<th>P8.24</th>
<th>P8.25</th>
<th>P8.26</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>22</td>
<td>192</td>
<td>191</td>
<td>124</td>
</tr>
<tr>
<td>BALL</td>
<td>AC8</td>
<td>AC6</td>
<td>AC7</td>
<td>B3</td>
</tr>
<tr>
<td>REG</td>
<td>0x1794</td>
<td>0x1788</td>
<td>0x1784</td>
<td>0x15B8</td>
</tr>
<tr>
<td>MODE</td>
<td>mmc3_dat4</td>
<td>mmc3_dat1</td>
<td>mmc3_dat0</td>
<td>vin2a_d20</td>
</tr>
<tr>
<td>1</td>
<td>spi4_sclk</td>
<td>spi3_d0</td>
<td>spi3_d1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uart10_rxd</td>
<td>uart5_txd</td>
<td>uart5_rxd</td>
<td>vin2b_d3</td>
</tr>
<tr>
<td>3</td>
<td>rgmi1_rxd3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>vin2b_d1</td>
<td>vin2b_d4</td>
<td>vin2b_d5</td>
<td>vout2_d3</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>vin3a_de0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>vin3a_d12</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>vin5a_d1</td>
<td>vin5a_d4</td>
<td>vin5a_d5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ehrpwm3A</td>
<td>eQEP3B_in</td>
<td>eQEP3A_in</td>
<td>eCAP3_in_PWM3_out</td>
</tr>
<tr>
<td>11</td>
<td>pr2_mii1_rxd3</td>
<td>pr2_mii1_txd0</td>
<td>pr2_mii1_txd1</td>
<td>pr1_mii1_rxer</td>
</tr>
<tr>
<td>12</td>
<td>pr2_pru0_gpi8</td>
<td>pr2_pru0_gpi5</td>
<td>pr2_pru0_gpi4</td>
<td>pr1_pru1_gpi17</td>
</tr>
<tr>
<td>13</td>
<td>pr2_pru0_gpo8</td>
<td>pr2_pru0_gpo5</td>
<td>pr2_pru0_gpo4</td>
<td>pr1_pru1_gpo17</td>
</tr>
<tr>
<td>14</td>
<td>gpio1_22</td>
<td>gpio7_0</td>
<td>gpio6_31</td>
<td>gpio4_28</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### Table 8.27-P8.29

<table>
<thead>
<tr>
<th></th>
<th>P8.27</th>
<th>P8.28</th>
<th>P8.29</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>119</td>
<td>115</td>
<td>118</td>
</tr>
<tr>
<td>BALL</td>
<td>E11</td>
<td>D11</td>
<td>C11</td>
</tr>
<tr>
<td>REG</td>
<td>0x15D8</td>
<td>0x15C8</td>
<td>0x15D4</td>
</tr>
<tr>
<td>MODE</td>
<td>vout1_vsync</td>
<td>vout1_clk</td>
<td>vout1_hsync</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>vin4a_vsync0</td>
<td>vin4a_fld0</td>
<td>vin4a_hsync0</td>
</tr>
<tr>
<td>4</td>
<td>vin3a_vsync0</td>
<td>vin3a_fld0</td>
<td>vin3a_hsync0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>spi3_sclk</td>
<td>spi3_cs0</td>
<td>spi3_d0</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>pr2_pru1_gpi17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>pr2_pru1_gpo17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>gpio4_23</td>
<td>gpio4_19</td>
<td>gpio4_22</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2nd BALL</td>
<td>A8</td>
<td>C9</td>
<td>A9</td>
</tr>
<tr>
<td>2nd REG</td>
<td>0x1628</td>
<td>0x162C</td>
<td>0x1630</td>
</tr>
<tr>
<td>2nd MODE</td>
<td>vout1_d19</td>
<td>vout1_d20</td>
<td>vout1_d21</td>
</tr>
<tr>
<td>2nd 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 2</td>
<td>emu15</td>
<td>emu16</td>
<td>emu17</td>
</tr>
<tr>
<td>2nd 3</td>
<td>vin4a_d3</td>
<td>vin4a_d4</td>
<td>vin4a_d5</td>
</tr>
<tr>
<td>2nd 4</td>
<td>vin3a_d3</td>
<td>vin3a_d4</td>
<td>vin3a_d5</td>
</tr>
<tr>
<td>2nd 5</td>
<td>obs12</td>
<td>obs13</td>
<td>obs14</td>
</tr>
<tr>
<td>2nd 6</td>
<td>obs28</td>
<td>obs29</td>
<td>obs30</td>
</tr>
<tr>
<td>2nd 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
## Table 2.25 – continued from previous page

<table>
<thead>
<tr>
<th>2nd 10</th>
<th>pr2edio_data_in3</th>
<th>pr2edio_data_in4</th>
<th>pr2edio_data_in5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd 11</td>
<td>pr2edio_data_out3</td>
<td>pr2edio_data_out4</td>
<td>pr2edio_data_out5</td>
</tr>
<tr>
<td>2nd 12</td>
<td>pr2pru0_gpi16</td>
<td>pr2pru0_gpi17</td>
<td>pr2pru0_gpi18</td>
</tr>
<tr>
<td>2nd 13</td>
<td>pr2pru0_gpo16</td>
<td>pr2pru0_gpo17</td>
<td>pr2pru0_gpo18</td>
</tr>
<tr>
<td>2nd 14</td>
<td>gpio8_19</td>
<td>gpio8_20</td>
<td>gpio8_21</td>
</tr>
<tr>
<td>2nd 15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P8.27-P8.29

<table>
<thead>
<tr>
<th>GPIO</th>
<th>P8.30</th>
<th>P8.31</th>
<th>P8.32</th>
</tr>
</thead>
<tbody>
<tr>
<td>116</td>
<td>238</td>
<td>239</td>
<td></td>
</tr>
<tr>
<td>B10</td>
<td>C8</td>
<td>C7</td>
<td></td>
</tr>
<tr>
<td>0x15CC</td>
<td>0x1614</td>
<td>0x1618</td>
<td></td>
</tr>
<tr>
<td>vout1_de</td>
<td>vout1_d14</td>
<td>vout1_d15</td>
<td></td>
</tr>
</tbody>
</table>

### P8.30-P8.32

<table>
<thead>
<tr>
<th>BALL</th>
<th>REG</th>
<th>MODE 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B9</td>
<td>0x1634</td>
<td>0x173C</td>
</tr>
<tr>
<td>G16</td>
<td>0x1740</td>
<td></td>
</tr>
<tr>
<td>D17</td>
<td>0x1740</td>
<td></td>
</tr>
<tr>
<td>Pr2uart0_txd</td>
<td>Pr2ecap0_ecap_capin_apwm_o</td>
<td></td>
</tr>
<tr>
<td>Pr2pru0_gpi11</td>
<td>Pr2pru0_gpi12</td>
<td></td>
</tr>
<tr>
<td>Pr2pru0_gpo11</td>
<td>Pr2pru0_gpo12</td>
<td></td>
</tr>
<tr>
<td>GPIO4_20</td>
<td>GPIO8_14</td>
<td>GPIO8_15</td>
</tr>
<tr>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P8.30-P8.32

<table>
<thead>
<tr>
<th>2nd BALL</th>
<th>2nd REG</th>
<th>2nd MODE 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B9</td>
<td>0x1634</td>
<td>0x173C</td>
</tr>
<tr>
<td>G16</td>
<td>0x1740</td>
<td>0x1740</td>
</tr>
<tr>
<td>Pr2uart0_txd</td>
<td>Pr2ecap0_ecap_capin_apwm_o</td>
<td></td>
</tr>
<tr>
<td>Pr2pru0_gpi11</td>
<td>Pr2pru0_gpi12</td>
<td></td>
</tr>
<tr>
<td>Pr2pru0_gpo11</td>
<td>Pr2pru0_gpo12</td>
<td></td>
</tr>
<tr>
<td>GPIO4_20</td>
<td>GPIO8_14</td>
<td>GPIO8_15</td>
</tr>
<tr>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P8.30-P8.32

![Image](image_url)
<table>
<thead>
<tr>
<th>GPIO</th>
<th>P8.33</th>
<th>P8.34</th>
<th>P8.35</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6</td>
<td>237</td>
<td>235</td>
<td>236</td>
</tr>
<tr>
<td>D8</td>
<td>0x1610</td>
<td>0x1608</td>
<td>0x160C</td>
</tr>
<tr>
<td>A5</td>
<td>vout1_d13</td>
<td>vout1_d11</td>
<td>vout1_d12</td>
</tr>
<tr>
<td>1</td>
<td>emu12</td>
<td>emu10</td>
<td>emu11</td>
</tr>
<tr>
<td>2</td>
<td>vin4a_d13</td>
<td>vin4a_d11</td>
<td>vin4a_d12</td>
</tr>
<tr>
<td>3</td>
<td>vin3a_d13</td>
<td>vin3a_d11</td>
<td>vin3a_d12</td>
</tr>
<tr>
<td>4</td>
<td>obs8</td>
<td>obs6</td>
<td>obs7</td>
</tr>
<tr>
<td>5</td>
<td>obs24</td>
<td>obs22</td>
<td>obs23</td>
</tr>
<tr>
<td>6</td>
<td>obs_dmarq2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>pr2_uart0_rxd</td>
<td>pr2_uart0_cts_n</td>
<td>pr2_uart0_rts_n</td>
</tr>
<tr>
<td>11</td>
<td>pr2_pr0_gpi10</td>
<td>pr2_pr0_gpi8</td>
<td>pr2_pr0_gpi9</td>
</tr>
<tr>
<td>12</td>
<td>pr2_pr0_gpo10</td>
<td>pr2_pr0_gpo8</td>
<td>pr2_pr0_gpo9</td>
</tr>
<tr>
<td>13</td>
<td>gpio8_13</td>
<td>gpio8_11</td>
<td>gpio8_12</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd BALL</td>
<td>AF9</td>
<td>G6</td>
<td>AD9</td>
</tr>
<tr>
<td>2nd REG</td>
<td>0x14E8</td>
<td>0x1564</td>
<td>0x14E4</td>
</tr>
<tr>
<td>2nd MODE0</td>
<td>vin1a_fld0</td>
<td>vin2a_vsnc0</td>
<td>vin1a_de0</td>
</tr>
<tr>
<td>2nd 1</td>
<td>vin1b_vsnc1</td>
<td></td>
<td>vin1b_vsnc1</td>
</tr>
<tr>
<td>2nd 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 3</td>
<td>vin2b_vsnc1</td>
<td></td>
<td>vout3_d17</td>
</tr>
<tr>
<td>2nd 4</td>
<td>vout3_clk</td>
<td>vout2_vsnc</td>
<td>vout3_de</td>
</tr>
<tr>
<td>2nd 5</td>
<td>uart7_txd</td>
<td>emu9</td>
<td>uart7_rxd</td>
</tr>
<tr>
<td>2nd 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 7</td>
<td>timer15</td>
<td>uart9_txd</td>
<td>timer16</td>
</tr>
<tr>
<td>2nd 8</td>
<td>spi3_d1</td>
<td>spi4_d1</td>
<td>spi3_sclk</td>
</tr>
<tr>
<td>2nd 9</td>
<td>kbd_row1</td>
<td>kbd_row3</td>
<td>kbd_row0</td>
</tr>
<tr>
<td>2nd 10</td>
<td>eQEP1B_in</td>
<td>ehrpwm1A</td>
<td>eQEP1A_in</td>
</tr>
<tr>
<td>2nd 11</td>
<td>pr1_uart0_rts_n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 12</td>
<td>pr1_edio_data_in4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 13</td>
<td>pr1_edio_data_out4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 14</td>
<td>gpio3_1</td>
<td>gpio4_0</td>
<td>gpio3_0</td>
</tr>
<tr>
<td>2nd 15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

P8.33-P8.35

<table>
<thead>
<tr>
<th>GPIO</th>
<th>P8.36</th>
<th>P8.37</th>
<th>P8.38</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6</td>
<td>234</td>
<td>232</td>
<td>233</td>
</tr>
<tr>
<td>D7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D8</td>
<td>0x1604</td>
<td>0x15FC</td>
<td>0x1600</td>
</tr>
<tr>
<td>A5</td>
<td>vout1_d10</td>
<td>vout1_d8</td>
<td>vout1_d9</td>
</tr>
<tr>
<td>1</td>
<td>emu3</td>
<td>uart6_rxd</td>
<td>uart6_txd</td>
</tr>
<tr>
<td>2</td>
<td>vin4a_d10</td>
<td>vin4a_d8</td>
<td>vin4a_d9</td>
</tr>
<tr>
<td>3</td>
<td>vin3a_d10</td>
<td>vin3a_d8</td>
<td>vin3a_d9</td>
</tr>
<tr>
<td>4</td>
<td>obs5</td>
<td>obs6</td>
<td>obs7</td>
</tr>
<tr>
<td>5</td>
<td>obs21</td>
<td>obs22</td>
<td>obs23</td>
</tr>
<tr>
<td>6</td>
<td>obs_irq2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 2.28 – continued from previous page

<table>
<thead>
<tr>
<th></th>
<th>P8.36</th>
<th>P8.37</th>
<th>P8.38</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>pr2_edio_sof</td>
<td>pr2_edc_sync1_out</td>
<td>pr2_edio_latch_in</td>
</tr>
<tr>
<td>10</td>
<td>pr2_pru0_gpi7</td>
<td>pr2_pru0_gpi5</td>
<td>pr2_pru0_gpi6</td>
</tr>
<tr>
<td>11</td>
<td>pr2_pru0_gpo7</td>
<td>pr2_pru0_gpo5</td>
<td>pr2_pru0_gpo6</td>
</tr>
<tr>
<td>14</td>
<td>gpio8_10</td>
<td>gpio8_8</td>
<td>gpio8_9</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2nd BALL</td>
<td>F2</td>
<td>A21</td>
<td>C18</td>
</tr>
<tr>
<td>2nd REG</td>
<td>0x1568</td>
<td>0x1738</td>
<td>0x1734</td>
</tr>
<tr>
<td>2nd MODE 0</td>
<td>vin2a_d0</td>
<td>mcsap4_fsx</td>
<td>mcsap4_aclkx</td>
</tr>
<tr>
<td>2nd 1</td>
<td>mcsap4_fsr</td>
<td>mcsap4_aclkkr</td>
<td></td>
</tr>
<tr>
<td>2nd 2</td>
<td>spi3_d1</td>
<td>spi3_scl</td>
<td></td>
</tr>
<tr>
<td>2nd 3</td>
<td>uart8_txd</td>
<td>uart8_rxdl</td>
<td></td>
</tr>
<tr>
<td>2nd 4</td>
<td>vout2_d23</td>
<td>i2c4_scl</td>
<td>i2c4_sda</td>
</tr>
<tr>
<td>2nd 5</td>
<td>emu10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 6</td>
<td>vout2_d17</td>
<td>vout2_d16</td>
<td></td>
</tr>
<tr>
<td>2nd 7</td>
<td>uart9_ctsn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 8</td>
<td>spi4_d0</td>
<td>vin4a_d17</td>
<td>vin4a_d16</td>
</tr>
<tr>
<td>2nd 9</td>
<td>kbd_row4</td>
<td>vin5a_d14</td>
<td>vin5a_d15</td>
</tr>
<tr>
<td>2nd 10</td>
<td>ehrpwm1B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 11</td>
<td>pr1_uart0_rxd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 12</td>
<td>pr1_edio_data_in5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 13</td>
<td>pr1_edio_data_out5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 14</td>
<td>gpio4_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

**P8.36-P8.38**

**P8.39-P8.41**

<table>
<thead>
<tr>
<th></th>
<th>P8.39</th>
<th>P8.40</th>
<th>P8.41</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>GPIO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>BALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>REG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>MODE 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>810</td>
<td>pr2_edc_latch1_in</td>
<td>pr2_edc_sync0_out</td>
<td>pr1_ecap0_ecap_capin_apwm_o</td>
</tr>
<tr>
<td>811</td>
<td>pr2_pru0_gpi3</td>
<td>pr2_pru0_gpi4</td>
<td>pr2_pru0_gpi1</td>
</tr>
<tr>
<td>812</td>
<td>pr2_pru0_gpo3</td>
<td>pr2_pru0_gpo4</td>
<td>pr2_pru0_gpo1</td>
</tr>
<tr>
<td>814</td>
<td>gpio8_6</td>
<td>gpio8_7</td>
<td>gpio8_4</td>
</tr>
<tr>
<td>815</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>
### P8.42-P8.44

<table>
<thead>
<tr>
<th></th>
<th>P8.42</th>
<th>P8.43</th>
<th>P8.44</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>229</td>
<td>226</td>
<td>227</td>
</tr>
<tr>
<td>BALL</td>
<td>F9</td>
<td>F10</td>
<td>G11</td>
</tr>
<tr>
<td>REG</td>
<td>0x15F0</td>
<td>0x15E4</td>
<td>0x15E8</td>
</tr>
<tr>
<td>MODE</td>
<td>vout1_d5</td>
<td>vout1_d2</td>
<td>vout1_d3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>emu7</td>
<td>emu2</td>
<td>emu5</td>
</tr>
<tr>
<td>3</td>
<td>vin4a_d21</td>
<td>vin4a_d18</td>
<td>vin4a_d19</td>
</tr>
<tr>
<td>4</td>
<td>vin3a_d21</td>
<td>vin3a_d18</td>
<td>vin3a_d19</td>
</tr>
<tr>
<td>5</td>
<td>obs3</td>
<td>obs0</td>
<td>obs1</td>
</tr>
<tr>
<td>6</td>
<td>obs19</td>
<td>obs16</td>
<td>obs17</td>
</tr>
<tr>
<td>7</td>
<td>obs_irq1</td>
<td>obs_dmarq1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>pr2_edc_latch0_in</td>
<td>pr1_uart0_rxd</td>
<td>pr1_uart0_txd</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>pr2_pru0_gpi2</td>
<td>pr2_pru1_gpi20</td>
<td>pr2_pru0_gpi0</td>
</tr>
<tr>
<td>12</td>
<td>pr2_pru0_gpo2</td>
<td>pr2_pru1_gpo20</td>
<td>pr2_pru0_gpo0</td>
</tr>
<tr>
<td>13</td>
<td>gpio8_5</td>
<td>gpio8_2</td>
<td>gpio8_3</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P8.45-P8.46

<table>
<thead>
<tr>
<th></th>
<th>P8.45</th>
<th>P8.46</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>224</td>
<td>225</td>
</tr>
<tr>
<td>BALL</td>
<td>F11</td>
<td>G10</td>
</tr>
<tr>
<td>REG</td>
<td>0x15DC</td>
<td>0x15E0</td>
</tr>
<tr>
<td>MODE</td>
<td>vout1_d0</td>
<td>vout1_d1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uart5_rxd</td>
<td>uart5_txd</td>
</tr>
<tr>
<td>3</td>
<td>vin4a_d16</td>
<td>vin4a_d17</td>
</tr>
<tr>
<td>4</td>
<td>vin3a_d16</td>
<td>vin3a_d17</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>spi3_cs2</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>pr1_uart0_cts_n</td>
<td>pr1_uart0_rts_n</td>
</tr>
<tr>
<td>11</td>
<td>pr2_pru1_gpi18</td>
<td>pr2_pru1_gpi19</td>
</tr>
<tr>
<td>12</td>
<td>pr2_pru1_gpo18</td>
<td>pr2_pru1_gpo19</td>
</tr>
<tr>
<td>13</td>
<td>gpio8_0</td>
<td>gpio8_1</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2nd BALL</td>
<td>B7</td>
<td>A10</td>
</tr>
<tr>
<td>2nd REG</td>
<td>0x161C</td>
<td>0x1638</td>
</tr>
<tr>
<td>2nd MODE</td>
<td>vout1_d16</td>
<td>vout1_d23</td>
</tr>
<tr>
<td>2nd 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 2.29 – continued from previous page

<table>
<thead>
<tr>
<th>2nd 10</th>
<th>P8.45</th>
<th>P8.46</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr2_edio_data_in0</td>
<td>pr2_edio_data_in7</td>
<td></td>
</tr>
<tr>
<td>pr2_edio_data_out0</td>
<td>pr2_edio_data_out7</td>
<td></td>
</tr>
<tr>
<td>pr2_pru0_gpi13</td>
<td>pr2_pru0_gpi20</td>
<td></td>
</tr>
<tr>
<td>pr2_pru0_gpo13</td>
<td>pr2_pru0_gpo20</td>
<td></td>
</tr>
<tr>
<td>gpio8_16</td>
<td>gpio8_23</td>
<td></td>
</tr>
<tr>
<td>Driver off</td>
<td>Driver off</td>
<td></td>
</tr>
</tbody>
</table>

**P8.45-P8.46** 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.)

**P9.01-P9.04**

**P9.05-P9.08**

**P9.09-P9.10**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>241</td>
<td>128</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td>B19</td>
<td>B14</td>
<td>C17</td>
<td></td>
</tr>
<tr>
<td>0x172C</td>
<td>0x16AC</td>
<td>0x1730</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>MODE 0</th>
<th>P9.11</th>
<th>P9.12</th>
<th>P9.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mcasp3_axr0</td>
<td>mcasp1_acikr</td>
<td>mcasp3_axr1</td>
</tr>
<tr>
<td>2</td>
<td>mcasp2_axr14</td>
<td>mcasp2_axr15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uart7_ctsn</td>
<td>uart7_rtsn</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uart5_rxdl</td>
<td>uart5_txd</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>vout2_d0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>vin6a_d1</td>
<td>vin6a_d0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>vin4a_d0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>vin5a_fld0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>i2c4_sda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>pr2_mii1_rxer</td>
<td>pr2_mii1_rxlink</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>pr2_pru0_gpi14</td>
<td>pr2_pru0_gpi15</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>pr2_pru0_gpo14</td>
<td>pr2_pru0_gpo15</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>gpio5_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Driver off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 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   |
| 2nd 15 | gpio6_12   |
| 2nd 16 | Driver off |
| 2nd 17 | Driver off |
### P9.14-P9.16

<table>
<thead>
<tr>
<th><strong>GPIO</strong></th>
<th><strong>P9.14</strong></th>
<th><strong>P9.15</strong></th>
<th><strong>P9.16</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>121</td>
<td>76</td>
<td>122</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ehrpwm3A</td>
<td>eQEP2_index</td>
<td>ehrpwm3B</td>
</tr>
<tr>
<td>11</td>
<td>pr1_mii1_rxd2</td>
<td>pr1_mii1_rxd1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>pr1 Pru1_gpi14</td>
<td>pr1 Pru0_gpi5</td>
<td>pr1 Pru1_gpi15</td>
</tr>
<tr>
<td>13</td>
<td>pr1 Pru1_gpo14</td>
<td>pr1 Pru0_gpo5</td>
<td>pr1 Pru1_gpo15</td>
</tr>
<tr>
<td>14</td>
<td>gpio4_25</td>
<td>gpio3_12</td>
<td>gpio4_26</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P9.17-P9.19

<table>
<thead>
<tr>
<th><strong>GPIO</strong></th>
<th><strong>P9.17</strong></th>
<th><strong>P9.18</strong></th>
<th><strong>P9.19</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>gpio7_17</td>
<td>gpio7_16</td>
<td>gpio7_3</td>
</tr>
<tr>
<td>14</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2nd BALL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd REG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd MODE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
### Table 2.31 – continued from previous page

<table>
<thead>
<tr>
<th>2nd 10</th>
<th>2nd 11</th>
<th>2nd 12</th>
<th>2nd 13</th>
<th>2nd 14</th>
<th>2nd 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c5_scl</td>
<td>pr2_mii_mti0_clk</td>
<td>pr2_mii0_rxer</td>
<td>pr2_prul_gpi9</td>
<td>pr2_prul_gpo9</td>
<td>Driver off</td>
</tr>
<tr>
<td>i2c5_sda</td>
<td>pr2_mii0_sda</td>
<td>pr2_prul_gpi8</td>
<td>pr2_prul_gpo8</td>
<td>pr2_prul_gpo2</td>
<td>Driver off</td>
</tr>
<tr>
<td>eQEP2A_in</td>
<td>pr1_edio_sof</td>
<td>pr1_prul_gpi2</td>
<td>pr1_prul_gpo2</td>
<td>pr1_prul_gpo2</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P9.17-P9.19

<table>
<thead>
<tr>
<th>P9.20</th>
<th>P9.21</th>
<th>P9.22</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>196</td>
<td>67</td>
</tr>
<tr>
<td>BALL</td>
<td>T9</td>
<td>AF8</td>
</tr>
<tr>
<td>REG</td>
<td>0x1444</td>
<td>0x14F0</td>
</tr>
<tr>
<td>MODE 0</td>
<td>gpmc_a1</td>
<td>vin1a_vsync0</td>
</tr>
<tr>
<td>1</td>
<td>vin1b_de1</td>
<td>mcasp2_axr10</td>
</tr>
<tr>
<td>2</td>
<td>vin3a_d17</td>
<td>mcasp1_axr6</td>
</tr>
<tr>
<td>3</td>
<td>vout3_d17</td>
<td>mcasp3_axclkx</td>
</tr>
<tr>
<td>4</td>
<td>vout4a_d1</td>
<td>vout3_vsync</td>
</tr>
<tr>
<td>5</td>
<td>uart7_rtsn</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>vin4b_d1</td>
<td>vout2_clk</td>
</tr>
<tr>
<td>7</td>
<td>i2c4_sda</td>
<td>timer13</td>
</tr>
<tr>
<td>8</td>
<td>uart5_txd</td>
<td>spi3_cs0</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>eQEP1_strobe</td>
<td>timer15</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>gpio7_4</td>
<td>gpio3_3</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2nd BALL</td>
<td>D2</td>
<td>B22</td>
</tr>
<tr>
<td>2nd REG</td>
<td>0x1578</td>
<td>0x17C4</td>
</tr>
<tr>
<td>2nd MODE 0</td>
<td>vin2a_d4</td>
<td>spi2_d1</td>
</tr>
<tr>
<td>2nd 1</td>
<td>uart3_txd</td>
<td>uart3_rxd</td>
</tr>
<tr>
<td>2nd 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 4</td>
<td>vout2_d19</td>
<td></td>
</tr>
<tr>
<td>2nd 5</td>
<td>emu14</td>
<td></td>
</tr>
<tr>
<td>2nd 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 8</td>
<td>uart10_ctsn</td>
<td></td>
</tr>
<tr>
<td>2nd 9</td>
<td>kbd_col1</td>
<td></td>
</tr>
<tr>
<td>2nd 10</td>
<td>ehrpwm1_senso</td>
<td></td>
</tr>
<tr>
<td>2nd 11</td>
<td>pr1_edc_sync0_out</td>
<td></td>
</tr>
<tr>
<td>2nd 12</td>
<td>pr1_prul_gpi1</td>
<td></td>
</tr>
<tr>
<td>2nd 13</td>
<td>pr1_prul_gpo1</td>
<td></td>
</tr>
<tr>
<td>2nd 14</td>
<td>gpio4_5</td>
<td>gpio7_15</td>
</tr>
<tr>
<td>2nd 15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P9.20-P9.22
### P9.23-P9.25

<table>
<thead>
<tr>
<th>GPIO</th>
<th>P9.23</th>
<th>P9.24</th>
<th>P9.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALL</td>
<td>A22</td>
<td>F20</td>
<td>D18</td>
</tr>
<tr>
<td>REG</td>
<td>0x17B4</td>
<td>0x168C</td>
<td>0x1694</td>
</tr>
<tr>
<td>MODE 0</td>
<td>spi1_cs1</td>
<td>gpio6_15</td>
<td>xref_clk0</td>
</tr>
<tr>
<td>1</td>
<td>mcap1_axr9</td>
<td>mcap2_axr8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>sata1_led</td>
<td>dcan2_rx</td>
<td>mcap1_axr4</td>
</tr>
<tr>
<td>3</td>
<td>spi2_cs1</td>
<td>uart10_txd</td>
<td>mcap1_ahclkx</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>mcap5_ahclkx</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>vout2_vsync</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>vina_d0</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>vina_vsync0</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>i2c3_scl</td>
<td>clkout2</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>timer2</td>
<td>timer13</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>pr2_mii1_col</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>pr2_pru1_gpi5</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>pr2_pru1_gpo5</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>gpio7_11</td>
<td>gpio6_15</td>
<td>gpio6_17</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P9.26-P9.29

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BALL</td>
<td>E21</td>
<td>C3</td>
<td>A12</td>
<td>A11</td>
</tr>
<tr>
<td>REG</td>
<td>0x1688</td>
<td>0x15A0</td>
<td>0x16E0</td>
<td>0x16D8</td>
</tr>
<tr>
<td>MODE 0</td>
<td>gpio6_14</td>
<td>vin2a_d14</td>
<td>mcap1_axr11</td>
<td>mcap1_axr9</td>
</tr>
<tr>
<td>1</td>
<td>mcap1_axr8</td>
<td>mcap6_fsx</td>
<td>mcap6_axr1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>dcan2_tx</td>
<td>mcap6_fsr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uart10_rxd</td>
<td>rgmi1_txd3</td>
<td>spi3_cs0</td>
<td>spi3_d1</td>
</tr>
<tr>
<td>4</td>
<td>vout2_d9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>vout2_hsync</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>vina_d12</td>
<td>vina_d14</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>vina_vsync0</td>
<td>mii1_txc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>i2c3_sda</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>timer1</td>
<td>eQEP3B_in</td>
<td>timer8</td>
<td>timer6</td>
</tr>
<tr>
<td>11</td>
<td>pr1_mii_mri1_clk</td>
<td>pr2_mii0_txd1</td>
<td>pr2_mii0_txd3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>pr1_pru1_gpi11</td>
<td>pr2_pru1_gpi13</td>
<td>pr2_pru1_gpi11</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>pr1_pru1_gpo11</td>
<td>pr2_pru1_gpo13</td>
<td>pr2_pru1_gpo11</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>gpio6_14</td>
<td>gpio4_15</td>
<td>gpio4_17</td>
<td>gpio5_11</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

2nd BALL  | AE2   | J14   | D14   |
2nd REG   | 0x1544| 0x16B0| 0x16A8|
2nd MODE 0| vin1a_d20| mcap1_fsr | mcap1_fsx |
2nd 1     | vin1b_d3  | mcap7_axr3 |
2nd 2     |        |
2nd 3     |        |
2nd 4     | vout3_d3  |
2nd 5     |        |
2nd 6     | vin3a_d4  | vout2_d1 |
2nd 7     |        | vina_de0 |
2nd 8     |        | vina_d1 |
2nd 9     | kbd_col5  |

continues on next page
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd 10</td>
<td>pr1_edio_data_in4</td>
<td>i2c4_scl</td>
<td>i2c3_scl</td>
</tr>
<tr>
<td>2nd 11</td>
<td>pr1_edio_data_out4</td>
<td></td>
<td>pr2_mdio_data</td>
</tr>
<tr>
<td>2nd 12</td>
<td>pr1_prut0_gpi17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 13</td>
<td>pr1_prut0_gpo17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 14</td>
<td>gpio3_24</td>
<td>gpio5_1</td>
<td>gpio7_30</td>
</tr>
<tr>
<td>2nd 15</td>
<td>Driver off</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

### P9.26-P9.29

<table>
<thead>
<tr>
<th></th>
<th>P9.30</th>
<th>P9.31</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>140</td>
<td>138</td>
</tr>
<tr>
<td>BALL</td>
<td>B13</td>
<td>B12</td>
</tr>
<tr>
<td>REG</td>
<td>0x16DC</td>
<td>0x16D4</td>
</tr>
<tr>
<td>MODE 0</td>
<td>mcasp1_axr10</td>
<td>mcasp1_axr8</td>
</tr>
<tr>
<td>1</td>
<td>mcasp6_aclkx</td>
<td>mcasp6_axr0</td>
</tr>
<tr>
<td>2</td>
<td>mcasp6_aclkx</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>spi3_d0</td>
<td>spi3_scl</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>vin6a_d13</td>
<td>vin6a_d15</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>timer7</td>
<td>timer5</td>
</tr>
<tr>
<td>11</td>
<td>pr2_mii0_txd2</td>
<td>pr2_mii0_txen</td>
</tr>
<tr>
<td>12</td>
<td>pr2_prut1_gpi12</td>
<td>pr2_prut1_gpi10</td>
</tr>
<tr>
<td>13</td>
<td>pr2_prut1_gpo12</td>
<td>pr2_prut1_gpo10</td>
</tr>
<tr>
<td>14</td>
<td>gpio5_12</td>
<td>gpio5_10</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2nd BALL</td>
<td>C14</td>
<td></td>
</tr>
<tr>
<td>2nd REG</td>
<td>0x16A4</td>
<td></td>
</tr>
<tr>
<td>2nd MODE 0</td>
<td>mcasp1_aclkx</td>
<td></td>
</tr>
<tr>
<td>2nd 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 7</td>
<td>vin6a_fld0</td>
<td></td>
</tr>
<tr>
<td>2nd 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 10</td>
<td>i2c3_sda</td>
<td></td>
</tr>
<tr>
<td>2nd 11</td>
<td>pr2_mdio_mdclk</td>
<td></td>
</tr>
<tr>
<td>2nd 12</td>
<td>pr2_prut1_gpi7</td>
<td></td>
</tr>
<tr>
<td>2nd 13</td>
<td>pr2_prut1_gpo7</td>
<td></td>
</tr>
<tr>
<td>2nd 14</td>
<td>gpio7_31</td>
<td></td>
</tr>
<tr>
<td>2nd 15</td>
<td>Driver off</td>
<td></td>
</tr>
</tbody>
</table>

### P9.30-P9.31

### P9.32-P9.40
<table>
<thead>
<tr>
<th></th>
<th>P9.41</th>
<th>P9.42</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>180</td>
<td>114</td>
</tr>
<tr>
<td>BALL</td>
<td>C23</td>
<td>E14</td>
</tr>
<tr>
<td>REG</td>
<td>0x16A0</td>
<td>0x16E4</td>
</tr>
<tr>
<td>MODE 0</td>
<td>xref_clk</td>
<td>mcasp1_axr12</td>
</tr>
<tr>
<td>1</td>
<td>mcasp2_axr11</td>
<td>mcasp7_axr0</td>
</tr>
<tr>
<td>2</td>
<td>mcasp1_axr7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>mcasp4_ahclkx</td>
<td>spi3_cs1</td>
</tr>
<tr>
<td>4</td>
<td>mcasp8_ahclkx</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>vout2_de</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>hdq0</td>
<td>vin6a_d11</td>
</tr>
<tr>
<td>8</td>
<td>vin4a_de0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>clkout3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>timer16</td>
<td>timer9</td>
</tr>
<tr>
<td>11</td>
<td>pr2_mii0_txd0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>pr2_pru1_gpi14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>pr2_pru1_gpo14</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>gpio6_20</td>
<td>gpio4_18</td>
</tr>
<tr>
<td>15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
<tr>
<td>2nd BALL</td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>2nd REG</td>
<td>0x1580</td>
<td>0x159C</td>
</tr>
<tr>
<td>2nd MODE 0</td>
<td>vin2a_d6</td>
<td>vin2a_d13</td>
</tr>
<tr>
<td>2nd 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 3</td>
<td>rgmii1_txctl</td>
<td></td>
</tr>
<tr>
<td>2nd 4</td>
<td>vout2_d17</td>
<td>vout2_d10</td>
</tr>
<tr>
<td>2nd 5</td>
<td>emu16</td>
<td></td>
</tr>
<tr>
<td>2nd 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd 8</td>
<td>mii1_rxd1</td>
<td>mii1_rxdv</td>
</tr>
<tr>
<td>2nd 9</td>
<td>kbd_col3</td>
<td>kbd_row8</td>
</tr>
<tr>
<td>2nd 10</td>
<td>eQEP2B_in</td>
<td>eQEP3A_in</td>
</tr>
<tr>
<td>2nd 11</td>
<td>pr1_mii_mt1_clk</td>
<td>pr1_mii1_txd0</td>
</tr>
<tr>
<td>2nd 12</td>
<td>pr1_pru1_gpi3</td>
<td>pr1_pru1_gpi10</td>
</tr>
<tr>
<td>2nd 13</td>
<td>pr1_pru1_gpo3</td>
<td>pr1_pru1_gpo10</td>
</tr>
<tr>
<td>2nd 14</td>
<td>gpio4_7</td>
<td>gpio4_14</td>
</tr>
<tr>
<td>2nd 15</td>
<td>Driver off</td>
<td>Driver off</td>
</tr>
</tbody>
</table>

**P9.41-P9.42**

**P9.43-P9.46**  TODO

Serial Debug

TODO

USB 3 Type-C

TODO
USB 2 Type-A
TODO

Gigabit Ethernet
TODO

Coaxial
TODO

microSD Memory
TODO

microHDMI
TODO

2.4.8 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

BeagleBone® Black Cape Compatibility
TODO
See https://elinux.org/Beagleboard:BeagleBone_cape_interface_spec for now.

EEPROM
TODO

Pin Usage Consideration
TODO

GPIO
TODO

I2C
TODO
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>
<thead>
<tr>
<th>Function</th>
<th>Pin</th>
<th>ABC Ball</th>
<th>Pinctrl Register</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart3_txd</td>
<td>P9.21</td>
<td>B22</td>
<td>0x17C4</td>
<td>1</td>
</tr>
<tr>
<td>uart3_rxd</td>
<td>P9.22</td>
<td>A26</td>
<td>0x17C0</td>
<td>1</td>
</tr>
<tr>
<td>uart5_txd</td>
<td>P9.13</td>
<td>C17</td>
<td>0x1730</td>
<td>4</td>
</tr>
<tr>
<td>uart5_rxd</td>
<td>P9.11</td>
<td>B19</td>
<td>0x172C</td>
<td>4</td>
</tr>
<tr>
<td>uart5_ctsn</td>
<td>P8.05</td>
<td>AC9</td>
<td>0x178C</td>
<td>2</td>
</tr>
<tr>
<td>uart5_rtsn</td>
<td>P8.06</td>
<td>AC3</td>
<td>0x1790</td>
<td>2</td>
</tr>
<tr>
<td>uart8_txd</td>
<td>P8.37</td>
<td>A21</td>
<td>0x1738</td>
<td>3</td>
</tr>
<tr>
<td>uart8_rxd</td>
<td>P8.38</td>
<td>C18</td>
<td>0x1734</td>
<td>3</td>
</tr>
<tr>
<td>uart8_ctsn</td>
<td>P8.31</td>
<td>G16</td>
<td>0x173C</td>
<td>3</td>
</tr>
<tr>
<td>uart8_rtsn</td>
<td>P8.32</td>
<td>D17</td>
<td>0x1740</td>
<td>3</td>
</tr>
<tr>
<td>uart10_txd</td>
<td>P9.24</td>
<td>F20</td>
<td>0x168C</td>
<td>3</td>
</tr>
<tr>
<td>uart10_rxd</td>
<td>P9.26</td>
<td>E21</td>
<td>0x1688</td>
<td>3</td>
</tr>
<tr>
<td>uart10_ctsn</td>
<td>P8.03</td>
<td>AB8</td>
<td>0x179C</td>
<td>2</td>
</tr>
<tr>
<td>uart10_rtsn</td>
<td>P8.04</td>
<td>AB5</td>
<td>0x17A0</td>
<td>2</td>
</tr>
<tr>
<td>uart10_txd</td>
<td>P9.24</td>
<td>F20</td>
<td>0x168C</td>
<td>3</td>
</tr>
<tr>
<td>uart10_rxd</td>
<td>P9.26</td>
<td>E21</td>
<td>0x1688</td>
<td>3</td>
</tr>
<tr>
<td>uart10_ctsn</td>
<td>P9.20</td>
<td>D2</td>
<td>0x1578</td>
<td>8</td>
</tr>
<tr>
<td>uart10_rtsn</td>
<td>P9.19</td>
<td>F4</td>
<td>0x157C</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Pin</th>
<th>ABC Ball</th>
<th>Pinctrl Register</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr2_uart0_txd</td>
<td>P8.31</td>
<td>C8</td>
<td>0x1614</td>
<td>10</td>
</tr>
<tr>
<td>pr2_uart0_rxd</td>
<td>P8.33</td>
<td>C6</td>
<td>0x1610</td>
<td>10</td>
</tr>
<tr>
<td>pr2_uart0_cts_n</td>
<td>P8.34</td>
<td>D8</td>
<td>0x1608</td>
<td>10</td>
</tr>
<tr>
<td>pr2_uart0_rts_n</td>
<td>P8.35</td>
<td>A5</td>
<td>0x160C</td>
<td>10</td>
</tr>
<tr>
<td>pr1_uart0_txd</td>
<td>P8.43</td>
<td>F10</td>
<td>0x15E4</td>
<td>10</td>
</tr>
<tr>
<td>pr1_uart0_rxd</td>
<td>P8.44</td>
<td>G11</td>
<td>0x15E8</td>
<td>10</td>
</tr>
<tr>
<td>pr1_uart0_cts_n</td>
<td>P8.45</td>
<td>F11</td>
<td>0x15DC</td>
<td>10</td>
</tr>
<tr>
<td>pr1_uart0_rts_n</td>
<td>P8.46</td>
<td>G10</td>
<td>0x15E0</td>
<td>10</td>
</tr>
</tbody>
</table>

TODO

SPI

TODO

Analog

TODO

PWM, TIMER, eCAP or PRU PWM/eCAP

TODO

eQEP

TODO

2.4. BeagleBone AI
CAN
TODO

McASP (audio serial like I2S and AC97)
TODO

MMC
TODO

LCD
TODO

PRU GPIO
TODO

CLKOUT
TODO

Expansion Connector Headers

TODO: discuss header options for working with the expansion connectors per https://github.com/beagleboard/beaglebone-black/wiki/System-Reference-Manual#84-expansion-connectors

Signal Usage
TODO

Cape Power
TODO

Mechanical
TODO

2.4.9 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://github.com/beagleboard/beaglebone-ai/tree/master/Mechanical
2.4.10 Pictures

BeagleBone AI Back of Board Image
2.4.11 Support Information

TODO: Reference https://beagleboard.org/support and https://beagleboard.org/resources
Related TI documentation: http://www.ti.com/tool/BEAGLE-3P-BBONE-AI

2.4.12 Terms and Conditions

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
BeagleBone AI 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. Cet appareil numérique de la classe A ou B est conforme à la norme NMB-003 du Canada. Les changements ou les modifications pas expressément approuvés par la partie responsable de la conformité ont pu vider l’autorité de l’utilisateur pour actionner l’équipement.

WARRANTY AND DISCLAIMERS

The design materials referred to in this document are *NOT SUPPORTED* and DO NOT constitute a reference design. Support of the open source developer community is provided through the the resources defined at https://beagleboard.org/support.

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 PERFORMANCE OF THE DESIGN MATERIALS IS WITH YOU. SHOULD THE DESIGN MATERIALS PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR 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.

Additional terms BeagleBoard.org Foundation and logo-licensed manufacturers (together, henceforth identified as “Supplier”) provide BeagleBone AI under the following conditions:

The user assumes all responsibility and liability for proper and safe handling of the goods. Further, the user indemnifies Supplier from all claims arising from the handling or use of the goods.

Should BeagleBone AI not meet the specifications indicated in the System Reference Manual, BeagleBone AI 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 assume no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein.

Warnings and Restrictions

For Feasibility Evaluation Only, in Laboratory/Development Environments BeagleBone AI 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:

1. 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 BeagleBone AI for evaluation, testing and other purposes.

2. 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 BeagleBone AI. Further, you are responsible to assure that any interfaces (electronic and/or mechanical) between BeagleBone AI 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.

3. Since BeagleBone AI 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 BeagleBone AI will not result in any property damage, injury or death, even if BeagleBone AI should fail to perform as described or expected.

**Certain Instructions**  It is important to operate BeagleBone AI within Supplier’s recommended specifications and environmental considerations per the user guidelines. Exceeding the specified BeagleBone AI 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 prior to 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 BeagleBone AI and/or interface electronics. Please consult the System Reference Manual prior to connecting any load to BeagleBone AI 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 BeagleBone AI’s schematic located at the link in BeagleBone AI’s 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 BeagleBone AI.

**Agreement to Defend, Indemnify and Hold Harmless**  You agree to defend, indemnify and hold Supplier, its 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 BeagleBone AI that is not in accordance with the terms of the agreement. This obligation shall apply whether Claims arise under law of tort or contract or any other legal theory, and even if BeagleBone AI 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 Supplier of such intent and enter into a separate Assurance and Indemnity Agreement.

### 2.5 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.

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 boards-terms-and-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.

2.5.1 Introduction

This document is the System Reference Manual for BeagleBone AI-64 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 AI-64 or BeagleBone AI-64 as a reminder.

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 frequently check the BeagleBone AI-64 git repository for the most up to date support documents.

2.5.2 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.

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>
<thead>
<tr>
<th>Rev</th>
<th>Changes</th>
<th>Date</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.1</td>
<td>AI-64 initial prototype</td>
<td>September 2021</td>
<td>James Anderson</td>
</tr>
<tr>
<td>0.0.2</td>
<td>AI-64 final prototype</td>
<td>December 2021</td>
<td>James Anderson</td>
</tr>
<tr>
<td>0.0.3</td>
<td>AI-64 initial production release</td>
<td>June 9, 2022</td>
<td>Deepak Khatri and Jason Kridner</td>
</tr>
</tbody>
</table>

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.
2.5.3 Connecting up your BeagleBone AI-64

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 adaptor for SBCs.
- 5V > 3A laptop/mobile adaptor with USB-C cable.

All the Fig 3.1 BeagleBone AI-64 connections ports we will use in this chapter are shown in the figure below.

![Fig. 3.1 BeagleBone AI-64 connections ports](image)

Methods of operation

1. Tethered to a PC, or
2. As a standalone development platform in a desktop PC configuration with a Display Port Monitor, power supply, keyboard, and mouse

What’s In the Box

In the box you will find three main items as shown in Fig: BeagleBone AI-64 box content.

- BeagleBone AI-64.
- Instruction card.

Get yourself a USB-A to USB-C and you have a setup for the tethered scenario and creates an out of box experience where the board can be used immediately with no other equipment needed.

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 and
Fig. 2.82: Fig: BeagleBone AI-64 box content

- a 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 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. *Fig: Tethered Configuration* shows this 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. You should always use an external 5V > 3A DC power supply connected to the barrel jack.

**Connect the Cable to the Board**

1. Connect the type C USB cable to the board as shown in *Fig: USB Connection to the Board*. The connector is on the top side of the board near barrel jack.
2. Connect the USB-A end of the cable to your PC or laptop USB port as shown in the usb-a-connect-figure below.

3. The board will power on and the power LED will be on as shown in Fig: Board Power LED 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 boot-status-figure 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 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-config-figure. 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. 2.86: Fig: Board Power LED

Fig. 2.87: Fig: Board Boot Status

Fig. 2.88: Fig: 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](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. 2.89: Connect miniDP-DP or active miniDP-HDMI cable to BeagleBone AI-64](image)

2. If you have an Display Port or HDMI monitor with HDMI-HDMI or DP-DP cable you can use adapters as shown in. *Fig: Display adaptors.*

![Fig. 2.90: Fig: Display adaptors](image)
3. If you have wired/wireless USB keyboard and mouse such as seen in FigKeyboard and Mouse below, you need to plug the receiver in the USB host port of the board as shown in FigKeyboard and Mouse.  

![FigKeyboard and Mouse](image)

**Fig. 2.91: FigKeyboard 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 Fig: Ethernet Cable Connection. Any standard 100M Ethernet cable should work.  

![Fig: Ethernet Cable Connection](image)

**Fig. 2.92: Fig: Ethernet Cable Connection**

5. The final step is to plug in the DC power supply to the DC power jack as shown in barrel-jack-figure 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 miniDP-figure 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.
Fig. 2.93: Fig: External DC Power

Fig. 2.94: Fig: Connect miniDP to DP or active miniDP to HDMI Cable to the Board

Fig. 2.95: Fig: BeagleBone AI-64 LEDs
• **USR0** is the heartbeat indicator from the Linux kernel.

• **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 *Fig: 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! *Fig: BeagleBone XFCE Desktop Screen* shows the desktop after booting.

![BeagleBone XFCE Desktop Screen](image)

**Fig. 2.96: BeagleBone XFCE Desktop Screen**

2.5.4 **BeagleBone AI-64 Overview**

BeagleBone AI-64 is the latest addition to BeagleBoard.org family and like its predecessors, 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 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.

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 AI-64 is manufactured and warranted by partners listed at [https://beagleboard.org/logo](https://beagleboard.org/logo) for the benefit of the community and its supporters including the current BeagleBoard.org Foundation board members.
• Jason Kridner, principal of JK Embedded Consulting an independent contractor and architect for new Beagle designs.
• Drew Fustini, independent Linux developer
• Robert Nelson, applications engineer at Digi-Key
• Mark Yoder, professor at Rose-Hulman Institute of Technology
• Kathy Giori, product engineer at ZEDEDA

See bbb.io/about

BeagleBone AI-64 has been designed by Seeed Studio (Seeed Development Limited) under guidance from BeagleBoard.org Foundation.

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>
<thead>
<tr>
<th>Feature</th>
<th>AI-64</th>
<th>AI</th>
<th>Black</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC</td>
<td>TDA4VM</td>
<td>AM5729</td>
<td>AM3358</td>
</tr>
<tr>
<td>Arm CPU</td>
<td>Cortex-A72 (64-bit)</td>
<td>Cortex-A15 (32-bit)</td>
<td>Cortex-A8 (32-bit)</td>
</tr>
<tr>
<td>Arm cores/MHz</td>
<td>2x 2GHz</td>
<td>2x 1.5GHz</td>
<td>1x 1GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>4GB</td>
<td>1GB</td>
<td>512MB</td>
</tr>
<tr>
<td>eMMC flash</td>
<td>16GB</td>
<td>16GB</td>
<td>4GB</td>
</tr>
<tr>
<td>Size</td>
<td>4” x 3.1”</td>
<td>3.4” x 2.1”</td>
<td>.4” x 2.1”</td>
</tr>
<tr>
<td>Display</td>
<td>miniDP + DSI</td>
<td>microHDMI</td>
<td>microHDMI</td>
</tr>
<tr>
<td>USB host (Type-A)</td>
<td>2x 5Gbps</td>
<td>1x 480Mbps</td>
<td>1x 480Mbps</td>
</tr>
<tr>
<td>USB dual-role</td>
<td>Type-C 5Gbps</td>
<td>Type-C 5Gbps</td>
<td>mini-AB 480Mbps</td>
</tr>
<tr>
<td>Ethernet</td>
<td>10/100/1000M</td>
<td>10/100/1000M</td>
<td>10/100M</td>
</tr>
<tr>
<td>M.2</td>
<td>E-key</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WiFi/ Bluetooth</td>
<td>-</td>
<td>AzureWave AW‑CM256SM</td>
<td>-</td>
</tr>
</tbody>
</table>

Note: TODO: add cape compatibility details

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 2.38: Table: BeagleBone AI-64 Features and Specification

| Feature                      | Processor               | Graphics Engine         | SDRAM Memory                           | Onboard Flash          | PMIC                        | Debug Support                      | Power Source | PCB               | Indicators                      | USB-3.0 Client Port              | USB-3.0 Host Port                  | Ethernet                        | SD/MMC Connector | User Input   | Video Out | Audio | Weight | Power |
|------------------------------|-------------------------|--------------------------|----------------------------------------|------------------------|-----------------------------|-------------------------------|----------------|-------------------|-----------------------------|-------------------------------|---------------------------------|---------------------|----------------|---------|--------|--------|-------|
| Processor                   | Texas Instruments TDA4VM| PowerVR® Series8XE GX8430| LPDDR4 3.2GHz (4GB) Kingston Q3222PM1WDGTK-U | eMMC (16GB) Kingston EMMC16G-TB29-PZ90 | TPS65941213 and TPS65941111 PMICs regulator and one additional LDO | 2×3 pin 3.3V TTL header       | USB C or DC Jack (5V, >3A) | 4" x 3.1" | 1-Power, 5-User Controllable LEDs | Access to USB0, SuperSpeed, dual-role mode via USB-C (no power output) | TUSB8041 4-port SuperSpeed hub on USB1, 2xType A Socket, up-to 2.8A total, depending on power input | Gigabit, RJ45, link indicator, speed indicator | microSD, 1.8/3.3V | 1. Reset Button | miniDP | via miniDP (stereo) | 192gm (with heatsink) | Refer to main-board-power section |

**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.

**Board components**

*Fig: 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.
• **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.

### 2.5.5 BeagleBone AI-64 High Level Specification

*Fig: 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. 2.98: Fig: 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™
  - 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 Fig: 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
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.

**Note**: 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.

**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 1.5A of current at 5V. Under USB power, the board will not be able to supply the full 1.5A, but should be sufficient to supply enough current for a lower power USB device supplying power between 50 to 100mA.
Power Sources

The board can be powered from two different sources:

- A 5V > 3A power supply plugged into the barrel jack.
- A wall adaptor with 5V > 3A output power.

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.

2.5.6 Connectors

Expansion Connectors

The 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.

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:

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.03</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 20</td>
</tr>
</tbody>
</table>

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.

**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.**

<table>
<thead>
<tr>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB1 DP</td>
<td>USB1 DN</td>
<td>VSYS_5V0</td>
<td>GND</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P8.01-P8.02</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.01 P8.02</td>
</tr>
<tr>
<td>GND GND</td>
</tr>
</tbody>
</table>

<p>| P8.03-P8.05 |</p>
<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.03</th>
<th>P8.04</th>
<th>P8.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 20</td>
<td>1 48</td>
<td>1 33</td>
</tr>
<tr>
<td>BALL</td>
<td>AH21</td>
<td>AC29</td>
<td>AH25</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C054</td>
<td>0x00011C0C4</td>
<td>0x00011C088</td>
</tr>
<tr>
<td>Page</td>
<td>46</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>MODE 0</td>
<td>PRG1_PRU0_GPO19</td>
<td>PRG0_PRU0_GPO5</td>
<td>PRG1_PRU1_GPO12</td>
</tr>
<tr>
<td>1</td>
<td>PRG1_PRU0_GPI19</td>
<td>PRG0_PRU0_GPI5</td>
<td>PRG1_PRU1_GPI12</td>
</tr>
<tr>
<td>2</td>
<td>PRG1_IEP0_EDC_SYNC_OUT0</td>
<td>~</td>
<td>PRG1_RGMII2_TD1</td>
</tr>
<tr>
<td>3</td>
<td>PRG1_PWM0_TZ_OUT</td>
<td>PRG0_PWM3_B2</td>
<td>PRG1_PWM1_A0</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>~</td>
<td>RGMII2_TD1</td>
</tr>
<tr>
<td>5</td>
<td>RMI5_TXD0</td>
<td>RMI3_TXD0</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>MCAN6_TX</td>
<td>~</td>
<td>MCAN7_TX</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_20</td>
<td>GPIO0_48</td>
<td>GPIO0_33</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>GPMC0_AD0</td>
<td>RGMII8_TD1</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>VOUT0_EXPCLCLKIN</td>
<td>~</td>
<td>VOUT0_DATA12</td>
</tr>
<tr>
<td>11</td>
<td>VPFE0_PCLK</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP4_AFSX</td>
<td>MCASP0_AXR3</td>
<td>MCASP9_AFSX</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.06</th>
<th>P8.07</th>
<th>P8.08</th>
<th>P8.09</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 34</td>
<td>1 15</td>
<td>1 14</td>
<td>1 17</td>
</tr>
<tr>
<td>BALL</td>
<td>AG25</td>
<td>AD24</td>
<td>AG24</td>
<td>AE24</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C08C</td>
<td>0x00011C03C</td>
<td>0x00011C038</td>
<td>0x00011C044</td>
</tr>
<tr>
<td>Page</td>
<td>51</td>
<td>44</td>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>MODE 0</td>
<td>PRG1_PRU1_GPO13</td>
<td>PRG1_PRU0_GPO14</td>
<td>PRG1_PRU0_GPO13</td>
<td>PRG1_PRU0_GPO16</td>
</tr>
<tr>
<td>1</td>
<td>PRG1_PRU1_GPI13</td>
<td>PRG1_PRU0_GPI14</td>
<td>PRG1_PRU0_GPI13</td>
<td>PRG1_PRU0_GPI16</td>
</tr>
<tr>
<td>2</td>
<td>PRG1_RGMII2_TD2</td>
<td>PRG1_RGMII1_TD3</td>
<td>PRG1_RGMII1_TD2</td>
<td>PRG1_RGMII1_TXC</td>
</tr>
<tr>
<td>3</td>
<td>PRG1_PWM1_B0</td>
<td>PRG1_PWM0_A1</td>
<td>PRG1_PWM0_B0</td>
<td>PRG1_PWM0_A2</td>
</tr>
<tr>
<td>4</td>
<td>RGMII2_TD2</td>
<td>RGMII1_TD3</td>
<td>RGMII1_TD2</td>
<td>RGMII1_TXC</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>MCAN7_RX</td>
<td>MCAN5_RX</td>
<td>MCAN5_TX</td>
<td>MCAN6_RX</td>
</tr>
<tr>
<td>7</td>
<td>GPI00_34</td>
<td>GPI00_15</td>
<td>GPI00_14</td>
<td>GPI00_17</td>
</tr>
<tr>
<td>8</td>
<td>RGMII8_TD2</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>RGMII7_TD3</td>
<td>RGMII7_TD2</td>
<td>RGMII7_TXC</td>
</tr>
<tr>
<td>10</td>
<td>VOUT0_DATA13</td>
<td>VOUT0_DATA19</td>
<td>VOUT0_DATA18</td>
<td>VOUT0_DATA21</td>
</tr>
<tr>
<td>11</td>
<td>VPFE0_DATA8</td>
<td>VPFE0_DATA3</td>
<td>VPFE0_DATA2</td>
<td>VPFE0_DATA5</td>
</tr>
<tr>
<td>12</td>
<td>MCASP9_AXR0</td>
<td>MCASP7_AXR1</td>
<td>MCASP7_AXR0</td>
<td>MCASP7_AXR3</td>
</tr>
<tr>
<td>13</td>
<td>MCASP4_ACLKR</td>
<td>~</td>
<td>~</td>
<td>MCASP7_AFSR</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

P8.06-P8.09

P8.10-P8.13

2.5. BeagleBone AI-64 189
### Pin P8.10
<table>
<thead>
<tr>
<th>GPIO</th>
<th>BALL</th>
<th>REG</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 16</td>
<td>AC24</td>
<td>0x00011C040</td>
<td>44</td>
</tr>
<tr>
<td>1 60</td>
<td>AB24</td>
<td>0x00011C0F4</td>
<td>116</td>
</tr>
<tr>
<td>1 59</td>
<td>AH28</td>
<td>0x00011C0F0</td>
<td>160</td>
</tr>
<tr>
<td>1 89</td>
<td>V27</td>
<td>0x00011C168</td>
<td>159</td>
</tr>
</tbody>
</table>

### Pin P8.11
<table>
<thead>
<tr>
<th>GPIO</th>
<th>BALL</th>
<th>REG</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 10</td>
<td>AC24</td>
<td>0x00011C040</td>
<td>44</td>
</tr>
<tr>
<td>2 11</td>
<td>AB24</td>
<td>0x00011C0F4</td>
<td>116</td>
</tr>
<tr>
<td>2 12</td>
<td>AH28</td>
<td>0x00011C0F0</td>
<td>160</td>
</tr>
<tr>
<td>2 13</td>
<td>V27</td>
<td>0x00011C168</td>
<td>159</td>
</tr>
</tbody>
</table>

### Pin P8.12
<table>
<thead>
<tr>
<th>GPIO</th>
<th>BALL</th>
<th>REG</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 10</td>
<td>AC24</td>
<td>0x00011C040</td>
<td>44</td>
</tr>
<tr>
<td>3 11</td>
<td>AB24</td>
<td>0x00011C0F4</td>
<td>116</td>
</tr>
<tr>
<td>3 12</td>
<td>AH28</td>
<td>0x00011C0F0</td>
<td>160</td>
</tr>
<tr>
<td>3 13</td>
<td>V27</td>
<td>0x00011C168</td>
<td>159</td>
</tr>
</tbody>
</table>

### Pin P8.13
<table>
<thead>
<tr>
<th>GPIO</th>
<th>BALL</th>
<th>REG</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 10</td>
<td>AC24</td>
<td>0x00011C040</td>
<td>44</td>
</tr>
<tr>
<td>4 11</td>
<td>AB24</td>
<td>0x00011C0F4</td>
<td>116</td>
</tr>
<tr>
<td>4 12</td>
<td>AH28</td>
<td>0x00011C0F0</td>
<td>160</td>
</tr>
<tr>
<td>4 13</td>
<td>V27</td>
<td>0x00011C168</td>
<td>159</td>
</tr>
</tbody>
</table>

### Pin P8.14-P8.16
<table>
<thead>
<tr>
<th>GPIO</th>
<th>BALL</th>
<th>REG</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 10</td>
<td>AC24</td>
<td>0x00011C040</td>
<td>44</td>
</tr>
<tr>
<td>5 11</td>
<td>AB24</td>
<td>0x00011C0F4</td>
<td>116</td>
</tr>
<tr>
<td>5 12</td>
<td>AH28</td>
<td>0x00011C0F0</td>
<td>160</td>
</tr>
<tr>
<td>5 13</td>
<td>V27</td>
<td>0x00011C168</td>
<td>159</td>
</tr>
</tbody>
</table>

### Mode 0
<table>
<thead>
<tr>
<th>MODE</th>
<th>PRG1_PRU0_GPO15</th>
<th>PRG0_PRU0_GPO17</th>
<th>PRG0_PRU0_GPO16</th>
<th>RGMII5_TD1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PRG1_PRU0_GPI15</td>
<td>PRG0_PRU0_GPI17</td>
<td>PRG0_PRU0_GPI16</td>
<td>RMI17_TXD1</td>
</tr>
<tr>
<td>2</td>
<td>PRG1_RGMII1_TX_CTL</td>
<td>PRG0_IEP0_EDC_SYNC_OUT</td>
<td>PRG0_RGMII1_TXC</td>
<td>12C3_SCL</td>
</tr>
<tr>
<td>3</td>
<td>PRG1_PWM0_B1</td>
<td>PRG0_PWM0_B2</td>
<td>PRG0_PWM0_A2</td>
<td>~</td>
</tr>
<tr>
<td>4</td>
<td>RGMII1_TX_CTL</td>
<td>PRG0_ECAP0_SYNC_OUT</td>
<td>RGMII3_TXC</td>
<td>VOUT1_DATA4</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>TRC_DATA2</td>
</tr>
<tr>
<td>6</td>
<td>MCAN6_TX</td>
<td>~</td>
<td>~</td>
<td>EHRPWM0_B</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_16</td>
<td>GPIO0_60</td>
<td>GPIO0_59</td>
<td>GPIO0_89</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>GPMC0_AD5</td>
<td>~</td>
<td>GPMC0_A5</td>
</tr>
<tr>
<td>9</td>
<td>RGMII7_TX_CTL</td>
<td>OBSCLK1</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>VOUT0_DATA20</td>
<td>~</td>
<td>DSS_SYNC1</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>VPFE0_DATA4</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP7_AXR2</td>
<td>MCASP0_AXR13</td>
<td>MCASP0_AXR12</td>
<td>MCASP11_ACLKX</td>
</tr>
<tr>
<td>13</td>
<td>MCASP7_ACLKR</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

Bootstrap
| ~ | BOOTMODE7 | ~ | ~ |

### Mode 1
<table>
<thead>
<tr>
<th>MODE</th>
<th>PRG0_PRU1_GPO12</th>
<th>PRG0_PRU0_GPO18</th>
<th>PRG0_PRU0_GPO19</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PRG0_PRU1_GPI12</td>
<td>PRG0_PRU0_GPI18</td>
<td>PRG0_PRU0_GPI19</td>
</tr>
<tr>
<td>2</td>
<td>PRG0_RGMII2_TD1</td>
<td>PRG0_IEP0_EDC_LATCH_IN0</td>
<td>PRG0_IEP0_EDC_SYNC_OUT0</td>
</tr>
<tr>
<td>3</td>
<td>PRG0_PWM1_A0</td>
<td>PRG0_PWM0_TZ_IN</td>
<td>PRG0_PWM0_TZ_OUT</td>
</tr>
<tr>
<td>4</td>
<td>RGMII4_TD1</td>
<td>PRG0_ECAP0_IN_APWM_OUT</td>
<td>~</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_75</td>
<td>GPIO0_61</td>
<td>GPIO0_62</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>GPMC0_AD6</td>
<td>GPMC0_AD7</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP1_AXR8</td>
<td>MCASP0_AXR14</td>
<td>MCASP0_AXR15</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>UART8_CTSn</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

Bootstrap
| ~ | ~ | ~ |
### P8.17-P8.19

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.17</th>
<th>P8.18</th>
<th>P8.19</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>BALL</td>
<td>AF22</td>
<td>AJ23</td>
<td>V29</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C00C</td>
<td>0x00011C010</td>
<td>0x00011C164</td>
</tr>
<tr>
<td>Page</td>
<td>40</td>
<td>40</td>
<td>57</td>
</tr>
<tr>
<td>MODE 0</td>
<td>PRG1_PRU0_GPO2</td>
<td>PRG1_PRU0_GPO3</td>
<td>RGMII5_TD2</td>
</tr>
<tr>
<td></td>
<td>PRG1_PRU0_GPI2</td>
<td>PRG1_PRU0_GPI3</td>
<td>UART3_TXD</td>
</tr>
<tr>
<td></td>
<td>PRG1_RGMII1_RD2</td>
<td>PRG1_RGMII1_RD3</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>PRG1_PWM2_A0</td>
<td>PRG1_PWM3_A2</td>
<td>SYNC3_OUT</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C164</td>
<td>0x00011C134</td>
<td>0x00011C07C</td>
</tr>
<tr>
<td>Page</td>
<td>41</td>
<td>37</td>
<td>49</td>
</tr>
<tr>
<td>MODE 1</td>
<td>PRG0_PRU1_GPO13</td>
<td>PRG1_PRU1_GPO9</td>
<td>PRG1_PRU0_GPO4</td>
</tr>
<tr>
<td></td>
<td>PRG0_PRU1_GPI13</td>
<td>PRG1_PRU1_GPI9</td>
<td>PRG1_PRU0_GPI4</td>
</tr>
<tr>
<td></td>
<td>PRG0_RGMII2_TD2</td>
<td>PRG1_UART0_RXD</td>
<td>PRG1_RGMII1_RX_CTL</td>
</tr>
<tr>
<td>REG</td>
<td>RGMII7_RD2</td>
<td>RGMII7_RD3</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>57</td>
<td>41</td>
<td>~</td>
</tr>
<tr>
<td>MODE 2</td>
<td>PRG0_PWM1_B0</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>PRG1_PWM2_B0</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>SPI6_CS3</td>
<td>RGMII1_RX_CTL</td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 3</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>MCAN8_TX</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 4</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 5</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 6</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 7</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 8</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 9</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 10</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 12</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 13</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 14</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

### P8.20-P8.22

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.20</th>
<th>P8.21</th>
<th>P8.22</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>BALL</td>
<td>AF26</td>
<td>AF21</td>
<td>AH23</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C134</td>
<td>0x00011C07C</td>
<td>0x00011C014</td>
</tr>
<tr>
<td>Page</td>
<td>49</td>
<td>49</td>
<td>41</td>
</tr>
<tr>
<td>MODE 0</td>
<td>PRG0_PRU1_GPO13</td>
<td>PRG1_PRU1_GPO9</td>
<td>PRG1_PRU0_GPO4</td>
</tr>
<tr>
<td></td>
<td>PRG0_PRU1_GPI13</td>
<td>PRG1_PRU1_GPI9</td>
<td>PRG1_PRU0_GPI4</td>
</tr>
<tr>
<td></td>
<td>PRG0_RGMII2_TD2</td>
<td>PRG1_UART0_RXD</td>
<td>PRG1_RGMII1_RX_CTL</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 1</td>
<td>PRG0_PWM1_B0</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>SPI6_CS3</td>
<td>RGMII1_RX_CTL</td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 2</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 3</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 4</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 5</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 6</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 7</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 8</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 9</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 10</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 12</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 13</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>MODE 14</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>REG</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

### P8.23-P8.26
### P8.23-P8.25

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.23</th>
<th>P8.24</th>
<th>P8.25</th>
<th>P8.26</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 31</td>
<td>1 6</td>
<td>1 35</td>
<td>1 51</td>
</tr>
<tr>
<td>BALL</td>
<td>AB23</td>
<td>AD20</td>
<td>AH26</td>
<td>AC27</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C080</td>
<td>0x00011C018</td>
<td>0x00011C090</td>
<td>0x00011C0D0</td>
</tr>
<tr>
<td>Page</td>
<td>50</td>
<td>41</td>
<td>51</td>
<td>31</td>
</tr>
<tr>
<td>MODE</td>
<td>0</td>
<td>PRG1_PRU1_GPO10</td>
<td>PRG1_PRU0_GPO5</td>
<td>PRG1_PRU1_GPO14</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>PRG1_PRU1_GPI10</td>
<td>PRG1_PRU0_GPI5</td>
<td>PRG1_PRU1_GPI14</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>PRG1_UART0_TXD</td>
<td>~</td>
<td>PRG1_RGMII2_TD3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>PRG1_PWM2_TZ_IN</td>
<td>PRG1_PWM3_B2</td>
<td>PRG1_PWM1_A1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>~</td>
<td>~</td>
<td>RGMII2_TD3</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>RMII6_CRS_DV</td>
<td>RMII1_TX_EN</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>MCAN8_RX</td>
<td>~</td>
<td>MCAN8_TX</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>GPIO0_31</td>
<td>GPIO0_6</td>
<td>GPIO0_35</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>GPMC0_CLKOUT</td>
<td>GPMC0_WEn</td>
<td>RGMII8_TD3</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>PRG1_IEP0_EDIO_DATA_IN_OUT1</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>VOUT0_DATA10</td>
<td>~</td>
<td>VOUT0_DATA14</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>GPMC0_FCLK_MUX</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>MCASP5_ACLKX</td>
<td>MCASP3_AXR0</td>
<td>MCASP9_AXR1</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>~</td>
<td>~</td>
<td>MCASP4_AFSR</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Boot-</td>
<td>strap</td>
<td>~</td>
<td>BOOSTMODE0</td>
<td>~</td>
</tr>
</tbody>
</table>

### P8.27-P8.29

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.27</th>
<th>P8.28</th>
<th>P8.29</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 71</td>
<td>1 72</td>
<td>1 73</td>
</tr>
<tr>
<td>BALL</td>
<td>AA28</td>
<td>Y24</td>
<td>AA25</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C120</td>
<td>0x00011C124</td>
<td>0x00011C128</td>
</tr>
<tr>
<td>Page</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>MODE</td>
<td>0</td>
<td>PRG0_PRU1_GPO8</td>
<td>PRG0_PRU1_GPO9</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>PRG0_PRU1_GPI8</td>
<td>PRG0_PRU1_GPI9</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>~</td>
<td>PRG0_UART0_TXD</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>PRG0_PWM2_TZ_OUT~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>~</td>
<td>SPI3_CS3</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>MCAN11_RX</td>
<td>PRG0_IEP0_EDIO_DATA_IN_OUT1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>GPIO0_71</td>
<td>GPIO0_72</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>GPMC0_AD10</td>
<td>GPMC0_AD11</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>~</td>
<td>DSS_FSYNC3</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>MCASP1_AFSX</td>
<td>MCASP1_AXR5</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>~</td>
<td>UART8_RXD</td>
</tr>
<tr>
<td>Boot-</td>
<td>strap</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

### P8.30-P8.32
<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.30</th>
<th>P8.31</th>
<th>~</th>
<th>P8.32</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>174</td>
<td>132</td>
<td>163</td>
<td>126</td>
<td>164</td>
</tr>
<tr>
<td>BALL</td>
<td>AG26</td>
<td>A125</td>
<td>AE29</td>
<td>AG21</td>
<td>AD28</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C12C</td>
<td>0x00011C084</td>
<td>0x00011C100</td>
<td>0x00011C06C</td>
<td>0x00011C104</td>
</tr>
<tr>
<td>Page</td>
<td>37</td>
<td>50</td>
<td>34</td>
<td>48</td>
<td>34</td>
</tr>
<tr>
<td>MODE</td>
<td>PRG0_PRU1_GPIO</td>
<td>PRG1_PRU1_GPIO</td>
<td>PRG0_PRU1_GPIO</td>
<td>PRG1_PRU1_GPIO</td>
<td>PRG0_PRU1_GPIO</td>
</tr>
<tr>
<td>0</td>
<td>PRG0_PRU1_GPIO</td>
<td>PRG1_PRU1_GPIO</td>
<td>PRG0_PRU1_GPIO</td>
<td>PRG1_PRU1_GPIO</td>
<td>PRG0_PRU1_GPIO</td>
</tr>
<tr>
<td></td>
<td>PRG0_RGMII2_TXD</td>
<td>PRG1_RGMII2_TXD</td>
<td>PRG0_RGMII2_RXD</td>
<td>PRG1_RGMII2_RXD</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>4</td>
<td>RGMII4_TXD0</td>
<td>RGMII4_TXD0</td>
<td>RGMII4_RXD0</td>
<td>~</td>
<td>RGMII4_RXD1</td>
</tr>
<tr>
<td>5</td>
<td>RMI14_TX_EN</td>
<td>RMI14_TX_EN</td>
<td>RMI14_RXD0</td>
<td>RMI14_RXD1</td>
<td>RMI14_TX_EN</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>MCAN6_RX</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_74</td>
<td>GPIO0_32</td>
<td>GPIO0_63</td>
<td>GPIO0_26</td>
<td>GPIO0_64</td>
</tr>
<tr>
<td>8</td>
<td>GPMC0_A26</td>
<td>RGMII8_TXD0</td>
<td>UART4_CTSn</td>
<td>GPMC0_WPn</td>
<td>UART4_RTSn</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>VOUT0_DATA11</td>
<td>~</td>
<td>VOUT0_DATA5</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP1_AXR7</td>
<td>MCASP9_ACLKX</td>
<td>MCASP1_AXR0</td>
<td>MCASP4_AXR0</td>
<td>MCASP1_AXR1</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Boot-strap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.33</th>
<th>~</th>
<th>P8.34</th>
<th>111</th>
<th>17</th>
<th>124</th>
<th>1116</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>125</td>
<td>111</td>
<td>17</td>
<td>124</td>
<td>1116</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BALL</td>
<td>AH24</td>
<td>AA2</td>
<td>AD22</td>
<td>AD23</td>
<td>Y3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C068</td>
<td>0x00011C10C</td>
<td>0x00011C064</td>
<td>0x00011C10C</td>
<td>0x00011C10D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>48</td>
<td>67</td>
<td>41</td>
<td>47</td>
<td>67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td>PRG1_PRU1_GPIO</td>
<td>SPI0_CS0</td>
<td>PRG0_IEP0_GPIO</td>
<td>PRG1_IEP0_GPIO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>PRG1_PRU1_GPIO</td>
<td>UART0_RTSn</td>
<td>SPI0_IEP0_GPIO</td>
<td>UART0_CTSn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRG1_RGMII2_RX</td>
<td>~</td>
<td>PRG1_RGMII1_RX</td>
<td>PRG1_RGMII2_TXD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PRG1_PWM2_B2</td>
<td>~</td>
<td>PRG1_PWM3_A1</td>
<td>~</td>
<td>UART5_RXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RGMII2_RXCtl</td>
<td>~</td>
<td>RGMII1_RXC</td>
<td>RGMII2_RXD3</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>RMI12_TXD0</td>
<td>~</td>
<td>RMI12_TXD1</td>
<td>RMI2_RXER</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>AU-</td>
<td>~</td>
<td>PRG0_IEP0_EDIO_OUTVALID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>~</td>
<td>~</td>
<td>DIO_EXT_REFClk</td>
<td>~</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_25</td>
<td>GPIO0_111</td>
<td>GPIO0_7</td>
<td>GPIO0_24</td>
<td>GPIO0_116</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>RGMII8_RXCtl</td>
<td>~</td>
<td>GPMC0_CSn3</td>
<td>RGMII8_RXD3</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ESEQ1_B</td>
<td>~</td>
<td>RGMII7_RXC</td>
<td>ESEQ1_A</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>VOUT0_DATA4</td>
<td>~</td>
<td>~</td>
<td>VOUT0_DATA3</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>VPFE0_DATA13</td>
<td>~</td>
<td>~</td>
<td>VPFE0_WEN</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MCASP7_AXR2</td>
<td>~</td>
<td>MCASP6_AXR3</td>
<td>MCASP7_AXR3</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>MCASP7_ACLKR</td>
<td>~</td>
<td>MCASP6_AFSR</td>
<td>MCASP7_AFSR</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>TIMER_IO3</td>
<td>~</td>
<td>UART2_TXD</td>
<td>TIMER_IO2</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boot-strap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.5. BeagleBone AI-64

193
<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.36</th>
<th>P8.37</th>
<th>~</th>
<th>P8.38</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 8</td>
<td>1 106</td>
<td>1 11</td>
<td>1 105</td>
<td>1 9</td>
</tr>
<tr>
<td>BALL</td>
<td>AE20</td>
<td>Y27</td>
<td>AD21</td>
<td>Y29</td>
<td>A120</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C020</td>
<td>0x00011C1A2</td>
<td>0x00011C02C</td>
<td>0x00011C1A8</td>
<td>0x00011C024</td>
</tr>
<tr>
<td>Page</td>
<td>42</td>
<td>58</td>
<td>43</td>
<td>58</td>
<td>42</td>
</tr>
<tr>
<td>MODE</td>
<td>0</td>
<td>PRG1_PRU0_GPO7</td>
<td>RGMII6_RD2</td>
<td>PRG1_PRU0_GPO10</td>
<td>RGMII6_RD3</td>
</tr>
<tr>
<td>1</td>
<td>PRG1_PRU0_GPI7</td>
<td>UART4_RTSn</td>
<td>PRG1_PRU0_GPI10</td>
<td>UART4_CTSn</td>
<td>PRG1_PRU0_GPI18</td>
</tr>
<tr>
<td>2</td>
<td>PRG1_IEP0_EDC_LATCH_IN1</td>
<td>PRG1_UART0_RTSn</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>3</td>
<td>PRG1_PWM3_B1</td>
<td>UART5_TXD</td>
<td>PRG1_PWM2_B1</td>
<td>UART5_RXD</td>
<td>PRG1_PWM2_A1</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>~</td>
<td>SPI16_CS2</td>
<td>CLKOUT</td>
<td>~</td>
</tr>
<tr>
<td>5</td>
<td>AU-DIO_EXT_REFCLK1</td>
<td>TRC_DATA</td>
<td>9RMIIS_CRS_DV</td>
<td>TRC_DATA18</td>
<td>RMIIS_RXD0</td>
</tr>
<tr>
<td>6</td>
<td>MCAN4_TX</td>
<td>EHRPWM5</td>
<td>A~</td>
<td>EHRPWM_TZA</td>
<td>MCAN4_RX</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_8</td>
<td>GPIO0_106</td>
<td>GPIO0_11</td>
<td>GPIO0_105</td>
<td>GPIO0_9</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>GPMC0_A22</td>
<td>GPMC0_BE0m_CLE</td>
<td>GPMC0_A21</td>
<td>GPMC0_OEn_REn</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>PRG1_IEP0_EDIO_DATA_IN_OUT29</td>
<td>IN_OUT29</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>OBSCLK2</td>
<td>VOUT0_DATA22</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP3_AXR1</td>
<td>MCASP11_AXR2</td>
<td>MCASP3_AFSX</td>
<td>MCASP11_AXR2</td>
<td>MCASP3_AXR2</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

P8.39-P8.41

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.39</th>
<th>P8.40</th>
<th>P8.41</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 69</td>
<td>1 70</td>
<td>1 67</td>
</tr>
<tr>
<td>BALL</td>
<td>AC26</td>
<td>AA24</td>
<td>AD29</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C118</td>
<td>0x00011C11C</td>
<td>0x00011C110</td>
</tr>
<tr>
<td>Page</td>
<td>35</td>
<td>36</td>
<td>35</td>
</tr>
<tr>
<td>MODE</td>
<td>0</td>
<td>PRG0_PRU1_GPO6</td>
<td>PRG0_PRU1_GPO7</td>
</tr>
<tr>
<td>1</td>
<td>PRG0_PRU1_GPI6</td>
<td>PRG0_PRU1_GPI7</td>
<td>PRG0_PRU1_GPI4</td>
</tr>
<tr>
<td>2</td>
<td>PRG0_RGMI12_RXC</td>
<td>PRG0_IEP1_EDC_LATCH_IN1</td>
<td>PRG0_RGMI12_RX_CTL</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>~</td>
<td>PRG0_PWM2_B2</td>
</tr>
<tr>
<td>4</td>
<td>RGMI14_RXC</td>
<td>SPI3_CS0</td>
<td>RGMI14_RX_CTL</td>
</tr>
<tr>
<td>5</td>
<td>RMI14_TXD0</td>
<td>~</td>
<td>RMI14_TXD1</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>MCAN11_TX</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_69</td>
<td>GPIO0_70</td>
<td>GPIO0_67</td>
</tr>
<tr>
<td>8</td>
<td>GPMC0_A25</td>
<td>GPMC0_AD9</td>
<td>GPMC0_A24</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP1_AXR3</td>
<td>MCASP1_AXR4</td>
<td>MCASP1_AXR2</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>UART2_TXD</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>
### P8.42-P8.44

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.42</th>
<th>P8.43</th>
<th>P8.44</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 68</td>
<td>1 65</td>
<td>1 66</td>
</tr>
<tr>
<td>BALL</td>
<td>AB27</td>
<td>AD27</td>
<td>AC25</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C114</td>
<td>0x00011C108</td>
<td>0x00011C10C</td>
</tr>
<tr>
<td>MODE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>PRG0_PRU1_GPO5</td>
<td>PRG0_PRU1_GPO2</td>
<td>PRG0_PRU1_GPO3</td>
</tr>
<tr>
<td>1</td>
<td>PRG0_PRU1_GPI5</td>
<td>PRG0_PRU1_GPI2</td>
<td>PRG0_PRU1_GPI3</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_RGMII2_RD2</td>
<td>PRG0_RGMII2_RD3</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>PRG0_PWM2_A2</td>
<td>~</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>RGMII4_RD2</td>
<td>RGMII4 RD3</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>RMI14_CRS_DV</td>
<td>RMI14_RX_ER</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_68</td>
<td>GPIO0_65</td>
<td>GPIO0_66</td>
</tr>
<tr>
<td>8</td>
<td>GPMC0_AD8</td>
<td>GPMC0_A23</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP1_ACLKX</td>
<td>MCASP1_ACLKR</td>
<td>MCASP1_AFSR</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>MCASP1_AXR10</td>
<td>MCASP1_AXR11</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>BOOTMODE6</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

### P8.45-P8.46

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.45</th>
<th>P8.46</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 79</td>
<td>1 80</td>
</tr>
<tr>
<td>BALL</td>
<td>AG29</td>
<td>Y25</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C140</td>
<td>0x00011C144</td>
</tr>
<tr>
<td>Page</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>MODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>PRG0_PRU1_GPO16</td>
<td>PRG0_PRU1_GPO17</td>
</tr>
<tr>
<td>1</td>
<td>PRG0_PRU1_GPI16</td>
<td>PRG0_PRU1_GPI17</td>
</tr>
<tr>
<td>2</td>
<td>PRG0_RGMII2_TXC</td>
<td>PRG0_IEP1_EDC_SYNC_OUT1</td>
</tr>
<tr>
<td>3</td>
<td>PRG0_PWM1_A2</td>
<td>PRG0_PWM1_B2</td>
</tr>
<tr>
<td>4</td>
<td>RGMII4_TXC</td>
<td>SPI3_CLK</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_79</td>
<td>GPIO0_80</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>GPMC0_AD13</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP2_AXR2</td>
<td>MCASP2_AXR3</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>BOOTMODE3</td>
</tr>
</tbody>
</table>

**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:~$ gpio set X Y=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpio set X Y=0

For Example:

+---------+----------+
<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.11</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 1</td>
</tr>
</tbody>
</table>
+---------+----------+

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:~$ gpio set 1 20=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpio set 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.**

<table>
<thead>
<tr>
<th>P9.01-P9.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>P9.01</td>
</tr>
<tr>
<td>GND</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P9.06-P9.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>P9.06</td>
</tr>
<tr>
<td>VIN</td>
</tr>
</tbody>
</table>
### P9.11-P9.13

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>11</td>
<td>145</td>
<td>12</td>
</tr>
<tr>
<td>BALL</td>
<td>AC23</td>
<td>AE27</td>
<td>AG22</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C004</td>
<td>0x00011C0B8</td>
<td>0x00011C008</td>
</tr>
<tr>
<td>Page</td>
<td>39</td>
<td>29</td>
<td>40</td>
</tr>
<tr>
<td>MODE 0</td>
<td>PRG1_PRU0_GPO0</td>
<td>PRG0_PRU0_GPO2</td>
<td>PRG1_PRU0_GPO1</td>
</tr>
<tr>
<td>1</td>
<td>PRG1_PRU0_GPI0</td>
<td>PRG0_PRU0_GPI2</td>
<td>PRG1_PRU0_GPI1</td>
</tr>
<tr>
<td>2</td>
<td>PRG1_RGMII1_RD0</td>
<td>PRG0_RGMII1_RD2</td>
<td>PRG1_RGMII1_RD1</td>
</tr>
<tr>
<td>3</td>
<td>PRG1_PWM3_A0</td>
<td>PRG0_PWM2_A0</td>
<td>PRG1_PWM3_B0</td>
</tr>
<tr>
<td>4</td>
<td>RGMII1_RD0</td>
<td>RGMII3_RD2</td>
<td>RGMII1_RD1</td>
</tr>
<tr>
<td>5</td>
<td>RMI11_RXD0</td>
<td>RMI13_CRS_DV</td>
<td>RMI11_RXD1</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0 1</td>
<td>GPIO0_45</td>
<td>GPIO0 2</td>
</tr>
<tr>
<td>8</td>
<td>GPMC0_BEIn</td>
<td>UART3_RXD</td>
<td>GPMC0_WAIT0</td>
</tr>
<tr>
<td>9</td>
<td>RGMII7_RD0</td>
<td>~</td>
<td>RGMII7_RD1</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP6_ACLKX</td>
<td>MCASP0_ACLKR</td>
<td>MCASP6_AFSX</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>UART0_RXD</td>
<td>~</td>
<td>UART0_TXD</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

### P9.14-P9.16

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.14</th>
<th>P9.15</th>
<th>P9.16</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>193</td>
<td>147</td>
<td>194</td>
</tr>
<tr>
<td>BALL</td>
<td>U27</td>
<td>AD25</td>
<td>U24</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C178</td>
<td>0x00011C0C0</td>
<td>0x00011C17C</td>
</tr>
<tr>
<td>Page</td>
<td>56</td>
<td>30</td>
<td>56</td>
</tr>
<tr>
<td>MODE 0</td>
<td>RGMII5_RD3</td>
<td>PRG0_PRU0_GPO4</td>
<td>RGMII5_RD2</td>
</tr>
<tr>
<td>1</td>
<td>UART3_CTSn</td>
<td>PRG0_PRU0_GPI4</td>
<td>UART3_RTSn</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_RGMII1_RX_CTL</td>
<td>~</td>
</tr>
<tr>
<td>3</td>
<td>UART6_RXD</td>
<td>PRG0_PWM2_B0</td>
<td>UART6_TXD</td>
</tr>
<tr>
<td>4</td>
<td>VOUT1_DATA8</td>
<td>RGMII3_RX_CTL</td>
<td>VOUT1_DATA9</td>
</tr>
<tr>
<td>5</td>
<td>TRC_DATA6</td>
<td>RMI13_TXD1</td>
<td>TRC_DATA7</td>
</tr>
<tr>
<td>6</td>
<td>EHRPWM2_A</td>
<td>~</td>
<td>EHRPWM2_B</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_93</td>
<td>GPIO0_47</td>
<td>GPIO0_94</td>
</tr>
<tr>
<td>8</td>
<td>GPMC0_A9</td>
<td>~</td>
<td>GPMC0_A10</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP11_AXR0</td>
<td>MCASP0_AXR2</td>
<td>MCASP11_AXR1</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

### P9.17-P9.18
### Pin  P9.17  ~  P9.18  ~  P9.19  ~  P9.20  ~
| GPIO | AC21 | 128 | 1 115 | 140 | 1 120 |
| BALL | AA3  | 22  | 217 | 22  | 22  |
| REG  | 0x00011C074 | 0x00011C1D | 0x00011C0A4 | 0x00011C1E4 |
| Page | 49 | 67 | 53 | 68 |
| MODE | PRG1_PRU1_GPO7 | SPI0_D1 | PRG1_PRU1_GPO19 | SPI1_D1 |
| 0 | ~ | ~ | ~ | ~ |
| 1 | PRG1_PRU1_GPI7 | ~ | PRG1_PRU1_GPI19 | ~ |
| 2 | PRG1_IEP1_EDC_LATCH | SPI0_SCL | PRG1_IEP1_EDC_SYNC | OUT0_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 | ~ | ~ | ~ | ~ |
| 9 | ~ | ~ | ~ | ~ |
| 10 | VOUT0_DATA7 | ~ | VOUT0_PCLK | ~ |
| 11 | VPFE0_DATA15 | ~ | ~ | ~ |
| 12 | MCASP4_AXR1 | ~ | MCASP5_AXR1 | ~ |
| 13 | ~ | ~ | ~ | ~ |
| 14 | UART3_TXD | ~ | ~ | ~ |

---

### P9.18-P9.20

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.18</th>
<th>~</th>
<th>P9.20</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 115</td>
<td>1 40</td>
<td>2 2</td>
<td>1 77</td>
</tr>
<tr>
<td>BALL</td>
<td>AA3</td>
<td>22</td>
<td>W6</td>
<td>AE25</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C1D</td>
<td>0x00011C0A4</td>
<td>0x00011C20C</td>
<td>0x00011C138</td>
</tr>
<tr>
<td>Page</td>
<td>68</td>
<td>53</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td>MCAN0_RX</td>
<td>PRG0_PRU1_GPO15</td>
<td>MCAN0_TX</td>
<td>PRG0_PRU1_GPO14</td>
</tr>
<tr>
<td>0</td>
<td>~</td>
<td>~</td>
<td>PRG0_PRU1_GPI15</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>~</td>
<td>~</td>
<td>PRG0_RGMII2_TX_CTL</td>
<td>~</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>~</td>
<td>PRG0_PWM1_B1</td>
<td>~</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>4</td>
<td>12C2_SCL</td>
<td>RMII4_TX_CTL</td>
<td>12C2_SDA</td>
<td>RMII4_TD3</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>GPIO1_1</td>
<td>GPIO1_78</td>
<td>GPIO1_2</td>
<td>GPIO0_77</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>~</td>
<td>MCASP2_AXR1</td>
<td>~</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>UART2_RTSn</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

---

### P9.21-P9.22
<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.21</th>
<th>~</th>
<th>P9.22</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 39</td>
<td>1 90</td>
<td>1 38</td>
<td>1 91</td>
</tr>
<tr>
<td>BALL</td>
<td>AJ22</td>
<td>U28</td>
<td>AC22</td>
<td>U29</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C0A0</td>
<td>0x00011C16C</td>
<td>0x00011C09C</td>
<td>0x00011C170</td>
</tr>
<tr>
<td>Page</td>
<td>52</td>
<td>56</td>
<td>52</td>
<td>54</td>
</tr>
<tr>
<td>MODE</td>
<td>0</td>
<td>PRG1_PRU1_GPO18</td>
<td>RGMII5_TDO</td>
<td>PRG1_PRU1_GPO17</td>
</tr>
<tr>
<td>1</td>
<td>PRG1_PRU1_GPI18</td>
<td>RMI17_TXD0</td>
<td>PRG1_PRU1_GPI17</td>
<td>RMI17_TX_EN</td>
</tr>
<tr>
<td>2</td>
<td>PRG1_IEP1_EDC_LATCH_IN</td>
<td>N2C3_SDA</td>
<td>PRG1_IEP1_EDC_SYNC_OUT</td>
<td>UEC6_SCL</td>
</tr>
<tr>
<td>3</td>
<td>PRG1_PWMI1_TZ_IN</td>
<td>~</td>
<td>PRG1_PWMI1_B2</td>
<td>~</td>
</tr>
<tr>
<td>4</td>
<td>SPI6_D0</td>
<td>VOUT1_DATA5</td>
<td>SPI6_CLK</td>
<td>VOUT1_DATA6</td>
</tr>
<tr>
<td>5</td>
<td>RMI16_TXD0</td>
<td>TRC_DATA3</td>
<td>RMI16_TX_EN</td>
<td>TRC_DATA4</td>
</tr>
<tr>
<td>6</td>
<td>PRG1_ECAP0_SYNC_IN</td>
<td>EHRPWM1_A</td>
<td>PRG1_ECAP0_SYNC_OUT</td>
<td>EHRPWM1_B</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_39</td>
<td>GPIO0_90</td>
<td>GPIO0_38</td>
<td>GPIO0_91</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>GPMC0_A6</td>
<td>~</td>
<td>GPMC0_A7</td>
</tr>
<tr>
<td>9</td>
<td>VOUT0_VP2_VSYNC</td>
<td>~</td>
<td>VOUT0_VP2_DE</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>VOUT0_VSYNC</td>
<td>~</td>
<td>VOUT0_DE</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>VPFE0_DATA10</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>MCASP5_AFR0</td>
<td>MCASP11_AFSX</td>
<td>MCASP5_AFSX</td>
<td>MCASP10_AXR</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>VOUT0_VP0_VSYNC</td>
<td>~</td>
<td>VOUT0_VP0_DE</td>
<td>~</td>
</tr>
</tbody>
</table>

### Boot-strap

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.23</th>
<th>P9.24</th>
<th>~</th>
<th>P9.25</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>1 10</td>
<td>1 119</td>
<td>1 13</td>
<td>1 127</td>
<td>1 104</td>
</tr>
<tr>
<td>BALL</td>
<td>AG20</td>
<td>Y5</td>
<td>AJ24</td>
<td>AC4</td>
<td>W26</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C028</td>
<td>0x00011C1E0</td>
<td>0x00011C034</td>
<td>0x00011C20</td>
<td>0x00011C1A4</td>
</tr>
<tr>
<td>Page</td>
<td>42</td>
<td>68</td>
<td>43</td>
<td>69</td>
<td>54</td>
</tr>
<tr>
<td>MODE</td>
<td>0</td>
<td>PRG1_PRU0_GPO9</td>
<td>SPI1_D0</td>
<td>PRG1_PRU0_GOUT1_CTS</td>
<td>S_GMI16_TXC</td>
</tr>
<tr>
<td>1</td>
<td>PRG1_PRU0_GPI9</td>
<td>UART5_RTSn</td>
<td>PRG1_PRU0_GPORT1_CTS</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PRG1_UART0_CTSn</td>
<td>I2C4_SCL</td>
<td>PRG1_RGMII1_TXD1</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PRG1_PWMI3_TZ_IN</td>
<td>UART2_TXD</td>
<td>PRG1_PWMI1_A0</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SPI6_CS1</td>
<td>~</td>
<td>RGMII1_TXD1</td>
<td>SPI2_D0</td>
<td>VOUT1 DE</td>
</tr>
<tr>
<td>5</td>
<td>RMI15_RXD1</td>
<td>~</td>
<td>~</td>
<td>EQEP0_S</td>
<td>TRC_DATA17</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>MCAN4_RX</td>
<td>~</td>
<td>EHRPWM4_B</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0_10</td>
<td>GPIO0_119</td>
<td>GPIO0_13</td>
<td>GPIO0_127</td>
<td>GPIO0_104</td>
</tr>
<tr>
<td>8</td>
<td>GPMC0_ADVn_ALE</td>
<td>PRG0_IEP1_EDC_LATCH_IN0</td>
<td>~</td>
<td>GPMC0_A20</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>PRG1_IEP0_EDIO_DATA_IN_OUT28</td>
<td>RGMII7_TXD1</td>
<td>~</td>
<td>VOUT1_VP0_DE</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>VOUT0_DATA23</td>
<td>~</td>
<td>VOUT0_DATA13</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>VPFE0_DATA1</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MCASP3_ACLKX</td>
<td>~</td>
<td>MCASP7_AFSX</td>
<td>MCASP10_AXR7</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
</tbody>
</table>

### Boot-strap

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.26</th>
<th>P9.27</th>
<th>~</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>P9.26</td>
<td>~</td>
<td>P9.27</td>
<td>~</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>GPIO</td>
<td>1 118</td>
<td>1 12</td>
<td>1 46</td>
<td>1 124</td>
</tr>
<tr>
<td>BALL</td>
<td>Y1</td>
<td>AF24</td>
<td>AD26</td>
<td>AB1</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C1DC</td>
<td>0x00011C030</td>
<td>0x00011C0BC</td>
<td>0x00011C1F4</td>
</tr>
<tr>
<td>Page</td>
<td>67</td>
<td>43</td>
<td>30</td>
<td>69</td>
</tr>
<tr>
<td>MODE</td>
<td>SP11_CLK</td>
<td>PRG1_PRU0_GPO11</td>
<td>PRG0_PRU0_GPO3</td>
<td>UART0_RTSn</td>
</tr>
<tr>
<td>0</td>
<td>UART5_CTSn</td>
<td>PRG1_PRU0_GPI11</td>
<td>PRG0_PRU0_GPI3</td>
<td>TIMER_IO7</td>
</tr>
<tr>
<td>1</td>
<td>12C4_SDA</td>
<td>PRG1_RGMII1_TD0</td>
<td>PRG0_RGMII1_RD3</td>
<td>SPI0_CS3</td>
</tr>
<tr>
<td>2</td>
<td>UART2_RXD</td>
<td>PRG1_PWM3_TZ_OUT</td>
<td>PRG0_PWM3_A2</td>
<td>MCAN2_TX</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>RGMII1_TDO</td>
<td>RGMII3_RD3</td>
<td>SPI2_CLK</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>~</td>
<td>RMII3_RX_ER</td>
<td>EQEP0_B</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>McAN4_TX</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>GPIO0_118</td>
<td>GPIO0_12</td>
<td>GPIO0_124</td>
</tr>
<tr>
<td>7</td>
<td>~</td>
<td>PRG0_IEP0_EDC_SYNC_OUT</td>
<td>~</td>
<td>UART3_TXD</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Boot-</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.28</th>
<th>~</th>
<th>P9.29</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>2 11</td>
<td>1 43</td>
<td>2 14</td>
<td>1 53</td>
</tr>
<tr>
<td>BALL</td>
<td>U2</td>
<td>AF28</td>
<td>V5</td>
<td>AB25</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C230</td>
<td>0x00011C0B0</td>
<td>0x00011C23C</td>
<td>0x00011C0D8</td>
</tr>
<tr>
<td>Page</td>
<td>18</td>
<td>29</td>
<td>68</td>
<td>31</td>
</tr>
<tr>
<td>MODE</td>
<td>EAP0_IN_APWM</td>
<td>PRG0_PRU0_GPO1</td>
<td>TIMER_IO1</td>
<td>PRG0_PRU0_GPO10</td>
</tr>
<tr>
<td>0</td>
<td>SYNCO_OUT</td>
<td>PRG0_PRU0_GPO3</td>
<td>EAP2_IN_APWM</td>
<td>EAP2_IN_APWM</td>
</tr>
<tr>
<td>1</td>
<td>CPTSO_RFT_CLK</td>
<td>PRG0_RGMII1_RDD_CLK</td>
<td>PRG0_UART0_RTSn</td>
<td>PRG0_UART0_RTSn</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_PWM3_A0</td>
<td>~</td>
<td>PRG0_PWM2_B1</td>
</tr>
<tr>
<td>3</td>
<td>SPI2_CS3</td>
<td>RGMII3_RD0</td>
<td>~</td>
<td>SPI3_CS2</td>
</tr>
<tr>
<td>4</td>
<td>I3C0_SDAPULLEN</td>
<td>RGMII3_RXD1</td>
<td>~</td>
<td>PRG0_IEP0_EDIO_DATA_IN_OUT29</td>
</tr>
<tr>
<td>5</td>
<td>SPI7_CS0</td>
<td>~</td>
<td>SPI7_D1</td>
<td>MCAN10_RX</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>GPMC0_Ad4</td>
</tr>
<tr>
<td>7</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Boot-</td>
<td>~</td>
<td>~</td>
<td>BOOTMODE5</td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.30</th>
<th>~</th>
<th>P9.31</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>2 11</td>
<td>1 43</td>
<td>1 53</td>
<td>1 53</td>
</tr>
<tr>
<td>BALL</td>
<td>U2</td>
<td>AF28</td>
<td>V5</td>
<td>AB25</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C230</td>
<td>0x00011C0B0</td>
<td>0x00011C23C</td>
<td>0x00011C0D8</td>
</tr>
<tr>
<td>Page</td>
<td>18</td>
<td>29</td>
<td>68</td>
<td>31</td>
</tr>
<tr>
<td>MODE</td>
<td>EAP0_IN_APWM</td>
<td>PRG0_PRU0_GPO1</td>
<td>TIMER_IO1</td>
<td>PRG0_PRU0_GPO10</td>
</tr>
<tr>
<td>0</td>
<td>SYNCO_OUT</td>
<td>PRG0_PRU0_GPO3</td>
<td>EAP2_IN_APWM</td>
<td>EAP2_IN_APWM</td>
</tr>
<tr>
<td>1</td>
<td>CPTSO_RFT_CLK</td>
<td>PRG0_RGMII1_RDD_CLK</td>
<td>PRG0_UART0_RTSn</td>
<td>PRG0_UART0_RTSn</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_PWM3_A0</td>
<td>~</td>
<td>PRG0_PWM2_B1</td>
</tr>
<tr>
<td>3</td>
<td>SPI2_CS3</td>
<td>RGMII3_RD0</td>
<td>~</td>
<td>SPI3_CS2</td>
</tr>
<tr>
<td>4</td>
<td>I3C0_SDAPULLEN</td>
<td>RGMII3_RXD1</td>
<td>~</td>
<td>PRG0_IEP0_EDIO_DATA_IN_OUT29</td>
</tr>
<tr>
<td>5</td>
<td>SPI7_CS0</td>
<td>~</td>
<td>SPI7_D1</td>
<td>MCAN10_RX</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>GPMC0_Ad4</td>
</tr>
<tr>
<td>7</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Boot-</td>
<td>~</td>
<td>~</td>
<td>BOOTMODE5</td>
<td>~</td>
</tr>
<tr>
<td>Pin</td>
<td>P9.30</td>
<td>~</td>
<td>P9.31</td>
<td>~</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>---</td>
<td>-------</td>
<td>---</td>
</tr>
<tr>
<td>GPIO</td>
<td>213</td>
<td>144</td>
<td>212</td>
<td>152</td>
</tr>
<tr>
<td>BALL</td>
<td>V6</td>
<td>AE28</td>
<td>U3</td>
<td>AB26</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C238</td>
<td>0x00011C0B4</td>
<td>0x00011C234</td>
<td>0x00011C0D4</td>
</tr>
<tr>
<td>Page</td>
<td>68</td>
<td>29</td>
<td>18</td>
<td>31</td>
</tr>
<tr>
<td>MODE</td>
<td>TIMER_IO0</td>
<td>PRG0_PRU0_GPO0</td>
<td>EXT_REFCLK</td>
<td>PRG0_PRU0_GPO9</td>
</tr>
<tr>
<td>1</td>
<td>ECAP1_IN_APWM_OUT</td>
<td>PRG0_PRU0_GPI1</td>
<td>SYNCl_OUT</td>
<td>PRG0_PRU0_GPI9</td>
</tr>
<tr>
<td>2</td>
<td>SYSCLKOUT0</td>
<td>PRG0_RGMII1_RD1</td>
<td>~</td>
<td>PRG0_UART0_CTSn</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>PRG0_PWM3_B0</td>
<td>~</td>
<td>PRG0_PWM3_TZ_IN</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>RGMII3_RD1</td>
<td>~</td>
<td>SPI3_CS1</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>RMI13_RXD0</td>
<td>~</td>
<td>PRG0_JEP0_DIO_DATA_IN_OUT</td>
</tr>
<tr>
<td>6</td>
<td>SPI7_D0</td>
<td>~</td>
<td>SPI7_CLK</td>
<td>MCAN10_TX</td>
</tr>
<tr>
<td>7</td>
<td>GPIO1_13</td>
<td>GPIO0_44</td>
<td>GPIO1_12</td>
<td>GPIO0_52</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>MCASP0_AXR1</td>
<td>~</td>
<td>MCASP0_ACLKX</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>UART6_TXD</td>
</tr>
<tr>
<td>Boot-strap</td>
<td>BOOMODE4</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

**P9.32-P9.35**

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.32</th>
<th>P9.34</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD_ADC</td>
<td>GND</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.33</th>
<th>~</th>
<th>P9.35</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>~</td>
<td>150</td>
<td>~</td>
<td>155</td>
</tr>
<tr>
<td>BALL</td>
<td>K24</td>
<td>AC28</td>
<td>K29</td>
<td>AH27</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C140</td>
<td>0x00011C0CC</td>
<td>0x00011C148</td>
<td>0x00011C0E0</td>
</tr>
<tr>
<td>Page</td>
<td>20</td>
<td>31</td>
<td>20</td>
<td>32</td>
</tr>
<tr>
<td>MODE</td>
<td>MCU_ADC0_AIN4</td>
<td>PRG0_PRU0_GPO7</td>
<td>MCU_ADC0_AIN6</td>
<td>PRG0_PRU0_GPO12</td>
</tr>
<tr>
<td>1</td>
<td>~</td>
<td>PRG0_PRU0_GPI7</td>
<td>~</td>
<td>PRG0_PRU0_GPI12</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_JEP0_EDC_LATCH_IN</td>
<td>~</td>
<td>PRG0_RGMII1_TD1</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>PRG0_RGMII1_B1</td>
<td>~</td>
<td>PRG0_PWM0_A0</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>PRG0_ECAP0_SYNC_IN</td>
<td>~</td>
<td>RGMII3_TD1</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>MCAN9_TX</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>~</td>
<td>GPIO0_50</td>
<td>~</td>
<td>GPIO0_55</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>GPMC0_AD1</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>DSS_FSYNC0</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>MCASP0_AXR5</td>
<td>~</td>
<td>MCASP0_AXR8</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Boot-strap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

**P9.36-P9.37**

2.5. BeagleBone AI-64

201
<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.36</th>
<th>~</th>
<th>P9.37</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>~</td>
<td>1 56</td>
<td>~</td>
<td>1 57</td>
</tr>
<tr>
<td>BALL</td>
<td>K27</td>
<td>AH29</td>
<td>K28</td>
<td>AG28</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C144</td>
<td>0x00011C0E4</td>
<td>0x00011C138</td>
<td>0x00011C0E8</td>
</tr>
<tr>
<td>Page</td>
<td>20</td>
<td>32</td>
<td>20</td>
<td>32</td>
</tr>
<tr>
<td>MODE 0</td>
<td>MCU_ADC0_AIN5</td>
<td>PRG0_PRU0_GPO13</td>
<td>MCU_ADC0_AIN2</td>
<td>PRG0_PRU0_GPO14</td>
</tr>
<tr>
<td>1</td>
<td>~</td>
<td>PRG0_PRU0_GPI13</td>
<td>~</td>
<td>PRG0_PRU0_GPI14</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_RGMII1_TX_CTL</td>
<td>~</td>
<td>PRG0_RGMII1_TD0</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>PRG0_PWM0_B0</td>
<td>~</td>
<td>PRG0_PWM0_A1</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>RGMII3_TX_CTL</td>
<td>~</td>
<td>RGMII3_TD0</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>~</td>
<td>GPIO0_56</td>
<td>~</td>
<td>GPIO0_57</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>UART4_TXD</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>DSS_FSYNC2</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>MCASP0_AXR9</td>
<td>~</td>
<td>MCASP0_AXR10</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.38</th>
<th>~</th>
<th>P9.39</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>~</td>
<td>1 58</td>
<td>~</td>
<td>1 54</td>
</tr>
<tr>
<td>BALL</td>
<td>L28</td>
<td>AG27</td>
<td>K25</td>
<td>AJ28</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C13C</td>
<td>0x00011C0EC</td>
<td>0x00011C130</td>
<td>0x00011C0DC</td>
</tr>
<tr>
<td>Page</td>
<td>~</td>
<td>33</td>
<td>20</td>
<td>32</td>
</tr>
<tr>
<td>MODE 0</td>
<td>MCU_ADC0_AIN3</td>
<td>PRG0_PRU0_GPO15</td>
<td>MCU_ADC0_AIN0</td>
<td>PRG0_PRU0_GPO11</td>
</tr>
<tr>
<td>1</td>
<td>~</td>
<td>PRG0_PRU0_GPI15</td>
<td>~</td>
<td>PRG0_PRU0_GPI11</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_RGMII1_TX_CTL</td>
<td>~</td>
<td>PRG0_RGMII1_TD0</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>PRG0_PWM0_B1</td>
<td>~</td>
<td>PRG0_PWM3_TZ_OUT</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>RGMII3_TX_CTL</td>
<td>~</td>
<td>RGMII3_TD0</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>~</td>
<td>GPIO0_56</td>
<td>~</td>
<td>GPIO0_54</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>UART4_TXD</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>CLKOUT</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>DSS_FSYNC3</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>MCASP0_AXR11</td>
<td>~</td>
<td>MCASP0_AXR7</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.40</th>
<th>~</th>
<th>P9.41</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>~</td>
<td>1 54</td>
<td>~</td>
<td>1 59</td>
</tr>
<tr>
<td>BALL</td>
<td>K25</td>
<td>AG27</td>
<td>K25</td>
<td>AJ28</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C0E8</td>
<td>0x00011C0E8</td>
<td>0x00011C0E8</td>
<td>0x00011C0E8</td>
</tr>
<tr>
<td>Page</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>MODE 0</td>
<td>MCU_ADC0_AIN4</td>
<td>PRG0_PRU0_GPO17</td>
<td>MCU_ADC0_AIN1</td>
<td>PRG0_PRU0_GPO18</td>
</tr>
<tr>
<td>1</td>
<td>~</td>
<td>PRG0_PRU0_GPI17</td>
<td>~</td>
<td>PRG0_PRU0_GPI18</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_RGMII1_TX_CTL</td>
<td>~</td>
<td>PRG0_RGMII1_TD0</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>PRG0_PWM0_B1</td>
<td>~</td>
<td>PRG0_PWM3_TZ_OUT</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>RGMII3_TX_CTL</td>
<td>~</td>
<td>RGMII3_TD0</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>~</td>
<td>GPIO0_54</td>
<td>~</td>
<td>GPIO0_54</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>UART4_TXD</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>DSS_FSYNC4</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>MCASP0_AXR11</td>
<td>~</td>
<td>MCASP0_AXR7</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>
### GPIO and Ball Assignment

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.40</th>
<th>~</th>
<th>P9.41</th>
<th>P9.42</th>
<th>~</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO</td>
<td>~</td>
<td>1 81</td>
<td>2 0</td>
<td>1 123</td>
<td>1 18</td>
</tr>
<tr>
<td>BALL</td>
<td>K26</td>
<td>AA26</td>
<td>AD5</td>
<td>AC2</td>
<td>AJ21</td>
</tr>
<tr>
<td>REG</td>
<td>0x00011C134</td>
<td>0x00011C148</td>
<td>0x00011C204</td>
<td>0x00011C1F0</td>
<td>0x00011C04C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode</th>
<th>MCU_ADC0_AIN1</th>
<th>PRG0_PRU1_GPO18</th>
<th>UART1_RTSn</th>
<th>UART0_CTSn</th>
<th>PRG1_PRU0_GPO17</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>~</td>
<td>PRG0_PRU1_GPI18</td>
<td>MCAN3_TX</td>
<td>TIMER_IO6</td>
<td>PRG1_PRU0_GPI17</td>
</tr>
<tr>
<td>2</td>
<td>~</td>
<td>PRG0_IEP1_EDC_LATCH_IN0</td>
<td>SPI0_CS2</td>
<td>PRG1_IEP0_EDC_SYNC_OUT1</td>
<td>~</td>
</tr>
<tr>
<td>3</td>
<td>~</td>
<td>PRG0_PWM1_TZ_IN</td>
<td>~</td>
<td>MCAN2_RX</td>
<td>PRG1_PWM0_B2</td>
</tr>
<tr>
<td>4</td>
<td>~</td>
<td>SPI3_D0</td>
<td>SPI2_D1</td>
<td>SPI2_CS0</td>
<td>~</td>
</tr>
<tr>
<td>5</td>
<td>~</td>
<td>~</td>
<td>EQEP0_I</td>
<td>EQEP0_A</td>
<td>RMII5_TXD1</td>
</tr>
<tr>
<td>6</td>
<td>~</td>
<td>MCAN12_TX</td>
<td>~</td>
<td>~</td>
<td>MCAN5_TX</td>
</tr>
<tr>
<td>7</td>
<td>~</td>
<td>GPIO0_81</td>
<td>GPIO1_0</td>
<td>GPIO0_123</td>
<td>GPIO0_18</td>
</tr>
<tr>
<td>8</td>
<td>~</td>
<td>GPMC0_AD14</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>10</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>11</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>VPF0_DATA6</td>
</tr>
<tr>
<td>12</td>
<td>~</td>
<td>MCASP2_AFSX</td>
<td>~</td>
<td>~</td>
<td>MCASP3_AXR3</td>
</tr>
<tr>
<td>13</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>14</td>
<td>~</td>
<td>UART2_RXD</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

#### Bootstrapping

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>

2.5.7 BeagleBone AI-64 Mechanical

**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**

2.5.8 Pictures

2.5.9 Support Information

All support for this design is through BeagleBoard.org community at: link: BeagleBoard.org forum.

**Hardware Design**

You can find all BeagleBone AI-64 hardware files here.
Fig. 2.100: Fig: Board Dimensions

Fig. 2.101: Fig: Top silkscreen
2.5. BeagleBone AI-64

Fig. 2.102: Fig: Bottom silkscreen

Fig. 2.103: Fig: BeagleBone AI-64 front

Fig. 2.104: Fig: BeagleBone AI-64 back
Fig. 2.105: Fig: BeagleBone AI-64 back with heatsink

Fig. 2.106: Fig: BeagleBone AI-64 front at 45° angle

Fig. 2.107: Fig: BeagleBone AI-64 back at 45° angle
Fig. 2.108: Fig: BeagleBone AI-64 back with heatsink at 45° angle

Fig. 2.109: Fig: BeagleBone AI-64 ports
Software Updates

You can download and flash the supported image onto your BeagleBone AI-64 from this source.

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
```

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.

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 Fig: Display adaptors.

Getting Help

If you need some up to date troubleshooting techniques, you can post your queries on link: BeagleBoard.org forum

2.5.10 Update software on BeagleBone AI-64

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)...

```
1 sudo apt update
2 sudo apt install --only-upgrade bb-j721e-evm-firmware generic-sys-mods
3 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...

```
1 sudo /opt/u-boot/bb-u-boot-beagleboneai64/install-emmc.sh
2 sudo /opt/u-boot/bb-u-boot-beagleboneai64/install-microsd.sh
3 sudo reboot
```

Update Kernel and SGX modules:

```
1 sudo apt install bbb.io-kernel-5.10-ti-k3-j721e
```

Update xfce:

```
1 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
```

### 2.5.11 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 and 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](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**
  - Nominal Output Voltage: 5VDC
  - Maximum Output Current: 5000 mA

Connect the components to the SK as shown in the image.

**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 pub_edgeai_multiple_usb_cams to stream from multiple USB cameras simultaneously.

---

**2.5. BeagleBone AI-64**

---

209
**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 SRGB8 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

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 usecases

Please refer pub_edgeai_camera_sources 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

```bash
CSI_CAM_0_FMT='[fmt:SRGBB8_1X10/1920x1080]' CSI_CAM_1_FMT='[fmt:SRGBB8_1X10/1920x1080]'
```

- Change the imaging binaries to use 10 bit versions

```bash
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-11.4-xfce-edgeai-arm64-2022-08-02-10gb.img.xz
  • To flash on eMMC: bbai64-emmc-flasher-debian-11.4-xfce-edgeai-arm64-2022-08-02-10gb.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.

![Balena Etcher tool](image)

Fig. 2.111: Balena Etcher tool to flash SD card with Processor linux image Linux for Edge AI

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

```
#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
```

(continues on next page)
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.

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

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 pub_edgeai_configuration

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,

```
1 # go to edge-ai-apps folder
2 debian@beaglebone:~$ cd /opt/edge_ai_apps/apps_python
3
4 # enable root (password: temppwd)
5 debian@beaglebone:~$ sudo su
6 [sudo] password for beaglebone:
7
8 # use edge-ai-apps
9 debian@beaglebone:/opt/edge_ai_apps/apps_python# 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 it's **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.

![Model Downloader Tool Menu](image)

**Fig. 2.112: 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-miperf-300x300
```

| param.yaml |

(continues on next page)
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 `pub_edgeai_compile_artifacts`
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 in `pub_edgeai_params`

**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](https://edgeai-benchmark)

**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.

```plaintext
inputs:
  input0:
    source: /dev/video2
    #Device file entry of the camera
    format: jpeg
    #Input data format supported by the camera
    width: 1280
    height: 720
    framerate: 30
    #Width and Height of the input
    #Framerate of the source

  input1:
    source: ../data/videos/video_0000_h264.mp4
    #Video file
    format: h264
    #File encoding format
    width: 1280
    height: 720
    framerate: 25
    #Video Input

  input2:
    source: ../data/images/%04d.jpg
    #Sequence of Image files, printf style formatting is used
    width: 1280
    height: 720
    index: 0
    framerate: 1
    #Starting Index (optional)
```

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:

```bash
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
black_level_in_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
    format: rggb12       #ISP will be added in the pipeline
    width: 1936
    height: 1100
    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 sequentially and incrementally and the demo plays the files at the fps specified by the user.

```plaintext
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.

```plaintext
input0:
  source: rtsp://172.24.145.220:8554/test # rtsp stream url, replace this with...
  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.

```plaintext
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
   
   \[\text{alpha} \times \text{mask} + (1 - \text{alpha}) \times \text{image}\]

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 pub_edgeai_import_custom_models

Outputs  The output section defines a list of supported outputs.
outputs:
  output0: # Display Output
    sink: kmssink
    width: 1920 # Width and Height
    height: 1080 # of the output
    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.

```bash
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.

```plaintext
output1:
    sink: ../data/output/videos/output_video.mkv # Output video file
    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.

```plaintext
output2:
    sink: ../data/output/images/output_image_%04d.jpg # Image file name,
        printf style formatting is used
    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.

2.5. BeagleBone AI-64
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_
      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 pub_edgeai_configuration.

**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,

![Sample output showing single input, multi-inference output](image)

Fig. 2.113: Sample output showing single input, multi-inference output

We can specify the output window location and sizes as shown in the configuration file,

```yaml
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
      mosaic3:
```

(continues on next page)
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,

![Sample output showing multi-input, multi-inference output](image)

We can specify the output window location and sizes as shown in the configuration file,

```yaml
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)
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 `pub_edgeai_install_dependencies` 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.

```bash
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 and 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/daemon.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.

   ```json
   {
   "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

   ```bash
   $ 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

---

2.5. BeagleBone AI-64 225
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 `pub_edgeai_configuration`.

**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:**

```gstreamer
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 ! jpegdec ! tiovxdicolorconvert ! video/x-raw, format=NV12 ! tiovxmultiscaler ! name=split_01
split_01. ! queue video/x-raw, width=454, height=256 ! tiovxdicolorconvert out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115 top=16 bottom=16 ! tiovxdipreproc 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 ! tiovxdicolorconvert ! target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_0 max-buffers=2 drop=true
```

**GStreamer output pipeline:**

```gstreamer
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true name=post_0 ! tiovxdicolorconvert ! 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 ! tiovxdicolorconvert ! 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
```

![Image classification dataflow](image_classification_dataflow.png)

Fig. 2.115: GStreamer based data-flow pipeline for image classification demo with USB camera and display

226 Chapter 2. Boards
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:

```
appsrc device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !
jpegdec ! tiovxcolorconvert ! video/x-raw, format=NV12 ! tiovxmultiscaler
.split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxpreproc data-type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.000000 !
.scale=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 ! tiovxcolorconvert !
split_01 ! queue ! video/x-raw, width=1280, height=720 ! tiovxmosaic name=mosaic_0 ! appsink name=mosaic_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 ! tiovxcolorconvert ! 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 !
tiovxcolorconvert ! 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 Dataflow

![Object Detection Dataflow Diagram](beagleboard_docs/object_detection_diagram.png)

Fig. 2.116: 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.
Fig. 2.117: 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/video18 io-mode=2 ! image/jpeg, width=1280, height=720 ! jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 ! tiovxmultiscaler ! name=spl_01
```

(continues on next page)
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:

```
! queue ! video/x-raw, width=320, height=320 ! tiovxdpreproc 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=2.000000 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
```

Fig. 2.118: GStreamer based data-flow pipeline for Human Pose Estimation demo with USB camera and display

2.5. BeagleBone AI-64
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:

```gapher
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 ! queue ! video/x-raw ! appsink name=pre_0 max-buffers=2 drop=true
```

Fig. 2.119: GStreamer based data-flow pipeline with video file input source and display

GStreamer output pipeline:

```gapher
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true name=post_ ! 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
```
GStreamer output pipeline:

```gstreamer
appsrc format=GSTREAMER_FORMAT_TIME is-live=true block=true do-timestamp=true name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280, height=720 ! tiovxdlpreproc ! queue ! video/x-raw, format=RGB ! appsink name=sen_0 max-buffers=2 drop=true
```

Fig. 2.120: GStreamer based data-flow pipeline with RTSP based video file source and display

**RTIv2 Camera Sensor (IMX219)** In this demo, raw frames in SRGGB8 format are captured from RTIv2 (imx219) 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. 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:

```gstreamer
v4l2src device=/dev/video2 io-mode=5 ! video/x-bayer, width=1920, height=1080, ! format=rgb ! 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-r raw, format=NV12 ! tiovxmultiscaler ! video/x-raw, width=1280, height=720 ! tiovxmultiscaler name=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=
```

(continues on next page)
IMX390 Camera Sensor  In this demo, raw frames in SRGB12 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:

```bash
```
Fig. 2.122: 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:

```
v412src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !
jpegdec ! tiovxcolorconvert ! video/x-raw, format=NV12 ! tiovxmultiscaler
name=split_01 split_01. ! queue ! video/x-raw, width=512, height=512 ! tiovxpreproc 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
```

(continues on next page)
GStreamer output pipeline:

```gstreamer
appsrc format=GSTREAMER_TIME is-live=true block=true do-timestamp=true name=post_0 ! tiovxdcolorconvert ! video/x-raw,format=NV12,width=1280,height=720 !
```

```gstreamer
appsrc format=GSTREAMER_TIME block=true num-buffers=1 name=background_0 !
```

```gstreamer
tiovxdcolorconvert ! video/x-raw,format=NV12,width=1920,height=1080 ! queue !
```

```gstreamer
tiovxmosaic name=mosaic_0
sink_0::startx=320 sink_0::starty=180 sink_0::width=1280 sink_0::height=720 !
```

```gstreamer
video/x-raw,format=NV12 ! v4l2h264enc bitrate=1000000 !
```

```gstreamer
descriptor ! matroskamux ! filesink location=/opt/edge_ai_apps/data/output/videos/output_video.mkv
```

---

**Video output, Single inference**

![Image of video output pipeline](image)

---

**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, 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:

```gstreamer
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !
```

```gstreamer
jpegdec ! tiovxdcolorconvert ! video/x-raw, format=NV12 !
```

```gstreamer
tee_split0. queue ! tiovxmultiscaler name=split_01 tee_split0. ! tee_split0. ! tee_split0. !
```

```gstreamer
appsink
```

(continues on next page)
tee_split0. ! queue ! tiovxmultiscaler name=split_03
split_01. ! queue ! video/x-raw, width=512, height=512 ! tiovxdpreproc data-
type=10 channel=order-0 mean=0-128.000000 mean-1=128.000000 mean-2=128.000000...
->scale=0.015625 scale-1=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 ! tiovxcolorconvert...
->target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=pre_1 max-
buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=320, height=320 ! tiovxdpreproc data-
type=10 channel=order-1 mean=0=128.000000 mean=1=128.000000 mean-2=128.000000...
->scale=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_02. ! queue ! video/x-raw, width=640, height=360 ! tiovxcolorconvert...
->target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_0 max-
buffers=2 drop=true
split_03. ! queue ! video/x-raw, width=454, height=256 ! tiovxcolorconvert out-
pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115 top=16...
->bottom=16 ! tiovxdpreproc data-type=10 channel-order-1 mean=0=128.000000 mean=1=128.000000 mean-2=128.000000 scale=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 ! tiovxcolorconvert...
->target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_1 max-
buffers=2 drop=true
split_04. ! queue ! video/x-raw, width=454, height=256 ! tiovxcolorconvert out-
pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115 top=16...
->bottom=16 ! tiovxdpreproc data-type=10 channel-order-0 mean=0=123.675000 mean-
1=116.280000 mean=2=103.530000 scale=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 ! tiovxcolorconvert...
->target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_2 max-
buffers=2 drop=true
GStreamer output pipeline:
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true name=post-
0 ! tiovxcolorconvert ! 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 ! tiovxcolorconvert ! 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 ! tiovxcolorconvert ! 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 ! tiovxcolorconvert ! 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 !
! tiovxcolorconvert ! 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
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 ! tiovxdicolorconvert ! video/x-raw, format=NV12 ! tee name=tee_split0
   tee_split0. ! queue ! tiovxmultiscaler name=split_01
split_01. ! queue ! tiovxdicolorconvert ! video/x-raw, format=NV12 ! tee name=tee_split0
split_01. ! queue ! tiovxmultiscaler name=split_02
split_02. ! queue ! video/x-raw, width=640, height=360 ! tiovxdilpreproc data-
   type=10 channel-order=1 mean=0-128.000000 mean=0-128.000000 mean=0-128.000000.
   ! 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 ! tiovxdicolorconvert.
   ! 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 ! tiovxdicolorconvert out-
   pool-size=4 ! video/x-raw, format=RGB ! vboxleft=115 right=115 top=16-
   ! bottom=16 ! tiovxdilpreproc data-type=10 channel-order=1 mean=0-128.000000 mean-
   =0-128.000000 mean=0-128.000000 mean=0-128.000000 mean=0-128.000000.
   ! 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 ! tiovxdicolorconvert.
   ! target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_1 max-
   buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GSTREAMER_FORMAT_TIME is-live=true block=true do-timestamp=true name=post_-
   0 ! tiovxdicolorconvert ! video/x-raw,format=NV12, width=640, height=360 ! queue
   ! mosaic_0.sink_0
appsrc format=GSTREAMER_FORMAT_TIME is-live=true block=true do-timestamp=true name=post_-
   1 ! tiovxdicolorconvert ! video/x-raw,format=NV12, width=640, height=360 ! queue.
```
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 pub_edgeai_available_statistics 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 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 pub-edgeai_docker_env 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 exec 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 exec 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. 2.124: 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 here, pub_edgeai_sdk_development_flow

Fig. 2.125: 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.

dedgeai-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

dedgeai-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 the pre-imported models and related documentation please visit: https://github.com/TexasInstruments/edgeai-modelzoo

dedge_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 pub_edgeai_install_dependencies 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 pub_edgeai_install_dependencies 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:

Source : USB Camera  Capture Framerate : 30 fps Resolution : 720p format : JPEG
Object detection dataflow

Fig. 2.126: GStreamer based data-flow pipeline with USB camera input and display output

<table>
<thead>
<tr>
<th>Model</th>
<th>FPS</th>
<th>Total time (ms)</th>
<th>Interference (ms)</th>
<th>A72 Load (%)</th>
<th>DDR Read BW (MB/s)</th>
<th>DDR Write BW (MB/s)</th>
<th>DDR C71 Load (%)</th>
<th>C66_0 Load (%)</th>
<th>C66_1 Load (%)</th>
<th>MCU2_0 Load (%)</th>
<th>MCU2_1 Load (%)</th>
<th>MSC_0 (%)</th>
<th>MSC_1 (%)</th>
<th>VISS (%)</th>
<th>NF (%)</th>
<th>LDC (%)</th>
<th>SDE (%)</th>
<th>DOF (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONR-CL-6150-mobileNetV2-1p4qat</td>
<td>30</td>
<td>3.01</td>
<td>22.10</td>
<td>21.0</td>
<td>1596</td>
<td>619</td>
<td>2215</td>
<td>9.0</td>
<td>20.0</td>
<td>9.0</td>
<td>6.0</td>
<td>1.0</td>
<td>22.170</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TFL-CL-0000-mobileNetV1mlperf</td>
<td>30</td>
<td>33.33</td>
<td>1.04</td>
<td>15.9</td>
<td>1425</td>
<td>563</td>
<td>1988</td>
<td>5.0</td>
<td>22.0</td>
<td>9.0</td>
<td>6.0</td>
<td>1.0</td>
<td>21.900</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320x320</td>
<td>30</td>
<td>393.33</td>
<td>55.00</td>
<td>10.2</td>
<td>1534</td>
<td>570</td>
<td>2104</td>
<td>15.0</td>
<td>29.0</td>
<td>9.0</td>
<td>6.0</td>
<td>1.0</td>
<td>22.670</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TVM-CL-3410-gluncnv-mxnet-mobv2</td>
<td>30</td>
<td>333.33</td>
<td>12.02</td>
<td>22.8</td>
<td>1522</td>
<td>617</td>
<td>2139</td>
<td>6.0</td>
<td>20.0</td>
<td>9.0</td>
<td>6.0</td>
<td>1.0</td>
<td>21.840</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Source: Video  Video Framerate: 30 fps Resolution: 720p Encoding: h264
Fig. 2.127: GStreamer based data-flow pipeline with video file input source and display output
CSI Camera (OV5640) input

Fig. 2.128: GStreamer based data-flow pipeline for with CSI camera (OV5640) input and display output

<table>
<thead>
<tr>
<th>Model</th>
<th>FPS</th>
<th>Total time (ms)</th>
<th>Inference time (ms)</th>
<th>A72 Load (%)</th>
<th>DDR Read BW (MB/s)</th>
<th>DDR Write BW (MB/s)</th>
<th>DDR C71 Load (%)</th>
<th>C66x Load (%)</th>
<th>C66y Load (%)</th>
<th>MCU Load (%)</th>
<th>VISS (%)</th>
<th>NF (%)</th>
<th>LDC (%)</th>
<th>SDE (%)</th>
<th>DOF (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONR-CL-6150- mobileNetV2-1p4-qat</td>
<td>29.57</td>
<td>734.09</td>
<td>93.02</td>
<td>12.2</td>
<td>1167</td>
<td>699</td>
<td>2370</td>
<td>8.0</td>
<td>45.0</td>
<td>9.0</td>
<td>6.0</td>
<td>1.0</td>
<td>21.35</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TFL-CL-0000- mobileNetV1-nlperf</td>
<td>29.4</td>
<td>134.35</td>
<td>51.01</td>
<td>10.2</td>
<td>1502</td>
<td>645</td>
<td>2147</td>
<td>5.0</td>
<td>47.0</td>
<td>9.0</td>
<td>6.0</td>
<td>1.0</td>
<td>20.56</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320-320</td>
<td>29.3</td>
<td>634.65</td>
<td>55.00</td>
<td>10.5</td>
<td>1619</td>
<td>655</td>
<td>2268</td>
<td>14.0</td>
<td>53.0</td>
<td>9.0</td>
<td>6.0</td>
<td>1.0</td>
<td>21.47</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>29.3</td>
<td>534.17</td>
<td>72.01</td>
<td>11.6</td>
<td>1596</td>
<td>698</td>
<td>2296</td>
<td>6.0</td>
<td>45.0</td>
<td>9.0</td>
<td>5.0</td>
<td>1.0</td>
<td>21.10</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Source: CSI Camera with VISS (imx219)  Capture Framerate: 30 fps  Resolution: 1080p format: SRGGB8
RPIv2 (IMX219) Sensor

![Diagram of GStreamer-based data-flow pipeline with IMX219 sensor, ISP, and display](image)

Fig. 2.129: GStreamer based data-flow pipeline with IMX219 sensor, ISP and display

<table>
<thead>
<tr>
<th>Model</th>
<th>FPS Total time (ms)</th>
<th>Inference time (ms)</th>
<th>A72 Load (%)</th>
<th>DDR Read BW (MB/s)</th>
<th>DDR Write BW (MB/s)</th>
<th>DDR C71 Load (%)</th>
<th>C66x Load (%)</th>
<th>MCU Load (%)</th>
<th>SCC Load (%)</th>
<th>MSC Load (%)</th>
<th>VISS Load (%)</th>
<th>NF Load (%)</th>
<th>LDC Load (%)</th>
<th>SDE Load (%)</th>
<th>DOF Load (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONR-CL-6150-ONN-1-p4-qat</td>
<td>30.64</td>
<td>433.3</td>
<td>93.01</td>
<td>15.7</td>
<td>2178</td>
<td>853</td>
<td>2634</td>
<td>9.0</td>
<td>16.0</td>
<td>9.0</td>
<td>13.0</td>
<td>1.0</td>
<td>31.78</td>
<td>22.370</td>
<td>0</td>
</tr>
<tr>
<td>TFL-CL-0000-TFL-1-mmlperf</td>
<td>30.59</td>
<td>33.1</td>
<td>41.04</td>
<td>12.7</td>
<td>8161</td>
<td>798</td>
<td>2410</td>
<td>5.0</td>
<td>18.0</td>
<td>9.0</td>
<td>13.0</td>
<td>1.0</td>
<td>31.650</td>
<td>22.190</td>
<td>0</td>
</tr>
<tr>
<td>TFL-OD-2020-TFL-1-mmlDet-DSP-coco-320x320</td>
<td>30.56</td>
<td>33.1</td>
<td>75.00</td>
<td>13.3</td>
<td>1730</td>
<td>809</td>
<td>2539</td>
<td>15.0</td>
<td>25.0</td>
<td>9.0</td>
<td>13.0</td>
<td>1.0</td>
<td>32.6</td>
<td>22.190</td>
<td>0</td>
</tr>
<tr>
<td>TVM-CL-3410-TVM-1-glucnvc-mxet-mobv2</td>
<td>30.48</td>
<td>33.1</td>
<td>42.01</td>
<td>12.9</td>
<td>11708</td>
<td>852</td>
<td>2560</td>
<td>7.0</td>
<td>16.0</td>
<td>9.0</td>
<td>13.0</td>
<td>1.0</td>
<td>31.830</td>
<td>22.260</td>
<td>0</td>
</tr>
</tbody>
</table>

Source: IMX390 over FPD-Link  Capture Framerate: 30 fps  Resolution: 1080p  format: SRGBB12
Fig. 2.130: GStreamer based data-flow pipeline with IMX390 sensor, ISP, LDC and display
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

<table>
<thead>
<tr>
<th>Category</th>
<th># test case</th>
<th>Pass</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host OS - Python</td>
<td>99</td>
<td>99</td>
<td>0</td>
</tr>
<tr>
<td>Host OS - C++</td>
<td>99</td>
<td>99</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S.No</th>
<th>Models</th>
<th>Input</th>
<th>Output</th>
<th>Host OS-C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>2</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>Image</td>
<td>Video-Filewrite</td>
<td>Fail</td>
</tr>
<tr>
<td>3</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>Image</td>
<td>Image-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>4</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>5</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>6</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>S.No</td>
<td>Models</td>
<td>Input</td>
<td>Output</td>
<td>Host OS-C++</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>7</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>8</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>9</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>CSI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>10</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>RTSP Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>11</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>RTSP Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>12</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>RTSP Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>13</td>
<td>TVM-CL-3410-gluoncv-mxnet-mobv2</td>
<td>RTSP Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>14</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>15</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>Image</td>
<td>Video-Filewrite</td>
<td>Fail</td>
</tr>
<tr>
<td>16</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>Image</td>
<td>Image-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>17</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>18</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>19</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>20</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>21</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>22</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>CSI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>23</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>RPI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>24</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>25</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>RTSP Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>26</td>
<td>TFL-CL-0000-mobileNetV1-mlperf</td>
<td>RTSP Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>27</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>28</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>Image</td>
<td>Video-Filewrite</td>
<td>Fail</td>
</tr>
<tr>
<td>29</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>Image</td>
<td>Image-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>30</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>31</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>32</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>33</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>34</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>35</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>CSI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>36</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>RPI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>37</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>38</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>RTSP Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>39</td>
<td>ONR-CL-6360-regNetx-200mf</td>
<td>RTSP Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>40</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>41</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>Image</td>
<td>Video-Filewrite</td>
<td>Fail</td>
</tr>
<tr>
<td>42</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>Image</td>
<td>Image-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>43</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>44</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>45</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>46</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>47</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>48</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>CSI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>49</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>RPI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>50</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>51</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>RTSP Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>52</td>
<td>TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416</td>
<td>RTSP Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>53</td>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320x320</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>54</td>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320x320</td>
<td>Image</td>
<td>Video-Filewrite</td>
<td>Fail</td>
</tr>
<tr>
<td>55</td>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320x320</td>
<td>Image</td>
<td>Image-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>56</td>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320x320</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>57</td>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320x320</td>
<td>Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>58</td>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320x320</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>59</td>
<td>TFL-OD-2020-ssdlite-mobDet-DSP-coco-320x320</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>S.No</td>
<td>Models</td>
<td>Input</td>
<td>Output</td>
<td>Host OS-C++</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>60</td>
<td>TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>61</td>
<td>TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>62</td>
<td>TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>63</td>
<td>TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>64</td>
<td>TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>65</td>
<td>TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320</td>
<td>RTSP - Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>66</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>67</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>68</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>Image</td>
<td>Video-Filewrite</td>
<td>Fail</td>
</tr>
<tr>
<td>69</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>70</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>71</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>72</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>73</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>74</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>CSI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>75</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>RPI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>76</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>77</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>RTSP - Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>78</td>
<td>ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512</td>
<td>RTSP - Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>79</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>80</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>81</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>82</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>83</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>84</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>85</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>86</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>87</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>CSI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>88</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>RPI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>89</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>90</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>RTSP - Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>91</td>
<td>TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512</td>
<td>RTSP - Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>92</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>93</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>94</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>Image</td>
<td>Video-Filewrite</td>
<td>Fail</td>
</tr>
<tr>
<td>95</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>96</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>97</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>98</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>99</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>100</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>CSI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>101</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>RPI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>102</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>103</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>RTSP - Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>104</td>
<td>TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512</td>
<td>RTSP - Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>105</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>106</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>Image</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>107</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>Image</td>
<td>Video-Filewrite</td>
<td>Fail</td>
</tr>
<tr>
<td>108</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>109</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>Video</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>110</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>USB Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
<tr>
<td>111</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>USB Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
</tr>
<tr>
<td>112</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>CSI Camera</td>
<td>Display</td>
<td>Pass</td>
</tr>
</tbody>
</table>
### Table 2.39 – continued from previous page

<table>
<thead>
<tr>
<th>S.No</th>
<th>Models</th>
<th>Input</th>
<th>Output</th>
<th>Host OS-C++</th>
<th>Host OS-Python</th>
<th>Docker-C++</th>
<th>Docker-Python</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>CSI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>RPI Camera</td>
<td>Display</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>RPI Camera</td>
<td>Video-Filewrite</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>RTSP - Video</td>
<td>Display</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512</td>
<td>RTSP - Video</td>
<td>Video-Filewrite</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td></td>
</tr>
</tbody>
</table>

### Single Input Multi Output

<table>
<thead>
<tr>
<th>Category</th>
<th># test case</th>
<th>Pass</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host OS - Python</td>
<td>15</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>docker - Python</td>
<td>15</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Host OS - C++</td>
<td>15</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Docker - C++</td>
<td>15</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

### Multi Input Multi Output

<table>
<thead>
<tr>
<th>Category</th>
<th># test case</th>
<th>Pass</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host OS - Python</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>docker - Python</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Host OS - C++</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Docker - C++</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>S.No</td>
<td>Models</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>1</td>
<td>2 Models (TVM-CL, TFL-OD)</td>
<td>%04d.jpg,video_0000.mp4</td>
<td>Display</td>
</tr>
<tr>
<td>2</td>
<td>2 Models (TVM-OD, ONR-SS)</td>
<td>%04d.jpg,rtsp</td>
<td>Video-Filewrite</td>
</tr>
<tr>
<td>3</td>
<td>2 Models (ONR-CL, TVM-SS)</td>
<td>%04d.jpg,USB_camera</td>
<td>Display</td>
</tr>
<tr>
<td>4</td>
<td>3-Models (TVM-CL, TFL-OD, ONR-SS)</td>
<td>%04d.jpg,CSI_camera</td>
<td>Video-Filewrite</td>
</tr>
<tr>
<td>5</td>
<td>3-Models (TVM-CL, TFL-OD, ONR-SS)</td>
<td>video_0000.mp4,rtsp</td>
<td>Display</td>
</tr>
<tr>
<td>7</td>
<td>4-Models (TVM-CL, TFL-SS, ONR-OD, TFL-CL)</td>
<td>USB_camera,CSI_camera</td>
<td>Display</td>
</tr>
<tr>
<td>8</td>
<td>4-Models (TVM-SS, TFL-SS, ONR-SS, ONR-OD)</td>
<td>USB_camera,video_0000.mp4</td>
<td>Video-Filewrite</td>
</tr>
</tbody>
</table>

**Note:**
- Video file from RTSP server used for RTSP input test
- Please refer to the pub_edgeai_known_issues section for more details

### 2.6 PocketBeagle

**Contributors**

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

- Maintaining author: Jason Kridner
- Contributing Editor: Cathy Wicks

**Note:** Make sure to read and accept all the terms & condition provided in the boards-terms-and-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.

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.
2.6.1 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.

2.6.2 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.
**Document Change History**

**Table 2.40: Change History**

<table>
<thead>
<tr>
<th>Rev</th>
<th>Changes</th>
<th>Date</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.x</td>
<td>Production Document</td>
<td>December 7, 2017</td>
<td>JK</td>
</tr>
<tr>
<td>0.0.5</td>
<td>Converted to .rst and gitlab hosting</td>
<td>July 21, 2022</td>
<td>DK</td>
</tr>
</tbody>
</table>

**Board Changes**

**Table 2.41: Board History**

<table>
<thead>
<tr>
<th>Rev</th>
<th>Changes</th>
<th>Date</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Preliminary</td>
<td>February 14, 2017</td>
<td>JK</td>
</tr>
<tr>
<td>A2</td>
<td>Production. Fixed mikroBUS Click reset pins (made GPIO).</td>
<td>September 22, 2017</td>
<td>JK</td>
</tr>
</tbody>
</table>

**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.


Known issues in rev A2:

<table>
<thead>
<tr>
<th>Issue</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO44 is incorrectly labelled as GPIO48</td>
<td><a href="https://github.com/beagleboard/pocketbeagle/issues/4">github.com/beagleboard/pocketbeagle/issues/4</a></td>
</tr>
</tbody>
</table>
2.6.3 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.

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.

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.

Tethered to a PC using Debian Images

In this configuration, you will need the following additional items:

- microUSB to USB Type A Cable
- microSD card (>=4GB and <128GB)
Fig. 2.133: PocketBeagle Package Insert front
Fig. 2.134: PocketBeagle Package Insert back
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 beagleboard.org/latest-images. 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.

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. 2.135: Tethered Configuration
Fig. 2.136: Getting Started Page

Fig. 2.137: Download Latest Software Image

Fig. 2.138: Download Etcher SD Card Utility

Fig. 2.139: Select the PocketBeagle Image
Fig. 2.140: Burn the Image to the SD Card

Fig. 2.141: Insert the microSD Card into PocketBeagle
Fig. 2.142: Insert the micro USB Connector into PocketBeagle
Fig. 2.143: Insert the USB connector into PC

Fig. 2.144: Board Power LED

Fig. 2.145: User LEDs
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, Cloud9 IDE, (Figure 17) so that you can teach your beagle new tricks!

1. Get Started Coding with Cloud9 IDE - blinking USR3 LED in JavaScript using the BoneScript library example

   1. Create a new file
Copy and paste the below code into the editor

```javascript
var b = require('bonescript');
var state = b.LOW;
b.pinMode("USR3", b.OUTPUT);
setInterval(toggle, 250); // toggle 4 times a second, every 250ms
function toggle() {
    if(state == b.LOW) state = b.HIGH;
    else state = b.LOW;
    b.digitalWrite("USR3", state);
}
```
Save the new text file as `blinkusr3.js` within the default directory

Execute .. code-block:

```bash
node blinkusr3.js
```

within the default (`/var/lib/cloud9`) directory

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 shutdown -h now”. 
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.

1. 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.

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 [github.com/beagleboard/pocketbeagle/wiki](http://github.com/beagleboard/pocketbeagle/wiki) See also the on-line discussion.

2.6.4 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.

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 2.42: PocketBeagle Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System-In-Package</td>
<td>Octavo Systems OSD335x-SM in 256 Ball BGA (21mm x 21mm)</td>
</tr>
<tr>
<td>SiP Incorporates</td>
<td>Texas Instruments 1GHz Sitara™ AM3358 ARM® Cortex®-A8 with NEON floating-point accelerator</td>
</tr>
<tr>
<td>Processor</td>
<td>Imagination Technologies PowerVR SGX530 Graphics Accelerator</td>
</tr>
<tr>
<td>Real-Time Units</td>
<td>2x programmable real-time unit (PRU) 32-bit 200MHz microcontrollers with single-cycle I/O latency</td>
</tr>
<tr>
<td>Coprocessor</td>
<td>ARM® Cortex®-M3 for power management functions</td>
</tr>
<tr>
<td>SDRAM Memory</td>
<td>512MB DDR3 800MHz RAM</td>
</tr>
<tr>
<td>Non-Volatile Memory</td>
<td>4KB I2C EEPROM for board configuration information</td>
</tr>
<tr>
<td>Power Management</td>
<td>TPS65217C PMIC along with TL5209 LDO to provide power to the system with integrated 1-cell LiPo battery support</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Bootable microSD card slot</td>
</tr>
<tr>
<td>SD/MMC</td>
<td>High speed USB 2.0 OTG (host/client) micro-B connector</td>
</tr>
<tr>
<td>Debug Support</td>
<td>JTAG test points and gdb/other monitor-mode debug possible</td>
</tr>
<tr>
<td>Power Source</td>
<td>microUSB connector, also expansion header options (battery, VIN or USB-VIN)</td>
</tr>
<tr>
<td>User I/O</td>
<td>Power Button with press detection interrupt via TPS65217C PMIC</td>
</tr>
<tr>
<td>Expansion Header</td>
<td>High speed USB 2.0 OTG (host/client) control signals</td>
</tr>
<tr>
<td>Analog Inputs</td>
<td>8 analog inputs with 6 @ 1.8V and 2 @ 3.3V along with 1.8V references</td>
</tr>
<tr>
<td>Digital I/O</td>
<td>44 digital GPIOs accessible with 18 enabled by default including 2 shared with the 3.3V analog input pins</td>
</tr>
<tr>
<td>UART</td>
<td>3 UARTs accessible with 2 enabled by default</td>
</tr>
<tr>
<td>I2C</td>
<td>2 I2C busses enabled by default</td>
</tr>
<tr>
<td>SPI</td>
<td>2 SPI busses with single chip selects enabled by default</td>
</tr>
<tr>
<td>PWM</td>
<td>4 Pulse Width Modulation outputs accessible with 2 enabled by default</td>
</tr>
<tr>
<td>QEP</td>
<td>2 Quadrature encoder inputs accessible</td>
</tr>
<tr>
<td>CAN</td>
<td>2 CAN bus controllers accessible</td>
</tr>
</tbody>
</table>

**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/](http://octavosystems.com/octavo_products/osd335x-sm/)

**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.

**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.

2.6.5 PocketBeagle High Level Specification

This section provides the high level specification of PocketBeagle.

Block Diagram

Figure 22 below is the high level block diagram of PocketBeagle.

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.
Fig. 2.152: OSD335x SIP Block Diagram
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 additional pin configurations available through software settings. Detailed information is available in Section 7.

![PocketBeagle Expansion Headers](image)

**Fig. 2.153: 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.

![microSD Connector](image)

**Fig. 2.154: 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)

![USB 2.0 Connector](image)

**Fig. 2.155: 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
BeagleBoard Docs, Release 0.0.20230323

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>
<thead>
<tr>
<th>Header.Pin</th>
<th>Silkscreen</th>
<th>Proc Ball</th>
<th>SiP Ball</th>
<th>Pin Name (Mode 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.22</td>
<td>GND</td>
<td></td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>P1.30</td>
<td>U0_TX</td>
<td>E16</td>
<td>B12</td>
<td>uart0_txd</td>
</tr>
<tr>
<td>P1.32</td>
<td>U0_RX</td>
<td>E15</td>
<td>A12</td>
<td>uart0_rxd</td>
</tr>
</tbody>
</table>

Table 2.43: UART Pins on Expansion Headers for Serial Boot

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.*

**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>
<thead>
<tr>
<th>Header.Pin</th>
<th>Silkscreen</th>
<th>Proc Ball</th>
<th>SiP Ball</th>
<th>Pin Name (Mode 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.01</td>
<td>VIN</td>
<td>P10, R10, T10</td>
<td>VIN</td>
<td></td>
</tr>
<tr>
<td>P1.07</td>
<td>USB1_VI</td>
<td>P9, R9, T9</td>
<td>VIN-USB</td>
<td></td>
</tr>
<tr>
<td>P2.14</td>
<td>BAT_+</td>
<td>P8, R8, T8</td>
<td>VIN-BAT</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.44: Power Inputs Available on Expansion Headers

<table>
<thead>
<tr>
<th>Header.Pin</th>
<th>Silkscreen</th>
<th>Proc Ball</th>
<th>SiP Ball</th>
<th>Pin Name (Mode 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.14</td>
<td>+3.3V</td>
<td>F6, F7, G6, G7</td>
<td>VOUT-3.3V</td>
<td></td>
</tr>
<tr>
<td>P1.24</td>
<td>VOUT</td>
<td>K6, K7, L6, L7</td>
<td>VOUT-5V</td>
<td></td>
</tr>
<tr>
<td>P2.13</td>
<td>VOUT</td>
<td>K6, K7, L6, L7</td>
<td>VOUT-5V</td>
<td></td>
</tr>
<tr>
<td>P2.23</td>
<td>+3.3V</td>
<td>F6, F7, G6, G7</td>
<td>VOUT-3.3V</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.45: Power Outputs Available on Expansion Headers
Table 2.46: Ground Pins Available on Expansion Headers

<table>
<thead>
<tr>
<th>Header.Pin</th>
<th>Silkscreen</th>
<th>Proc Ball</th>
<th>SiP Ball</th>
<th>Pin Name (Mode 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.15</td>
<td>USB1_GND</td>
<td></td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>P1.16</td>
<td>GND</td>
<td></td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>P1.22</td>
<td>GND</td>
<td></td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>P2.15</td>
<td>GND</td>
<td></td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>P2.21</td>
<td>GND</td>
<td></td>
<td>GND</td>
<td></td>
</tr>
</tbody>
</table>

**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.

**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. 2.156: JTAG Pad Connections](image)

**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](image)

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).
2.6.6 Detailed Hardware Design

The following sections contain schematic references for PocketBeagle. Full schematics in both PDF and Eagle are available on the ‘PocketBeagle Wiki’

OSD3358-SM SiP Design

Schematics for the OSD3358-SM SiP are divided into several diagrams.

Fig. 2.157: SiP A OSD3358 SiP System and Power Signals

SiP A OSD3358 SiP System and Power Signals
Fig. 2.158: SiP B OSD3358 SiP JTAG, USB & Analog Signals

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

MicroSD Connection

The Micro Secure Digital (microSD) connector design is highlighted in Figure 35.

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.

Power Button Design

The power button design is highlighted in Figure 37.
Fig. 2.159: SiP C OSD3358 SiP Peripheral Signals
Fig. 2.160: SiP D OSD3358 SiP System Boot Configuration
Fig. 2.161: SiP E OSD3358 SiP Power Signals
Fig. 2.162: microSD Connections
Fig. 2.163: USB Connection

Fig. 2.164: Power Button
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.

![User LEDs Diagram]

Table 2.47: User LED Control Signals/Pins

<table>
<thead>
<tr>
<th>LED</th>
<th>Signal Name</th>
<th>Proc Ball</th>
<th>SIP Ball</th>
</tr>
</thead>
<tbody>
<tr>
<td>USR0</td>
<td>GPIO1_21</td>
<td>V15</td>
<td>P13</td>
</tr>
<tr>
<td>USR1</td>
<td>GPIO1_22</td>
<td>U15</td>
<td>T14</td>
</tr>
<tr>
<td>USR2</td>
<td>GPIO1_23</td>
<td>T15</td>
<td>R14</td>
</tr>
<tr>
<td>USR3</td>
<td>GPIO1_24</td>
<td>V16</td>
<td>P14</td>
</tr>
</tbody>
</table>

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'.

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 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](https://beagleboard.org/pru).

Additional documentation is located on the Texas Instruments website at [processors.wiki.ti.com/index.php/PRU-ICSS](http://processors.wiki.ti.com/index.php/PRU-ICSS) and also located at [http://github.com/beagleboard/am335x_pru_package](http://github.com/beagleboard/am335x_pru_package).


**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 Block Diagram](image)

**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>
<thead>
<tr>
<th>Header.Pin</th>
<th>Silkscreen</th>
<th>Processor Ball</th>
<th>SIP Ball</th>
<th>Mode3</th>
<th>Mode4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.02</td>
<td>A6/87</td>
<td>R5</td>
<td>F2</td>
<td></td>
<td>pr1_pru0_pru_r30_9 (Output)</td>
</tr>
<tr>
<td>P1.04</td>
<td>89</td>
<td>R6</td>
<td>E1</td>
<td></td>
<td>pr1_pru0_pru_r31_9 (Input)</td>
</tr>
<tr>
<td>P1.06</td>
<td>SPI0_CSM</td>
<td>A16</td>
<td>A14</td>
<td></td>
<td>pr1_uart0_cts_n (Input)</td>
</tr>
<tr>
<td>P1.08</td>
<td>SPI0_CLK</td>
<td>A17</td>
<td>A13</td>
<td></td>
<td>pr1_uart0_rts_n (Output)</td>
</tr>
<tr>
<td>P1.10</td>
<td>SPI0_MISO</td>
<td>B17</td>
<td>B13</td>
<td></td>
<td>pr1_uart0_rts_n (Input)</td>
</tr>
<tr>
<td>P1.12</td>
<td>SPI0_MOSI</td>
<td>B16</td>
<td>B14</td>
<td></td>
<td>pr1_uart0_rxd (Input)</td>
</tr>
<tr>
<td>P1.20</td>
<td>20</td>
<td>D14</td>
<td>B4</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.26</td>
<td>I2C2_SDA</td>
<td>D18</td>
<td>B10</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.28</td>
<td>I2C2_SCL</td>
<td>D17</td>
<td>A10</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.29</td>
<td>PRU0_7</td>
<td>A14</td>
<td>C4</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.30</td>
<td>U0_TX</td>
<td>E16</td>
<td>B12</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.31</td>
<td>PRU0_4</td>
<td>B12</td>
<td>A3</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.32</td>
<td>U0_RX</td>
<td>E15</td>
<td>A12</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.33</td>
<td>PRU0_1</td>
<td>B13</td>
<td>A2</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.35</td>
<td>P1.10</td>
<td>V5</td>
<td>F1</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P1.36</td>
<td>PWM0A</td>
<td>A13</td>
<td>A1</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.09</td>
<td>I2C1_SCL</td>
<td>D15</td>
<td>B11</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.11</td>
<td>I2C1_SDA</td>
<td>D16</td>
<td>A11</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.17</td>
<td>65</td>
<td>V12</td>
<td>T7</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.18</td>
<td>47</td>
<td>U13</td>
<td>P7</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.20</td>
<td>64</td>
<td>T13</td>
<td>R7</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.22</td>
<td>46</td>
<td>V13</td>
<td>T6</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.24</td>
<td>48</td>
<td>T12</td>
<td>P6</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.28</td>
<td>PRU0_6</td>
<td>D13</td>
<td>C3</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.29</td>
<td>SPI1_CLK</td>
<td>C18</td>
<td>C5</td>
<td></td>
<td>pr1_ecap0_ecap_capin_apwm_o</td>
</tr>
<tr>
<td>P2.30</td>
<td>PRU0_3</td>
<td>C12</td>
<td>B1</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.31</td>
<td>SPI1_CS</td>
<td>A15</td>
<td>A4</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.32</td>
<td>PRU0_2</td>
<td>D12</td>
<td>B2</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.33</td>
<td>45</td>
<td>R12</td>
<td>R6</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.34</td>
<td>PRU0_5</td>
<td>C13</td>
<td>B3</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
<tr>
<td>P2.35</td>
<td>A5/86</td>
<td>U5</td>
<td>F3</td>
<td></td>
<td>pr1_pru0_pru_r31_16 (Input)</td>
</tr>
</tbody>
</table>

---

2.6. PocketBeagle
2.6.7 Connectors

This section describes each of the connectors on the board.

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.

![Expansion Headers for PocketBeagle](image)

Fig. 2.167: 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.

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.

<table>
<thead>
<tr>
<th>Header</th>
<th>Silkscreen PocketBeagle wiring</th>
<th>Proc Ball</th>
<th>SiP Ball</th>
<th>Model (Name)</th>
<th>Mode1</th>
<th>Mode2</th>
<th>Mode3</th>
<th>Mode4</th>
<th>Mode5</th>
<th>Mode6</th>
<th>Mode7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.01</td>
<td>VIN</td>
<td>P1.01 (VIN)</td>
<td>P10 &amp; R10 &amp; T10</td>
<td>VIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
### Table 2.49 – continued from previous page

<table>
<thead>
<tr>
<th>Header.Pin</th>
<th>Proc Ball</th>
<th>SIP Ball</th>
<th>Model (Name)</th>
<th>Mode0</th>
<th>Mode1</th>
<th>Mode2</th>
<th>Mode3</th>
<th>Mode4</th>
<th>Mode5</th>
<th>Mode6</th>
<th>Mode7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.02</td>
<td>A6/87</td>
<td>A8 (AIN6)</td>
<td>C9</td>
<td>ain6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.02</td>
<td>A6/87</td>
<td>R5 (AIN6)</td>
<td>F2</td>
<td>lcd_hsv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.03</td>
<td>USB1</td>
<td>F15</td>
<td>M14</td>
<td>USB1_DRVVBUS</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.04</td>
<td>89</td>
<td>R6</td>
<td>E1</td>
<td>lcd_ac</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.05</td>
<td>USB1</td>
<td>T18</td>
<td>M15</td>
<td>USB1_VBUS</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.06</td>
<td>SPI0_CS</td>
<td>A16</td>
<td>A14</td>
<td>spi0_cs</td>
<td>mmcm2</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.07</td>
<td>USB1</td>
<td>P9</td>
<td>VIN_USB</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>P1.08</td>
<td>SPI0_CLN</td>
<td>A17</td>
<td>A13</td>
<td>spi0_scl</td>
<td>uart2_rxd2C2</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.09</td>
<td>USB1</td>
<td>R18</td>
<td>L16</td>
<td>USB1_DM</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.10</td>
<td>SPI0_MISO</td>
<td>B17</td>
<td>B13</td>
<td>spi0_sdo</td>
<td>uart2_rxd2C2</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.11</td>
<td>USB1</td>
<td>R17</td>
<td>L15</td>
<td>USB1_DP</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.12</td>
<td>SPI0_MOSI</td>
<td>B16</td>
<td>B14</td>
<td>spi0_sd</td>
<td>mmcm1</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.13</td>
<td>USB1</td>
<td>P17</td>
<td>L14</td>
<td>USB1_ID</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.14</td>
<td>+3.3V</td>
<td>P1.14</td>
<td>F6 &amp; F7 &amp; G6 &amp; G7</td>
<td>VOUT 3.3V</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>P1.15</td>
<td>USB1</td>
<td>GND15 (GND)</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.16</td>
<td>GND</td>
<td>P1.16 (GND)</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.17</td>
<td>AIN(1.8V)</td>
<td>A9</td>
<td>B9</td>
<td>VREPN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Header Pin</th>
<th>Silkscreen</th>
<th>Proc Ball</th>
<th>SIP Ball</th>
<th>Mode0 (Name)</th>
<th>Mode1</th>
<th>Mode2</th>
<th>Mode3</th>
<th>Mode4</th>
<th>Mode5</th>
<th>Mode6</th>
<th>Mode7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.18</td>
<td>AIN(1.8V)A6 (VREFP)</td>
<td>B9</td>
<td>B7</td>
<td>VREFP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.19</td>
<td>AIN(1.8V)09 (AIN0-1.8V)</td>
<td>B6</td>
<td>A8</td>
<td>ain0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.20</td>
<td>20</td>
<td>P1.20 (PRU0)</td>
<td>D14</td>
<td>B4</td>
<td>xdma_event_intr</td>
<td>Itc1kin</td>
<td>clk-out2</td>
<td>timer7</td>
<td>pr1_pru0_pru_r31_16</td>
<td>gpio0_20</td>
<td></td>
</tr>
<tr>
<td>P1.21</td>
<td>AIN(1.8V)121 (AIN1-1.8V)</td>
<td>C7</td>
<td>B8</td>
<td>ain1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.22</td>
<td>GND</td>
<td>P1.22 (GND)</td>
<td></td>
<td></td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.23</td>
<td>AIN(1.8V)123 (AIN2-1.8V)</td>
<td>B7</td>
<td>B6</td>
<td>ain2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.24</td>
<td>VOUT</td>
<td>P1.24 (VOUT-5V)</td>
<td>K6 &amp; K7 &amp; L6 &amp; L7</td>
<td>VOUT-5V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.25</td>
<td>AIN(1.8V)25 (AIN3-1.8V)</td>
<td>A7</td>
<td>C6</td>
<td>ain3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.26</td>
<td>I2C2_SCL</td>
<td>P1.26 (I2C2-SCL)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.27</td>
<td>AIN(1.8V)427 (AIN4-1.8V)</td>
<td>C8</td>
<td>C7</td>
<td>ain4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.28</td>
<td>I2C2_SDA</td>
<td>P1.28 (I2C2-SDA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.29</td>
<td>PRU00</td>
<td>P1.29 (PRU00)</td>
<td>A14</td>
<td>C4</td>
<td>mcasp0_aqEP0_strobe0</td>
<td>mcasp0_aqEP0_strobe0</td>
<td>mcasp0_aqEP0_strobe0</td>
<td>pr1_pru0_pru_r31_16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.30</td>
<td>U0_TX</td>
<td>P1.30 (UART0-TX)</td>
<td>E16</td>
<td>B12</td>
<td>uart0_txd</td>
<td>uart0_txd</td>
<td>uart0_txd</td>
<td>pr1_pru0_pru_r31_16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.31</td>
<td>PRU00</td>
<td>P1.31 (PRU00)</td>
<td>B12</td>
<td>A3</td>
<td>mcasp0_aqEP0_strobe0</td>
<td>mcasp0_aqEP0_strobe0</td>
<td>mcasp0_aqEP0_strobe0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.32</td>
<td>U0_RX</td>
<td>P1.32 (UART0-RX)</td>
<td>E15</td>
<td>A12</td>
<td>uart0_rxd</td>
<td>uart0_rxd</td>
<td>uart0_rxd</td>
<td>pr1_pru0_pru_r31_16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.33</td>
<td>PRU00</td>
<td>P1.33 (PRU00)</td>
<td>B13</td>
<td>A2</td>
<td>mcasp0_escpwm0B</td>
<td>mcasp0_escpwm0B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.34</td>
<td>26</td>
<td>P1.34 (GPIO0)</td>
<td>T11</td>
<td>R5</td>
<td>gpmc_adc1_data2</td>
<td>gpmc_adc1_data2</td>
<td>gpmc_adc1_data2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 2.49 – continued from previous page

<table>
<thead>
<tr>
<th>Header</th>
<th>Silkscreen</th>
<th>Pocket-Beagle wiring</th>
<th>Proc Ball</th>
<th>SIP Ball</th>
<th>Mode0</th>
<th>Mode1</th>
<th>Mode2</th>
<th>Mode3</th>
<th>Mode4</th>
<th>Mode5</th>
<th>Mode6</th>
<th>Mode7</th>
<th>Mode8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.35</td>
<td>P1.10</td>
<td>V5</td>
<td>F1</td>
<td></td>
<td>lcd_pclk</td>
<td>gpmc_al</td>
<td>gpmc_al</td>
<td>mii0_m0</td>
<td>pr1_prpr</td>
<td>pr1_prpr</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td>sp1_sp</td>
</tr>
<tr>
<td>P1.36</td>
<td>PWM0</td>
<td>A13</td>
<td>A1</td>
<td></td>
<td>mcap0_pwm0</td>
<td></td>
<td>gpmc_al</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sp1_sp</td>
<td>sp1_sp</td>
</tr>
</tbody>
</table>

P2 Header

Figure 44 shows the schematic diagram for the 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 2.50: P2 Header Pinout

<table>
<thead>
<tr>
<th>Header</th>
<th>Silkscreen</th>
<th>Pocket-Beagle wiring</th>
<th>Proc Ball</th>
<th>SIP Ball</th>
<th>Mode0</th>
<th>Mode1</th>
<th>Mode2</th>
<th>Mode3</th>
<th>Mode4</th>
<th>Mode5</th>
<th>Mode6</th>
<th>Mode7</th>
<th>Mode8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2.01</td>
<td>PWM0</td>
<td>AP2.01 (PWM)</td>
<td>U14</td>
<td></td>
<td>P12</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.02</td>
<td>59</td>
<td>P2.02 (GPIO)</td>
<td>V17</td>
<td>T16</td>
<td>T16</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.03</td>
<td>23</td>
<td>P2.03 (GPIO)</td>
<td>T10</td>
<td>P5</td>
<td>P5</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.04</td>
<td>58</td>
<td>P2.04 (GPIO)</td>
<td>T16</td>
<td>R15</td>
<td>R15</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.05</td>
<td>U1_RX</td>
<td>P2.05 (UART-RX)</td>
<td>T17</td>
<td>P15</td>
<td>P15</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.06</td>
<td>57</td>
<td>P2.06 (GPIO)</td>
<td>U16</td>
<td>T15</td>
<td>T15</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.07</td>
<td>U1_TX</td>
<td>P2.07 (UART-TX)</td>
<td>U17</td>
<td>R16</td>
<td>R16</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.08</td>
<td>60</td>
<td>P2.08 (GPIO)</td>
<td>U18</td>
<td>N14</td>
<td>N14</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.09</td>
<td>I2C1_SCL</td>
<td>P2.09 (I2C1_SCL)</td>
<td>D15</td>
<td>B11</td>
<td>B11</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.10</td>
<td>52</td>
<td>P2.10 (GPIO)</td>
<td>R14</td>
<td>R13</td>
<td>R13</td>
<td>gpmc_a2</td>
<td>mii0_m1</td>
<td>pr1_prpr</td>
<td>sp1_sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Header.Pin</th>
<th>Header.Silkscreen</th>
<th>Proc Ball</th>
<th>SIP Ball</th>
<th>Model (Name)</th>
<th>Mode0 (Name)</th>
<th>Mode1 (Name)</th>
<th>Mode2 (Name)</th>
<th>Mode3 (Name)</th>
<th>Mode4 (Name)</th>
<th>Mode5 (Name)</th>
<th>Mode6 (Name)</th>
<th>Mode7 (Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2.11</td>
<td>I2C1_SDA11</td>
<td>D16</td>
<td>A11</td>
<td>uart1 mmc1</td>
<td>sd1 mmc1</td>
<td>od2C1_SDA</td>
<td>pr1 uart1</td>
<td>pr1 mmc1</td>
<td>pr1 mmc1</td>
<td>pr1 mmc1</td>
<td>pr1 mmc1</td>
<td>pr1 mmc1</td>
</tr>
<tr>
<td>P2.12</td>
<td>PB</td>
<td>P2.12</td>
<td>T11</td>
<td>POWER_BTN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.13</td>
<td>VOUT</td>
<td>P2.13</td>
<td>K6, K7, L6, L7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.14</td>
<td>BAT +</td>
<td>P2.14</td>
<td>P8, R8, T8</td>
<td>VIN-BAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.15</td>
<td>GND</td>
<td>P2.15</td>
<td>N6</td>
<td>BAT-TEMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.16</td>
<td>BAT -</td>
<td>P2.16</td>
<td>N6</td>
<td>BAT-TEMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.17</td>
<td>65</td>
<td>P2.17</td>
<td>T7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.18</td>
<td>47</td>
<td>P2.18</td>
<td>P7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.19</td>
<td>27</td>
<td>P2.19</td>
<td>T5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.20</td>
<td>64</td>
<td>P2.20</td>
<td>R7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.21</td>
<td>GND</td>
<td>P2.21</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.22</td>
<td>46</td>
<td>P2.22</td>
<td>T6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.23</td>
<td>+3.3V</td>
<td>P2.23</td>
<td>F6 &amp; F7 &amp; G6 &amp; G7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.24</td>
<td>48</td>
<td>P2.24</td>
<td>P6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.25</td>
<td>SPI1_MOSI15</td>
<td>E17</td>
<td>C13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.26</td>
<td>RST</td>
<td>P2.26</td>
<td>A10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.27</td>
<td>SPI1_MISO27</td>
<td>E18</td>
<td>C12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
### 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 1x8 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).

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’.

### Table 2.50 – continued from previous page

<table>
<thead>
<tr>
<th>Header.Pin</th>
<th>Silkscreen</th>
<th>Proc Ball</th>
<th>SLP Ball</th>
<th>Mode0 (Name)</th>
<th>Mode1</th>
<th>Mode2</th>
<th>Mode3</th>
<th>Mode4</th>
<th>Mode5</th>
<th>Mode6</th>
<th>Mode7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2.28</td>
<td>PRU0_6</td>
<td>D13(6)</td>
<td>C3</td>
<td>mcasp0</td>
<td>_QEP0_1_index</td>
<td>mcasp1</td>
<td>_EMU3</td>
<td>pr1_pr0_pru0_pru_r30_61</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.29</td>
<td>SPI1_CLK</td>
<td>C18</td>
<td>C5</td>
<td>eCAP0</td>
<td>_PWM_Bits</td>
<td>pr1_ecap1_cap</td>
<td>_spi0_cap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.30</td>
<td>PRU0_3</td>
<td>C12(3)</td>
<td>B1</td>
<td>mcasp1</td>
<td>_ahrpwm0_tripzone</td>
<td>mcasp2</td>
<td>_spi0_cap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.31</td>
<td>SPI1_CLK</td>
<td>A15</td>
<td>A4</td>
<td>xdma_event_int</td>
<td>timer4</td>
<td>clk-out1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.32</td>
<td>PRU0_2</td>
<td>D12(2)</td>
<td>B2</td>
<td>mcasp0</td>
<td>_ahrpwm0_tripzone</td>
<td>mcasp1</td>
<td>_mm2_cap</td>
<td>_spio_viol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.33</td>
<td>45</td>
<td>R12(13)</td>
<td>R6</td>
<td>gpmc_ad12</td>
<td>ad11nc1</td>
<td>dma_in2</td>
<td>daQEP2</td>
<td>_spi4_mii0_txd1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.34</td>
<td>PRU0_1</td>
<td>C13(5)</td>
<td>B3</td>
<td>mcasp0</td>
<td>_QEP0</td>
<td>mm2sp1</td>
<td>_EMU2</td>
<td>pr1_pr0_pru0_pru_r30_51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.35</td>
<td>A5/86</td>
<td>B8</td>
<td>C8</td>
<td>gpmc_A5_vers</td>
<td>gpmc_A5</td>
<td>pr1_gpio86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.36</td>
<td>A7/18</td>
<td>A7</td>
<td>N13</td>
<td>gpmc_A7_vers</td>
<td>gpmc_A7</td>
<td>pr1_gpio7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**2.6. PocketBeagle**

287
Fig. 2.168: Expansion Header Popular Functions - Color Coded
### 2.6. PocketBeagle

**Fig. 2.169: P2 Header**

**Fig. 2.170: mikroBUS**

**Fig. 2.171: PocketBeagle Both Headers**
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.

2.6.8 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.

2.6.9 PocketBeagle Mechanical

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 github.com/beagleboard/pocketbeagle/tree/master/models

2.6.10 Additional Pictures

2.6.11 Support Information

All support for this design is through the BeagleBoard.org community at:

- beagleboard@googlegroups.com or
Fig. 2.172: PocketBeagle Front BW

Fig. 2.173: PocketBeagle Back BW
Hardware Design

Design documentation can be found on the wiki. https://github.com/beagleboard/pocketbeagle Including:

- Schematic in PDF https://github.com/beagleboard/pocketbeagle/blob/master/PocketBeagle_sch.pdf
- Schematic and layout in EAGLE https://github.com/beagleboard/pocketbeagle/tree/master/EAGLE
- Schematic and layout in KiCAD https://github.com/beagleboard/pocketbeagle/tree/master/KiCAD
- Bill of Materials https://github.com/beagleboard/pocketbeagle/blob/master/PocketBeagle_BOM.csv

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 beagleboard.org/latest-images

Export Information

- ECCN: EAR99
- CCATS: G173833
- Documentation: github.com/beagleboard/pocketbeagle/blob/master/regulatory/PocketBeagle_Export_Classification.pdf

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.

Getting Help

If you need some up to date troubleshooting techniques, the Wiki is a great place to start github.com/beagleboard/pocketbeagle/wiki.

If you need professional support, check out beagleboard.org/resources.

2.7 Capes

Note: This page is under development.

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 boards-terms-and-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

### 2.7.1 BeagleBone cape interface spec

This page is a fork of BeagleBone cape interface spec page on elinux. This is the new official home.

**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.
**Note:** Legend

- **D**: Digital general purpose input and output (GPIO)
- **I**: Inter-integrated circuit bus (I2C) ports
- **S**: Serial peripheral interface (SPI) ports
- **U**: Universal asynchronous receiver/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 2.51: Overall

<table>
<thead>
<tr>
<th>P9</th>
<th>Func-</th>
<th>odd</th>
<th>even</th>
<th>Func-</th>
<th>odd</th>
<th>even</th>
<th>Func-</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB D+</td>
<td>E1</td>
<td>E2</td>
<td>USB D-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5V OUT</td>
<td>E3</td>
<td>E4</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3V3 OUT</td>
<td>3</td>
<td>4</td>
<td>3V3 OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5V IN</td>
<td>5</td>
<td>6</td>
<td>5V IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5V OUT</td>
<td>7</td>
<td>8</td>
<td>5V OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWR BUT</td>
<td>9</td>
<td>10</td>
<td>RESET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P8</th>
<th>Func-</th>
<th>odd</th>
<th>even</th>
<th>Func-</th>
<th>odd</th>
<th>even</th>
<th>Func-</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBD+</td>
<td>E1</td>
<td>E2</td>
<td>USBD-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5VOUT</td>
<td>E3</td>
<td>E4</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3V3 OUT</td>
<td>D M</td>
<td>3</td>
<td>4</td>
<td>D M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5V IN</td>
<td>D M C</td>
<td>5</td>
<td>6</td>
<td>D M C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5V OUT</td>
<td>D C</td>
<td>7</td>
<td>8</td>
<td>D C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWR BUT</td>
<td>D C</td>
<td>9</td>
<td>10</td>
<td>D C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Digital GPIO

The compatibility layer comes with simple reference nodes for attaching the Linuux 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 2.1: Example device tree overlay to enable LED driver on header P8 pin 3

```bash
/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

2.7. Capes
other properties defined in the \texttt{gpio-leds} schema.

Table 2.52: GPIO pins

<table>
<thead>
<tr>
<th>P9</th>
<th></th>
<th></th>
<th>P8</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>odd</td>
<td>even</td>
<td>Functions</td>
<td>odd</td>
<td>even</td>
</tr>
<tr>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3V3 OUT</td>
<td>3</td>
<td>4</td>
<td>3V3 OUT</td>
<td>D M</td>
<td>3</td>
</tr>
<tr>
<td>5V IN</td>
<td>5</td>
<td>6</td>
<td>5V IN</td>
<td>D M C4t</td>
<td>5</td>
</tr>
<tr>
<td>5V OUT</td>
<td>7</td>
<td>8</td>
<td>5V OUT</td>
<td>D C2r</td>
<td>7</td>
</tr>
<tr>
<td>PWR BUT</td>
<td>9</td>
<td>10</td>
<td>RESET</td>
<td>D C3r</td>
<td>9</td>
</tr>
<tr>
<td>D U4r</td>
<td>11</td>
<td>12</td>
<td>D</td>
<td>D P0 o</td>
<td>11</td>
</tr>
<tr>
<td>D U4t</td>
<td>13</td>
<td>14</td>
<td>D E1a</td>
<td>D E2 b</td>
<td>13</td>
</tr>
<tr>
<td>D</td>
<td>15</td>
<td>16</td>
<td>D E1b</td>
<td>D P0 i</td>
<td>15</td>
</tr>
<tr>
<td>D I1c S00</td>
<td>17</td>
<td>18</td>
<td>D I1d S0o</td>
<td>D</td>
<td>17</td>
</tr>
<tr>
<td>C0r D I2c</td>
<td>19</td>
<td>20</td>
<td>C0t D I2d</td>
<td>D E2a</td>
<td>19</td>
</tr>
<tr>
<td>D E0b S0i U2t</td>
<td>21</td>
<td>22</td>
<td>D E0a S0c U2r</td>
<td>D M P1</td>
<td>21</td>
</tr>
<tr>
<td>D S01</td>
<td>23</td>
<td>24</td>
<td>C1t D I3c U1t</td>
<td>D M</td>
<td>23</td>
</tr>
<tr>
<td>D P0</td>
<td>25</td>
<td>26</td>
<td>C1t D I3d U1r</td>
<td>D M</td>
<td>25</td>
</tr>
<tr>
<td>D P0 Q0b</td>
<td>27</td>
<td>28</td>
<td>D P0 S10</td>
<td>D L P1</td>
<td>27</td>
</tr>
<tr>
<td>D E S1i P0</td>
<td>29</td>
<td>30</td>
<td>D P0 S1o</td>
<td>D L P1 U6t</td>
<td>29</td>
</tr>
<tr>
<td>D E S1c P0</td>
<td>31</td>
<td>32</td>
<td>ADC VDD</td>
<td>D L</td>
<td>31</td>
</tr>
<tr>
<td>A4</td>
<td>33</td>
<td>34</td>
<td>ADC GND</td>
<td>D L Q1b</td>
<td>33</td>
</tr>
<tr>
<td>A6</td>
<td>35</td>
<td>36</td>
<td>A5</td>
<td>D L Q1a</td>
<td>35</td>
</tr>
<tr>
<td>A2</td>
<td>37</td>
<td>38</td>
<td>A3</td>
<td>D L U5t</td>
<td>37</td>
</tr>
<tr>
<td>A0</td>
<td>39</td>
<td>40</td>
<td>A1</td>
<td>D L P1</td>
<td>39</td>
</tr>
<tr>
<td>D P0</td>
<td>41</td>
<td>42</td>
<td>D Q0a S11 U3t P0</td>
<td>D L P1</td>
<td>41</td>
</tr>
<tr>
<td>GND</td>
<td>43</td>
<td>44</td>
<td>GND</td>
<td>D L P1</td>
<td>43</td>
</tr>
<tr>
<td>GND</td>
<td>45</td>
<td>46</td>
<td>GND</td>
<td>D E L P1</td>
<td>45</td>
</tr>
</tbody>
</table>

Table 2.53: Bone GPIO LEDs interface

<table>
<thead>
<tr>
<th>LED SYSFS</th>
<th>Header pin</th>
<th>Black</th>
<th>AI</th>
<th>Al-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sys/class/leds/P8_03</td>
<td>P8_03</td>
<td>gpio1_6</td>
<td>gpio1_24</td>
<td>gpio0_20</td>
</tr>
<tr>
<td>/sys/class/leds/P8_04</td>
<td>P8_04</td>
<td>gpio1_7</td>
<td>gpio1_25</td>
<td>gpio0_48</td>
</tr>
<tr>
<td>/sys/class/leds/P8_05</td>
<td>P8_05</td>
<td>gpio1_2</td>
<td>gpio7_1</td>
<td>gpio0_33</td>
</tr>
<tr>
<td>/sys/class/leds/P8_06</td>
<td>P8_06</td>
<td>gpio1_3</td>
<td>gpio7_2</td>
<td>gpio0_34</td>
</tr>
<tr>
<td>/sys/class/leds/P8_07</td>
<td>P8_07</td>
<td>gpio2_2</td>
<td>gpio6_5</td>
<td>gpio0_15</td>
</tr>
<tr>
<td>/sys/class/leds/P8_08</td>
<td>P8_08</td>
<td>gpio2_3</td>
<td>gpio6_6</td>
<td>gpio0_14</td>
</tr>
<tr>
<td>/sys/class/leds/P8_09</td>
<td>P8_09</td>
<td>gpio2_5</td>
<td>gpio6_18</td>
<td>gpio0_17</td>
</tr>
<tr>
<td>/sys/class/leds/P8_10</td>
<td>P8_10</td>
<td>gpio2_4</td>
<td>gpio6_4</td>
<td>gpio0_16</td>
</tr>
</tbody>
</table>
### Table 2.53 – continued from previous page

<table>
<thead>
<tr>
<th>LED SYSFS</th>
<th>Header pin</th>
<th>Black</th>
<th>AI</th>
<th>AI-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sys/class/leds/P8_11</td>
<td>P8_11</td>
<td>gpio1_13</td>
<td>gpio3_11</td>
<td>gpio0_60</td>
</tr>
<tr>
<td>/sys/class/leds/P8_12</td>
<td>P8_12</td>
<td>gpio1_12</td>
<td>gpio3_10</td>
<td>gpio0_59</td>
</tr>
<tr>
<td>/sys/class/leds/P8_13</td>
<td>P8_13</td>
<td>gpio0_23</td>
<td>gpio4_11</td>
<td>gpio0_89</td>
</tr>
<tr>
<td>/sys/class/leds/P8_14</td>
<td>P8_14</td>
<td>gpio0_26</td>
<td>gpio4_13</td>
<td>gpio0_75</td>
</tr>
<tr>
<td>/sys/class/leds/P8_15</td>
<td>P8_15</td>
<td>gpio1_15</td>
<td>gpio4_3</td>
<td>gpio0_61</td>
</tr>
<tr>
<td>/sys/class/leds/P8_16</td>
<td>P8_16</td>
<td>gpio1_14</td>
<td>gpio4_29</td>
<td>gpio0_62</td>
</tr>
<tr>
<td>/sys/class/leds/P8_17</td>
<td>P8_17</td>
<td>gpio0_27</td>
<td>gpio8_18</td>
<td>gpio0_3</td>
</tr>
<tr>
<td>/sys/class/leds/P8_18</td>
<td>P8_18</td>
<td>gpio2_1</td>
<td>gpio4_9</td>
<td>gpio0_4</td>
</tr>
<tr>
<td>/sys/class/leds/P8_19</td>
<td>P8_19</td>
<td>gpio0_22</td>
<td>gpio4_10</td>
<td>gpio0_88</td>
</tr>
<tr>
<td>/sys/class/leds/P8_20</td>
<td>P8_20</td>
<td>gpio1_31</td>
<td>gpio6_30</td>
<td>gpio0_76</td>
</tr>
<tr>
<td>/sys/class/leds/P8_21</td>
<td>P8_21</td>
<td>gpio1_30</td>
<td>gpio6_29</td>
<td>gpio0_30</td>
</tr>
<tr>
<td>/sys/class/leds/P8_22</td>
<td>P8_22</td>
<td>gpio1_5</td>
<td>gpio1_23</td>
<td>gpio0_5</td>
</tr>
<tr>
<td>/sys/class/leds/P8_23</td>
<td>P8_23</td>
<td>gpio1_4</td>
<td>gpio1_22</td>
<td>gpio0_31</td>
</tr>
<tr>
<td>/sys/class/leds/P8_24</td>
<td>P8_24</td>
<td>gpio1_1</td>
<td>gpio7_0</td>
<td>gpio0_6</td>
</tr>
<tr>
<td>/sys/class/leds/P8_25</td>
<td>P8_25</td>
<td>gpio1_0</td>
<td>gpio6_31</td>
<td>gpio0_35</td>
</tr>
<tr>
<td>/sys/class/leds/P8_26</td>
<td>P8_26</td>
<td>gpio2_22</td>
<td>gpio4_23</td>
<td>gpio0_71</td>
</tr>
<tr>
<td>/sys/class/leds/P8_27</td>
<td>P8_27</td>
<td>gpio2_24</td>
<td>gpio4_19</td>
<td>gpio0_72</td>
</tr>
<tr>
<td>/sys/class/leds/P8_28</td>
<td>P8_28</td>
<td>gpio2_23</td>
<td>gpio4_22</td>
<td>gpio0_73</td>
</tr>
<tr>
<td>/sys/class/leds/P8_29</td>
<td>P8_29</td>
<td>gpio2_25</td>
<td>gpio4_20</td>
<td>gpio0_74</td>
</tr>
<tr>
<td>/sys/class/leds/P8_30</td>
<td>P8_30</td>
<td>gpio0_10</td>
<td>gpio8_14</td>
<td>gpio0_32</td>
</tr>
<tr>
<td>/sys/class/leds/P8_31</td>
<td>P8_31</td>
<td>gpio0_11</td>
<td>gpio8_15</td>
<td>gpio0_26</td>
</tr>
<tr>
<td>/sys/class/leds/P8_32</td>
<td>P8_32</td>
<td>gpio0_9</td>
<td>gpio8_13</td>
<td>gpio0_25</td>
</tr>
<tr>
<td>/sys/class/leds/P8_33</td>
<td>P8_33</td>
<td>gpio2_17</td>
<td>gpio8_11</td>
<td>gpio0_7</td>
</tr>
<tr>
<td>/sys/class/leds/P8_34</td>
<td>P8_34</td>
<td>gpio0_8</td>
<td>gpio8_12</td>
<td>gpio0_24</td>
</tr>
<tr>
<td>/sys/class/leds/P8_35</td>
<td>P8_35</td>
<td>gpio2_16</td>
<td>gpio8_10</td>
<td>gpio0_8</td>
</tr>
<tr>
<td>/sys/class/leds/P8_36</td>
<td>P8_36</td>
<td>gpio2_14</td>
<td>gpio8_8</td>
<td>gpio0_106</td>
</tr>
<tr>
<td>/sys/class/leds/P8_37</td>
<td>P8_37</td>
<td>gpio2_15</td>
<td>gpio8_9</td>
<td>gpio0_105</td>
</tr>
<tr>
<td>/sys/class/leds/P8_38</td>
<td>P8_38</td>
<td>gpio2_12</td>
<td>gpio8_6</td>
<td>gpio0_69</td>
</tr>
<tr>
<td>/sys/class/leds/P8_39</td>
<td>P8_39</td>
<td>gpio2_13</td>
<td>gpio8_7</td>
<td>gpio0_70</td>
</tr>
<tr>
<td>/sys/class/leds/P8_40</td>
<td>P8_40</td>
<td>gpio2_10</td>
<td>gpio8_4</td>
<td>gpio0_67</td>
</tr>
<tr>
<td>/sys/class/leds/P8_41</td>
<td>P8_41</td>
<td>gpio2_11</td>
<td>gpio8_5</td>
<td>gpio0_68</td>
</tr>
<tr>
<td>/sys/class/leds/P8_42</td>
<td>P8_42</td>
<td>gpio2_8</td>
<td>gpio8_2</td>
<td>gpio0_65</td>
</tr>
<tr>
<td>/sys/class/leds/P8_43</td>
<td>P8_43</td>
<td>gpio2_9</td>
<td>gpio8_3</td>
<td>gpio0_66</td>
</tr>
<tr>
<td>/sys/class/leds/P8_44</td>
<td>P8_44</td>
<td>gpio2_6</td>
<td>gpio8_0</td>
<td>gpio0_79</td>
</tr>
<tr>
<td>/sys/class/leds/P8_45</td>
<td>P8_45</td>
<td>gpio2_7</td>
<td>gpio8_1</td>
<td>gpio0_80</td>
</tr>
<tr>
<td>/sys/class/leds/P8_46</td>
<td>P8_46</td>
<td>gpio0_30</td>
<td>gpio8_17</td>
<td>gpio0_1</td>
</tr>
<tr>
<td>/sys/class/leds/P9_11</td>
<td>P9_11</td>
<td>gpio0_28</td>
<td>gpio5_0</td>
<td>gpio0_45</td>
</tr>
<tr>
<td>/sys/class/leds/P9_12</td>
<td>P9_12</td>
<td>gpio0_31</td>
<td>gpio6_12</td>
<td>gpio0_2</td>
</tr>
<tr>
<td>/sys/class/leds/P9_13</td>
<td>P9_13</td>
<td>gpio0_18</td>
<td>gpio4_25</td>
<td>gpio0_93</td>
</tr>
<tr>
<td>/sys/class/leds/P9_14</td>
<td>P9_14</td>
<td>gpio1_16</td>
<td>gpio3_12</td>
<td>gpio0_47</td>
</tr>
<tr>
<td>/sys/class/leds/P9_15</td>
<td>P9_15</td>
<td>gpio1_19</td>
<td>gpio4_26</td>
<td>gpio0_94</td>
</tr>
<tr>
<td>/sys/class/leds/P9_16</td>
<td>P9_16</td>
<td>gpio0_5</td>
<td>gpio7_17</td>
<td>gpio0_28</td>
</tr>
<tr>
<td>/sys/class/leds/P9_17</td>
<td>P9_17</td>
<td>gpio0_4</td>
<td>gpio7_16</td>
<td>gpio0_40</td>
</tr>
<tr>
<td>/sys/class/leds/P9_18</td>
<td>P9_18</td>
<td>gpio0_13</td>
<td>gpio7_3</td>
<td>gpio0_78</td>
</tr>
<tr>
<td>/sys/class/leds/P9_19</td>
<td>P9_19</td>
<td>gpio0_12</td>
<td>gpio7_4</td>
<td>gpio0_77</td>
</tr>
<tr>
<td>/sys/class/leds/P9_20</td>
<td>P9_20</td>
<td>gpio0_3</td>
<td>gpio3_3</td>
<td>gpio0_39</td>
</tr>
<tr>
<td>/sys/class/leds/P9_21</td>
<td>P9_21</td>
<td>gpio0_2</td>
<td>gpio6_19</td>
<td>gpio0_38</td>
</tr>
<tr>
<td>/sys/class/leds/P9_22</td>
<td>P9_22</td>
<td>gpio1_17</td>
<td>gpio7_11</td>
<td>gpio0_10</td>
</tr>
<tr>
<td>/sys/class/leds/P9_23</td>
<td>P9_23</td>
<td>gpio0_15</td>
<td>gpio6_15</td>
<td>gpio0_13</td>
</tr>
<tr>
<td>/sys/class/leds/P9_24</td>
<td>P9_24</td>
<td>gpio3_21</td>
<td>gpio6_17</td>
<td>gpio0_127</td>
</tr>
<tr>
<td>/sys/class/leds/P9_25</td>
<td>P9_25</td>
<td>gpio0_14</td>
<td>gpio6_14</td>
<td>gpio0_12</td>
</tr>
<tr>
<td>/sys/class/leds/P9_26</td>
<td>P9_26</td>
<td>gpio3_19</td>
<td>gpio4_15</td>
<td>gpio0_46</td>
</tr>
</tbody>
</table>

continues on next page
Table 2.53 – continued from previous page

<table>
<thead>
<tr>
<th>LED SYSFS</th>
<th>Header pin</th>
<th>Black</th>
<th>AI</th>
<th>AI-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sys/class/leds/P9_28</td>
<td>P9_28</td>
<td>gpio3_17</td>
<td>gpio4_17</td>
<td>gpio1_11</td>
</tr>
<tr>
<td>/sys/class/leds/P9_29</td>
<td>P9_29</td>
<td>gpio3_15</td>
<td>gpio5_11</td>
<td>gpio0_53</td>
</tr>
<tr>
<td>/sys/class/leds/P9_30</td>
<td>P9_30</td>
<td>gpio3_16</td>
<td>gpio5_12</td>
<td>gpio0_44</td>
</tr>
<tr>
<td>/sys/class/leds/P9_31</td>
<td>P9_31</td>
<td>gpio3_14</td>
<td>gpio5_10</td>
<td>gpio0_52</td>
</tr>
<tr>
<td>/sys/class/leds/P9_33</td>
<td>P9_33</td>
<td>n/a</td>
<td>n/a</td>
<td>gpio0_50</td>
</tr>
<tr>
<td>/sys/class/leds/P9_35</td>
<td>P9_35</td>
<td>n/a</td>
<td>n/a</td>
<td>gpio0_55</td>
</tr>
<tr>
<td>/sys/class/leds/P9_36</td>
<td>P9_36</td>
<td>n/a</td>
<td>n/a</td>
<td>gpio0_56</td>
</tr>
<tr>
<td>/sys/class/leds/P9_37</td>
<td>P9_37</td>
<td>n/a</td>
<td>n/a</td>
<td>gpio0_57</td>
</tr>
<tr>
<td>/sys/class/leds/P9_38</td>
<td>P9_38</td>
<td>n/a</td>
<td>n/a</td>
<td>gpio0_58</td>
</tr>
<tr>
<td>/sys/class/leds/P9_39</td>
<td>P9_39</td>
<td>n/a</td>
<td>n/a</td>
<td>gpio0_54</td>
</tr>
<tr>
<td>/sys/class/leds/P9_40</td>
<td>P9_40</td>
<td>n/a</td>
<td>n/a</td>
<td>gpio0_81</td>
</tr>
<tr>
<td>/sys/class/leds/P9_41</td>
<td>P9_41</td>
<td>gpio0_20</td>
<td>gpio6_20</td>
<td>gpio1_0</td>
</tr>
<tr>
<td>/sys/class/leds/P9_42</td>
<td>P9_42</td>
<td>gpio0_7</td>
<td>gpio4_18</td>
<td>gpio0_123</td>
</tr>
<tr>
<td>/sys/class/leds/A15</td>
<td>A15</td>
<td>gpio0_19</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

I²C

Compatibility layer provides simple I²C bone bus nodes for creating compatible overlays for Black, AI and AI-64. The format followed for these nodes is **bone_i2c_#**.

Table 2.54: I²C pins

<table>
<thead>
<tr>
<th>P9 Functions</th>
<th>odd</th>
<th>even</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SCL</td>
<td>17</td>
<td>18</td>
<td>1 SDA</td>
</tr>
<tr>
<td>2 SCL</td>
<td>19</td>
<td>20</td>
<td>2 SDA</td>
</tr>
<tr>
<td>4 SCL&lt;sup&gt;25&lt;/sup&gt;</td>
<td>21</td>
<td>22</td>
<td>4 SDA&lt;sup&gt;25&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>24</td>
<td>3 SCL&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>26</td>
<td>3 SDA&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

Table 2.55: I²C port mapping

<table>
<thead>
<tr>
<th>SYSFS</th>
<th>DT symbol</th>
<th>Black</th>
<th>AI</th>
<th>AI-64</th>
<th>SCL</th>
<th>SDA</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/bone/i2c/0</td>
<td>bone_i2c_0</td>
<td>I2C0</td>
<td>I2C1</td>
<td>TBD</td>
<td>On-board</td>
<td></td>
<td>TBD</td>
</tr>
<tr>
<td>/dev/bone/i2c/1</td>
<td>bone_i2c_1</td>
<td>I2C1</td>
<td>I2C5</td>
<td>MAIN_I2C6</td>
<td>P9.17</td>
<td>P9.18</td>
<td>BONE-I2C1</td>
</tr>
<tr>
<td>/dev/bone/i2c/2</td>
<td>bone_i2c_2</td>
<td>I2C2</td>
<td>I2C4</td>
<td>MAIN_I2C3</td>
<td>P9.19</td>
<td>P9.20</td>
<td>BONE-I2C2</td>
</tr>
<tr>
<td>/dev/bone/i2c/3</td>
<td>bone_i2c_3</td>
<td>I2C1</td>
<td>I2C3</td>
<td>MAIN_I2C4</td>
<td>P9.24</td>
<td>P9.26</td>
<td>BONE-I2C3</td>
</tr>
<tr>
<td>/dev/bone/i2c/4</td>
<td>bone_i2c_4</td>
<td>I2C2</td>
<td>n/a</td>
<td>MAIN_I2C3</td>
<td>P9.21</td>
<td>P9.22</td>
<td>BONE-I2C4</td>
</tr>
</tbody>
</table>

**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 usernode device driver loading can be performed, depending on the driver. See *Using I²C with Linux drivers* for information on loading I²C drivers from userspace.

<sup>1</sup> Mutually exclusive with port 1 on Black
<sup>2</sup> On Black and AI-64 only
<sup>3</sup> Mutually exclusive with port 2 on Black
Listing 2.2: Example device tree overlay to enable I2C driver

```c
/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/Documentation/devicetree/bindings/i2c/

**SPI**

SPI bone bus nodes allow creating compatible overlays for Black, AI and AI-64.

<table>
<thead>
<tr>
<th>P9 Functions</th>
<th>odd</th>
<th>even</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CS0</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>0 SDI</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>0 CS1</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>1 SDI</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>1 CLK</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>41</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bone bus</th>
<th>DT symbol</th>
<th>Black</th>
<th>AI</th>
<th>AI-64</th>
<th>SDO</th>
<th>SDI</th>
<th>CLK</th>
<th>CS</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/bone/spi/0</td>
<td>.bone_spi_0</td>
<td>SPI0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P9.17 (CS0)</td>
<td>BONE-SPI0_0</td>
</tr>
<tr>
<td>/dev/bone/spi/0..1</td>
<td>.bone_spi_0</td>
<td>SPI2</td>
<td></td>
<td>SPI2</td>
<td></td>
<td></td>
<td></td>
<td>P9.21</td>
<td>P9.22 (CS1)</td>
</tr>
<tr>
<td>/dev/bone/spi/1</td>
<td>.bone_spi_1</td>
<td>SPI1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P9.30</td>
<td>P9.23 (CS1)</td>
</tr>
<tr>
<td>/dev/bone/spi/1..1</td>
<td>.bone_spi_1</td>
<td>SPI3</td>
<td></td>
<td>SPI3</td>
<td></td>
<td></td>
<td></td>
<td>P9.29</td>
<td>P9.27 (CS0)</td>
</tr>
</tbody>
</table>

2 Only available on AI and AI-64
**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.

**Note:** #TODO# figure out if BONE-SPI0_0 and BONE-SPI0_1 can be loaded at the same time

Listing 2.3: 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.


**UART**

UART bone bus nodes allow creating compatible overlays for Black, AI and AI-64.

<table>
<thead>
<tr>
<th>P9</th>
<th>P9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>odd</td>
</tr>
<tr>
<td>4 RX</td>
<td>11</td>
</tr>
<tr>
<td>TX</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>2 TX</td>
<td>21</td>
</tr>
<tr>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
</tr>
</tbody>
</table>

6 This port is shared with the console UART on AI-64
Important: RTSn and CTSn mappings are not compatible across boards in the family and are therefore not part of the cape specification.

Table 2.59: UART port mapping

<table>
<thead>
<tr>
<th>Bone bus</th>
<th>DT symbol</th>
<th>Black</th>
<th>AI</th>
<th>AI-64</th>
<th>TX</th>
<th>RX</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/bone/uart/0</td>
<td>bone_uart_0</td>
<td>UART0</td>
<td>UART1</td>
<td>MAIN_UART0</td>
<td></td>
<td></td>
<td>Console debug header pins</td>
</tr>
<tr>
<td>/dev/bone/uart/1</td>
<td>bone_uart_1</td>
<td>UART1</td>
<td>UART10</td>
<td>MAIN_UART2</td>
<td>P9.24</td>
<td>P9.26</td>
<td>BONE-UART1</td>
</tr>
<tr>
<td>/dev/bone/uart/2</td>
<td>bone_uart_2</td>
<td>UART2</td>
<td>UART3</td>
<td>n/a</td>
<td>P9.21</td>
<td>P9.22</td>
<td>BONE-UART2</td>
</tr>
<tr>
<td>/dev/bone/uart/3</td>
<td>bone_uart_3</td>
<td>UART3</td>
<td>n/a</td>
<td>n/a</td>
<td>P9.42</td>
<td>n/a</td>
<td>BONE-UART3</td>
</tr>
<tr>
<td>/dev/bone/uart/4</td>
<td>bone_uart_4</td>
<td>UART4</td>
<td>UART5</td>
<td>MAIN_UART0</td>
<td>P9.13</td>
<td>P9.11</td>
<td>BONE-UART4</td>
</tr>
<tr>
<td>/dev/bone/uart/5</td>
<td>bone_uart_5</td>
<td>UART5</td>
<td>UART8</td>
<td>MAIN_UART5</td>
<td>P8.37</td>
<td>P8.38</td>
<td>BONE-UART5</td>
</tr>
<tr>
<td>/dev/bone/uart/6</td>
<td>bone_uart_6</td>
<td>n/a</td>
<td>n/a</td>
<td>MAIN_UART8</td>
<td>P8.29</td>
<td>P8.28</td>
<td>BONE-UART6</td>
</tr>
<tr>
<td>/dev/bone/uart/7</td>
<td>bone_uart_7</td>
<td>n/a</td>
<td>n/a</td>
<td>MAIN_UART2</td>
<td>P8.34</td>
<td>P8.22</td>
<td>BONE-UART7</td>
</tr>
</tbody>
</table>

Important: In the case the same controller is used for 2 different bone bus nodes, usage of those nodes is mutually-exclusive.

CAN

CAN bone bus nodes allow creating compatible overlays for Black, AI and AI-64.

Table 2.60: CAN pins

<table>
<thead>
<tr>
<th>P9 Functions</th>
<th>odd</th>
<th>even</th>
<th>P8 Functions</th>
<th>odd</th>
<th>even</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td>4 TX</td>
<td>5</td>
<td>6</td>
<td>4 RX</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
<td>2 TX</td>
<td>7</td>
<td>8</td>
<td>2 RX</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td></td>
<td>3 RX</td>
<td>9</td>
<td>10</td>
<td>3 TX</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td></td>
<td></td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td></td>
<td></td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td></td>
<td></td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>0 RX</td>
<td>19</td>
<td>20</td>
<td>0 TX</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td></td>
<td></td>
<td>21</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>1 RX</td>
<td></td>
<td>23</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>1 TX</td>
<td></td>
<td>25</td>
<td>26</td>
<td></td>
</tr>
</tbody>
</table>

2.7. Capes
Table 2.61: CAN port mapping

<table>
<thead>
<tr>
<th>Bone bus</th>
<th>Black</th>
<th>AI</th>
<th>AI-64</th>
<th>TX</th>
<th>RX</th>
<th>Overlays</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/bone/can/0</td>
<td>CAN0</td>
<td>n/a</td>
<td>MAIN_MCAN0</td>
<td>P9.20</td>
<td>P9.19</td>
<td>BONE-CAN0</td>
</tr>
<tr>
<td>/dev/bone/can/1</td>
<td>CAN1</td>
<td>CAN2</td>
<td>MAIN_MCAN4</td>
<td>P9.26</td>
<td>P9.24</td>
<td>BONE-CAN1</td>
</tr>
<tr>
<td>/dev/bone/can/2</td>
<td>n/a</td>
<td>CAN1</td>
<td>MAIN_MCAN5</td>
<td>P8.08</td>
<td>P8.07</td>
<td>BONE-CAN2</td>
</tr>
<tr>
<td>/dev/bone/can/3</td>
<td>n/a</td>
<td>n/a</td>
<td>MAIN_MCAN6</td>
<td>P8.10</td>
<td>P8.09</td>
<td>BONE-CAN3</td>
</tr>
<tr>
<td>/dev/bone/can/4</td>
<td>n/a</td>
<td>n/a</td>
<td>MAIN_MCAN7</td>
<td>P8.05</td>
<td>P8.06</td>
<td>BONE-CAN4</td>
</tr>
</tbody>
</table>

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

¹ BeagleBone AI rev A2 and later only
## Table 2.62: ADC pins

<table>
<thead>
<tr>
<th>P9 Functions</th>
<th>odd</th>
<th>even</th>
<th>P8 Functions</th>
<th>odd</th>
<th>even</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB D+</td>
<td>E1</td>
<td>E2</td>
<td>USB D-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5V OUT</td>
<td>E3</td>
<td>E4</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
</tr>
<tr>
<td>3V3 OUT</td>
<td>3</td>
<td>4</td>
<td>3V3 OUT</td>
<td>D M</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5V IN</td>
<td>5</td>
<td>6</td>
<td>D M C4t</td>
<td>5</td>
<td>6</td>
<td>D M C4r</td>
</tr>
<tr>
<td>5V OUT</td>
<td>7</td>
<td>8</td>
<td>D C2r</td>
<td>7</td>
<td>8</td>
<td>D C2t</td>
</tr>
<tr>
<td>PWR BUT</td>
<td>9</td>
<td>10</td>
<td>RESET</td>
<td>D C3r</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>D U4r</td>
<td>11</td>
<td>12</td>
<td>D</td>
<td>D P0</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>D U4t</td>
<td>13</td>
<td>14</td>
<td>D E1a</td>
<td>D E2b</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>D</td>
<td>15</td>
<td>16</td>
<td>D E1b</td>
<td>D P0i</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>D 11c S00</td>
<td>17</td>
<td>18</td>
<td>D 11d S0o</td>
<td>D</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>C0r D 12c</td>
<td>19</td>
<td>20</td>
<td>C0t D 12d</td>
<td>D E2a</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>D E0b S0i U2t</td>
<td>21</td>
<td>22</td>
<td>D E0a S0c U2r</td>
<td>D M P1</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>D S01</td>
<td>23</td>
<td>24</td>
<td>C1r D 13c U1t</td>
<td>D M</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>D P0</td>
<td>25</td>
<td>26</td>
<td>C1t D 13d U1r</td>
<td>D M</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>D 0 P0 Q0b</td>
<td>27</td>
<td>28</td>
<td>D P0 S10</td>
<td>D L P1</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>D E S11 P0</td>
<td>29</td>
<td>30</td>
<td>D P0 S1o</td>
<td>D L P1 U6t</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>D E S1c P0</td>
<td>31</td>
<td>32</td>
<td>ADC VDD</td>
<td>D L</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>A 4</td>
<td>33</td>
<td>34</td>
<td>ADC GND</td>
<td>D L Q1b</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>A 6</td>
<td>35</td>
<td>36</td>
<td>A 5</td>
<td>D L Q1a</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>A 2</td>
<td>37</td>
<td>38</td>
<td>A 3</td>
<td>D L U5t</td>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td>A 0</td>
<td>39</td>
<td>40</td>
<td>A 1</td>
<td>D L P1</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>D P0</td>
<td>41</td>
<td>42</td>
<td>D Q0a S11 U3t</td>
<td>D L P1</td>
<td>41</td>
<td>42</td>
</tr>
<tr>
<td>GND</td>
<td>43</td>
<td>44</td>
<td>GND</td>
<td>D L P1</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>GND</td>
<td>45</td>
<td>46</td>
<td>GND</td>
<td>D L P1</td>
<td>45</td>
<td>46</td>
</tr>
</tbody>
</table>

## Table 2.63: Bone ADC

<table>
<thead>
<tr>
<th>Index</th>
<th>Header pin</th>
<th>Black/Al-64</th>
<th>Al</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P9_39</td>
<td>in_voltage0_raw</td>
<td>in_voltage0_raw</td>
</tr>
<tr>
<td>1</td>
<td>P9_40</td>
<td>in_voltage1_raw</td>
<td>in_voltage1_raw</td>
</tr>
<tr>
<td>2</td>
<td>P9_37</td>
<td>in_voltage2_raw</td>
<td>in_voltage3_raw</td>
</tr>
<tr>
<td>3</td>
<td>P9_38</td>
<td>in_voltage3_raw</td>
<td>in_voltage2_raw</td>
</tr>
<tr>
<td>4</td>
<td>P9_33</td>
<td>in_voltage4_raw</td>
<td>in_voltage7_raw</td>
</tr>
<tr>
<td>5</td>
<td>P9_36</td>
<td>in_voltage5_raw</td>
<td>in_voltage6_raw</td>
</tr>
<tr>
<td>6</td>
<td>P9_35</td>
<td>in_voltage6_raw</td>
<td>in_voltage4_raw</td>
</tr>
</tbody>
</table>

### 2.7. Capes
Table 2.64: Bone ADC Overlay

<table>
<thead>
<tr>
<th>Internal</th>
<th>External (STMPE811)</th>
<th>AI-64</th>
<th>TBD</th>
<th>BONE-ADC.dts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>AI</td>
<td>AI-64</td>
<td>overlay</td>
<td></td>
</tr>
</tbody>
</table>

PWM  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

Table 2.65: PWM pins

<table>
<thead>
<tr>
<th>P9</th>
<th>Functions</th>
<th>odd</th>
<th>even</th>
<th>Functions</th>
<th>odd</th>
<th>even</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB D+</td>
<td>E1</td>
<td>E2</td>
<td>USB D-</td>
<td>E3</td>
<td>E4</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>5V OUT</td>
<td>1</td>
<td>2</td>
<td>GND</td>
<td>3</td>
<td>4</td>
<td>3V3 OUT</td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>5</td>
<td>6</td>
<td>5V IN</td>
<td>D M</td>
<td>3</td>
<td>4</td>
<td>D M</td>
</tr>
<tr>
<td>5V OUT</td>
<td>7</td>
<td>8</td>
<td>3V3 OUT</td>
<td>D M</td>
<td>9</td>
<td>10</td>
<td>D M</td>
</tr>
<tr>
<td>PWR BUT</td>
<td>2</td>
<td>7</td>
<td>5V OUT</td>
<td>D M</td>
<td>9</td>
<td>10</td>
<td>D M</td>
</tr>
<tr>
<td>D U4r</td>
<td>11</td>
<td>12</td>
<td>D</td>
<td>11</td>
<td>12</td>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>D U4t</td>
<td>13</td>
<td>14</td>
<td>D E1a</td>
<td>D E2b</td>
<td>13</td>
<td>14</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>15</td>
<td>16</td>
<td>D E1b</td>
<td>D P0i</td>
<td>15</td>
<td>16</td>
<td>D P0i</td>
</tr>
<tr>
<td>D 11c S00</td>
<td>17</td>
<td>18</td>
<td>D 11d S0o</td>
<td>D</td>
<td>17</td>
<td>18</td>
<td>D</td>
</tr>
<tr>
<td>C0r D 12c</td>
<td>19</td>
<td>20</td>
<td>C0t D 12d</td>
<td>D E2a</td>
<td>19</td>
<td>20</td>
<td>D M P1</td>
</tr>
<tr>
<td>D E0b S0i U2t</td>
<td>21</td>
<td>22</td>
<td>D E0a S0c U2r</td>
<td>D M P1</td>
<td>21</td>
<td>22</td>
<td>D M Q2b</td>
</tr>
<tr>
<td>D S01</td>
<td>23</td>
<td>24</td>
<td>C1r D 13c U1t</td>
<td>D M</td>
<td>23</td>
<td>24</td>
<td>D M</td>
</tr>
<tr>
<td>D P0</td>
<td>25</td>
<td>26</td>
<td>C1t D 13d U1r</td>
<td>D M</td>
<td>25</td>
<td>26</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>27</td>
<td>28</td>
<td>D P0 S10</td>
<td>D L P1</td>
<td>27</td>
<td>28</td>
<td>D L P1</td>
</tr>
<tr>
<td>D E S1t P0</td>
<td>29</td>
<td>30</td>
<td>D P0 S1o</td>
<td>D L P1</td>
<td>29</td>
<td>30</td>
<td>D L P1</td>
</tr>
<tr>
<td>D E S1c P0</td>
<td>31</td>
<td>32</td>
<td>ADC VDD</td>
<td>D L</td>
<td>31</td>
<td>32</td>
<td>D L</td>
</tr>
<tr>
<td>A 4</td>
<td>33</td>
<td>34</td>
<td>ADC GND</td>
<td>D L Q1b</td>
<td>33</td>
<td>34</td>
<td>D E L</td>
</tr>
<tr>
<td>A 6</td>
<td>35</td>
<td>36</td>
<td>A 5</td>
<td>D L Q1a</td>
<td>35</td>
<td>36</td>
<td>D E L</td>
</tr>
<tr>
<td>A 2</td>
<td>37</td>
<td>38</td>
<td>A 3</td>
<td>D L U5t</td>
<td>37</td>
<td>38</td>
<td>D L U5r</td>
</tr>
<tr>
<td>A 0</td>
<td>39</td>
<td>40</td>
<td>A 1</td>
<td>D L P1</td>
<td>39</td>
<td>40</td>
<td>D L P1</td>
</tr>
<tr>
<td>D P0</td>
<td>41</td>
<td>42</td>
<td>D Q0a S11 U3t P0</td>
<td>D L P1</td>
<td>41</td>
<td>42</td>
<td>D L P1</td>
</tr>
<tr>
<td>GND</td>
<td>43</td>
<td>44</td>
<td>GND</td>
<td>D L P1</td>
<td>43</td>
<td>44</td>
<td>D L P1</td>
</tr>
<tr>
<td>GND</td>
<td>45</td>
<td>46</td>
<td>GND</td>
<td>D L P1</td>
<td>45</td>
<td>46</td>
<td>D L P1</td>
</tr>
</tbody>
</table>
Table 2.66: Bone bus PWM

<table>
<thead>
<tr>
<th>Bone bus</th>
<th>Black</th>
<th>AI</th>
<th>AI-64</th>
<th>A</th>
<th>B</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/bone/pwm/PWM0</td>
<td></td>
<td>.</td>
<td>PWM1</td>
<td>P9.22</td>
<td>P9.21</td>
<td>BONE-PWM0.dts</td>
</tr>
<tr>
<td>/dev/bone/pwm/PWM1</td>
<td>PWM3</td>
<td>PWM2</td>
<td>P9.14</td>
<td>P9.16</td>
<td></td>
<td>BONE-PWM1.dts</td>
</tr>
<tr>
<td>/dev/bone/pwm/PWM2</td>
<td>PWM2</td>
<td>PWM0</td>
<td>P8.19</td>
<td>P8.13</td>
<td></td>
<td>BONE-PWM2.dts</td>
</tr>
</tbody>
</table>

**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 2.67: Bone TIMER PWMs

<table>
<thead>
<tr>
<th>Bone bus</th>
<th>Header pin</th>
<th>Black</th>
<th>AI</th>
<th>overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sys/bus/platform/devices/bone_timer_pwm_0/timer6</td>
<td></td>
<td>timer10</td>
<td>BONE-TIMER_PWM_0.dts</td>
<td></td>
</tr>
<tr>
<td>/sys/bus/platform/devices/bone_timer_pwm_1/timer4</td>
<td></td>
<td>timer11</td>
<td>BONE-TIMER_PWM_1.dts</td>
<td></td>
</tr>
<tr>
<td>/sys/bus/platform/devices/bone_timer_pwm_2/timer7</td>
<td></td>
<td>timer12</td>
<td>BONE-TIMER_PWM_2.dts</td>
<td></td>
</tr>
<tr>
<td>/sys/bus/platform/devices/bone_timer_pwm_3/timer13</td>
<td></td>
<td></td>
<td>BONE-TIMER_PWM_3.dts</td>
<td></td>
</tr>
<tr>
<td>/sys/bus/platform/devices/bone_timer_pwm_4/timer5</td>
<td></td>
<td>timer14</td>
<td>BONE-TIMER_PWM_4.dts</td>
<td></td>
</tr>
<tr>
<td>/sys/bus/platform/devices/bone_timer_pwm_5/timer15</td>
<td></td>
<td></td>
<td>BONE-TIMER_PWM_5.dts</td>
<td></td>
</tr>
</tbody>
</table>
### eQEP

#### Table 2.68: eQEP pins

<table>
<thead>
<tr>
<th>P9</th>
<th></th>
<th>P8</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>odd</td>
<td>even</td>
<td>Functions</td>
</tr>
<tr>
<td>USB D+</td>
<td>E1</td>
<td>E2</td>
<td>USB D-</td>
</tr>
<tr>
<td>3V OUT</td>
<td>E3</td>
<td>E4</td>
<td>GND</td>
</tr>
<tr>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
</tr>
<tr>
<td>3V3 OUT</td>
<td>3</td>
<td>4</td>
<td>3V3 OUT</td>
</tr>
<tr>
<td>5V IN</td>
<td>5</td>
<td>6</td>
<td>5V IN</td>
</tr>
<tr>
<td>5V OUT</td>
<td>7</td>
<td>8</td>
<td>5V OUT</td>
</tr>
<tr>
<td>PWR BUT</td>
<td>9</td>
<td>10</td>
<td>RESET</td>
</tr>
<tr>
<td>D U4r</td>
<td>11</td>
<td>12</td>
<td>D</td>
</tr>
<tr>
<td>D U4t</td>
<td>13</td>
<td>14</td>
<td>D E1a</td>
</tr>
<tr>
<td>D</td>
<td>15</td>
<td>16</td>
<td>D E1b</td>
</tr>
<tr>
<td>D I1c S00</td>
<td>17</td>
<td>18</td>
<td>D I1d S0o</td>
</tr>
<tr>
<td>C0r D I2c</td>
<td>19</td>
<td>20</td>
<td>C0t D I2d</td>
</tr>
<tr>
<td>D E0b S0i U2t</td>
<td>21</td>
<td>22</td>
<td>D E0a S0c U2r</td>
</tr>
<tr>
<td>D S01</td>
<td>23</td>
<td>24</td>
<td>C1r D I3c U1t</td>
</tr>
<tr>
<td>D P0</td>
<td>25</td>
<td>26</td>
<td>C1t D I3d U1r</td>
</tr>
<tr>
<td>D P0 Q0b</td>
<td>27</td>
<td>28</td>
<td>D P0 S10</td>
</tr>
<tr>
<td>D E S1i P0</td>
<td>29</td>
<td>30</td>
<td>D P0 S1o</td>
</tr>
<tr>
<td>D E S1c P0</td>
<td>31</td>
<td>32</td>
<td>ADC VDD</td>
</tr>
<tr>
<td>A 4</td>
<td>33</td>
<td>34</td>
<td>ADC GND</td>
</tr>
<tr>
<td>A 6</td>
<td>35</td>
<td>36</td>
<td>A 5</td>
</tr>
<tr>
<td>A 2</td>
<td>37</td>
<td>38</td>
<td>A 3</td>
</tr>
<tr>
<td>A 0</td>
<td>39</td>
<td>40</td>
<td>A 1</td>
</tr>
<tr>
<td>D P0</td>
<td>41</td>
<td>42</td>
<td>D Q0a S11 U3t P0</td>
</tr>
<tr>
<td>GND</td>
<td>43</td>
<td>44</td>
<td>GND</td>
</tr>
<tr>
<td>GND</td>
<td>45</td>
<td>46</td>
<td>GND</td>
</tr>
</tbody>
</table>

On BeagleBone's without an eQEP on specific pins, consider using the PRU to perform a software counter function.
## Table 2.69: Bone eQEP

<table>
<thead>
<tr>
<th>Bone bus</th>
<th>Black</th>
<th>AI</th>
<th>AI-64</th>
<th>A</th>
<th>B</th>
<th>strobe</th>
<th>index</th>
<th>overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/bone/eQEP0</td>
<td>eQEP2</td>
<td>eQEP0</td>
<td>eQEP0</td>
<td>P9.42</td>
<td>P9.27</td>
<td>• Black/AI-64: P9.25</td>
<td>• Black/AI-64: P9.41</td>
<td>• AI: P8.06</td>
</tr>
<tr>
<td>/dev/bone/eQEP1</td>
<td>eQEP0</td>
<td>eQEP1</td>
<td>eQEP1</td>
<td>P8.35</td>
<td>P8.33</td>
<td>• Black/AI-64: P8.32</td>
<td>• Black/AI-64: P8.31</td>
<td>• AI: P9.21</td>
</tr>
<tr>
<td>/dev/bone/eQEP2</td>
<td>eQEP1</td>
<td>–</td>
<td>–</td>
<td>P8.12</td>
<td>P8.22</td>
<td>• Black: P8.15</td>
<td>• Black: P8.15</td>
<td>• AI: P8.18</td>
</tr>
</tbody>
</table>

**eCAP** #TODO: This doesn’t include any abstraction yet.
### Table 2.70: ECAP pins

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Odd</th>
<th>Even</th>
<th>Function</th>
<th>Odd</th>
<th>Even</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P9</td>
<td>USB D+</td>
<td>E1</td>
<td>E2</td>
<td>USB D-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3V OUT</td>
<td>E3</td>
<td>E4</td>
<td>GND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
</tr>
<tr>
<td></td>
<td>3V3 OUT</td>
<td>3</td>
<td>4</td>
<td>3V3 OUT</td>
<td>3</td>
<td>4</td>
<td>DM</td>
</tr>
<tr>
<td></td>
<td>5V IN</td>
<td>5</td>
<td>6</td>
<td>5V IN</td>
<td>5</td>
<td>6</td>
<td>DM C4t</td>
</tr>
<tr>
<td></td>
<td>5V OUT</td>
<td>7</td>
<td>8</td>
<td>5V OUT</td>
<td>7</td>
<td>8</td>
<td>DC2r</td>
</tr>
<tr>
<td></td>
<td>PWR BUT</td>
<td>9</td>
<td>10</td>
<td>RESET</td>
<td>9</td>
<td>10</td>
<td>DC3r</td>
</tr>
<tr>
<td></td>
<td>D U4r</td>
<td>11</td>
<td>12</td>
<td>D</td>
<td>11</td>
<td>12</td>
<td>D P0o</td>
</tr>
<tr>
<td></td>
<td>D U4t</td>
<td>13</td>
<td>14</td>
<td>D E1a</td>
<td>13</td>
<td>14</td>
<td>D Q2a</td>
</tr>
<tr>
<td></td>
<td>D E1b</td>
<td>15</td>
<td>16</td>
<td>D E1b</td>
<td>15</td>
<td>16</td>
<td>D P0i</td>
</tr>
<tr>
<td></td>
<td>D 11c</td>
<td>17</td>
<td>18</td>
<td>D 11d</td>
<td>17</td>
<td>18</td>
<td>D C3t</td>
</tr>
<tr>
<td></td>
<td>S00</td>
<td></td>
<td></td>
<td>S00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C0r</td>
<td>19</td>
<td>20</td>
<td>C0t</td>
<td>19</td>
<td>20</td>
<td>DM P1</td>
</tr>
<tr>
<td></td>
<td>D 12c</td>
<td></td>
<td></td>
<td>D 12d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D E0b</td>
<td>21</td>
<td>22</td>
<td>D E0a</td>
<td>21</td>
<td>22</td>
<td>DM P1</td>
</tr>
<tr>
<td></td>
<td>S01</td>
<td>23</td>
<td>24</td>
<td>C1r</td>
<td>23</td>
<td>24</td>
<td>D M P1</td>
</tr>
<tr>
<td></td>
<td>D S01</td>
<td></td>
<td></td>
<td>C1t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D P0</td>
<td>25</td>
<td>26</td>
<td>D 13d</td>
<td>25</td>
<td>26</td>
<td>D M</td>
</tr>
<tr>
<td></td>
<td>P0 Q0b</td>
<td>27</td>
<td>28</td>
<td>P0 S10</td>
<td>27</td>
<td>28</td>
<td>D L P1</td>
</tr>
<tr>
<td></td>
<td>D E S11</td>
<td>29</td>
<td>30</td>
<td>D P0 S10</td>
<td>29</td>
<td>30</td>
<td>D L P1</td>
</tr>
<tr>
<td></td>
<td>P0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D E S1c</td>
<td>31</td>
<td>32</td>
<td>ADC</td>
<td>31</td>
<td>32</td>
<td>D L</td>
</tr>
<tr>
<td></td>
<td>S0 P0</td>
<td></td>
<td></td>
<td>VDD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>33</td>
<td>34</td>
<td>ADC</td>
<td>33</td>
<td>34</td>
<td>ADC</td>
<td>D L Q1b</td>
</tr>
<tr>
<td>A6</td>
<td>35</td>
<td>36</td>
<td>A5</td>
<td>35</td>
<td>36</td>
<td>A5</td>
<td>D L Q1a</td>
</tr>
<tr>
<td>A2</td>
<td>37</td>
<td>38</td>
<td>A3</td>
<td>37</td>
<td>38</td>
<td>A3</td>
<td>D L U5t</td>
</tr>
<tr>
<td>A0</td>
<td>39</td>
<td>40</td>
<td>A1</td>
<td>39</td>
<td>40</td>
<td>A1</td>
<td>D L P1</td>
</tr>
<tr>
<td>D P0</td>
<td>41</td>
<td>42</td>
<td>D Q0a</td>
<td>41</td>
<td>42</td>
<td>D Q0a</td>
<td>D L P1</td>
</tr>
<tr>
<td></td>
<td>S11 U3t</td>
<td></td>
<td></td>
<td>S11 U3t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P0</td>
<td></td>
<td></td>
<td>P0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td>43</td>
<td>44</td>
<td>GND</td>
<td>43</td>
<td>44</td>
<td>GND</td>
<td>D L P1</td>
</tr>
<tr>
<td>GND</td>
<td>45</td>
<td>46</td>
<td>GND</td>
<td>45</td>
<td>46</td>
<td>GND</td>
<td>D E L P1</td>
</tr>
</tbody>
</table>

### Table 2.71: Black eCAP PWMs

<table>
<thead>
<tr>
<th>Bone bus</th>
<th>Header pin</th>
<th>Peripheral</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sys/bus/platform/drivers/ecap/48302100.ecap</td>
<td>P9.42</td>
<td>eCAP0_in_PWM0_out</td>
<td>BBB-ECAP0.dts</td>
</tr>
<tr>
<td>/sys/bus/platform/drivers/ecap/48304100.ecap</td>
<td>P9.28</td>
<td>eCAP2_in_PWM2_out</td>
<td>BBB-ECAP2.dts</td>
</tr>
</tbody>
</table>
Table 2.72: AI eCAP PWMs

<table>
<thead>
<tr>
<th>Bone bus</th>
<th>Header pin</th>
<th>peripheral</th>
<th>overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sys/bus/platform/drivers/ecap/4843e100.ecap</td>
<td>P8.15</td>
<td>eCAP1_in_PWM1_out</td>
<td>BBAI-ECAP1.dts</td>
</tr>
<tr>
<td>/sys/bus/platform/drivers/ecap/48440100.ecap</td>
<td>P8.14</td>
<td>eCAP2_in_PWM2_out</td>
<td>BBAI-ECAP2.dts</td>
</tr>
<tr>
<td>/sys/bus/platform/drivers/ecap/48440100.ecap</td>
<td>P8.20</td>
<td>eCAP2_in_PWM2_out</td>
<td>BBAI-ECAP2A.dts</td>
</tr>
<tr>
<td>/sys/bus/platform/drivers/ecap/48442100.ecap</td>
<td>P8.04</td>
<td>eCAP3_in_PWM3_out</td>
<td>BBAI-ECAP3.dts</td>
</tr>
<tr>
<td>/sys/bus/platform/drivers/ecap/48442100.ecap</td>
<td>P8.26</td>
<td>eCAP3_in_PWM3_out</td>
<td>BBAI-ECAP3A.dts</td>
</tr>
</tbody>
</table>

**eMMC**

Table 2.73: Bone eMMC

<table>
<thead>
<tr>
<th>Header pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.3</td>
<td>DAT6</td>
</tr>
<tr>
<td>P8.4</td>
<td>DAT7</td>
</tr>
<tr>
<td>P8.5</td>
<td>DAT2</td>
</tr>
<tr>
<td>P8.6</td>
<td>DAT3</td>
</tr>
<tr>
<td>P8.20</td>
<td>CMD</td>
</tr>
<tr>
<td>P8.21</td>
<td>CLK</td>
</tr>
<tr>
<td>P8.22</td>
<td>DAT5</td>
</tr>
<tr>
<td>P8.23</td>
<td>DAT4</td>
</tr>
<tr>
<td>P8.24</td>
<td>DAT1</td>
</tr>
<tr>
<td>P8.25</td>
<td>DAT0</td>
</tr>
</tbody>
</table>

Table 2.74: Bone eMMC Overlay

<table>
<thead>
<tr>
<th>Black</th>
<th>AI</th>
<th>overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMC2</td>
<td>MMC3</td>
<td>BONE-eMMC.dts</td>
</tr>
</tbody>
</table>

**LCD**

Table 2.75: 16bit LCD interface

<table>
<thead>
<tr>
<th>Header pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8_45</td>
<td>lcd_data0</td>
</tr>
<tr>
<td>P8_46</td>
<td>lcd_data1</td>
</tr>
<tr>
<td>P8_43</td>
<td>lcd_data2</td>
</tr>
<tr>
<td>P8_44</td>
<td>lcd_data3</td>
</tr>
<tr>
<td>P8_41</td>
<td>lcd_data4</td>
</tr>
<tr>
<td>P8_42</td>
<td>lcd_data5</td>
</tr>
<tr>
<td>P8_39</td>
<td>lcd_data6</td>
</tr>
<tr>
<td>P8_40</td>
<td>lcd_data7</td>
</tr>
<tr>
<td>P8_37</td>
<td>lcd_data8</td>
</tr>
<tr>
<td>P8_38</td>
<td>lcd_data9</td>
</tr>
<tr>
<td>P8_36</td>
<td>lcd_data10</td>
</tr>
<tr>
<td>P8_34</td>
<td>lcd_data11</td>
</tr>
<tr>
<td>P8_35</td>
<td>lcd_data12</td>
</tr>
<tr>
<td>P8_33</td>
<td>lcd_data13</td>
</tr>
<tr>
<td>P8_31</td>
<td>lcd_data14</td>
</tr>
<tr>
<td>P8_32</td>
<td>lcd_data15</td>
</tr>
<tr>
<td>P8_27</td>
<td>lcd_vsync</td>
</tr>
<tr>
<td>P8_29</td>
<td>lcd_hsync</td>
</tr>
<tr>
<td>P8_28</td>
<td>lcd_pclk</td>
</tr>
<tr>
<td>P8_30</td>
<td>lcd_ac_bias_en</td>
</tr>
</tbody>
</table>
### Table 2.76: 16bit LCD interface Overlay

<table>
<thead>
<tr>
<th>Black</th>
<th>AI</th>
<th>overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcdc</td>
<td>dss</td>
<td></td>
</tr>
</tbody>
</table>

### McASP

#### Table 2.77: Bone McASP0

<table>
<thead>
<tr>
<th>Header pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P9.12</td>
<td>aclkr</td>
</tr>
<tr>
<td>P9.25</td>
<td>ahclkx</td>
</tr>
<tr>
<td>P9.27</td>
<td>fsr</td>
</tr>
<tr>
<td>P9.28</td>
<td>Black: axr2 AI: axr9</td>
</tr>
<tr>
<td>P9.29</td>
<td>fsx</td>
</tr>
<tr>
<td>P9.30</td>
<td>Black: axr0 AI: axr10</td>
</tr>
<tr>
<td>P9.31</td>
<td>aclkx</td>
</tr>
</tbody>
</table>

#### Table 2.78: Bone McASP0 Overlay

<table>
<thead>
<tr>
<th>Black</th>
<th>AI</th>
<th>overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>McASP0</td>
<td>McASP1</td>
<td></td>
</tr>
</tbody>
</table>

### PRU


- `/dev/remoteproc/prussX-coreY` (AM3358 X = “”, other x = “112”)

#### Table 2.79: Bone PRU eCAP

<table>
<thead>
<tr>
<th>Header Pin</th>
<th>Black</th>
<th>AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.15</td>
<td>pr1_ecap0</td>
<td>pr1_ecap0</td>
</tr>
<tr>
<td>P8.32</td>
<td>.</td>
<td>pr2_ecap0</td>
</tr>
<tr>
<td>P9.42</td>
<td>pr1_ecap0</td>
<td>.</td>
</tr>
</tbody>
</table>

#### Table 2.80: AI PRU UART

<table>
<thead>
<tr>
<th>UART</th>
<th>TX</th>
<th>RX</th>
<th>RTSn</th>
<th>CTSn</th>
<th>Overlays</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRU1 UART0</td>
<td>P8_31</td>
<td>P8_33</td>
<td>P8_34</td>
<td>P8_35</td>
<td></td>
</tr>
<tr>
<td>PRU2 UART0</td>
<td>P8_43</td>
<td>P8_44</td>
<td>P8_45</td>
<td>P8_46</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 2.81: Bone PRU

<table>
<thead>
<tr>
<th>Header Pin</th>
<th>Black</th>
<th>AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.03</td>
<td>.</td>
<td>pr2_pru0 10</td>
</tr>
<tr>
<td>P8.04</td>
<td>.</td>
<td>pr2_pru0 11</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Header Pin</th>
<th>Black</th>
<th>AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.05</td>
<td>•</td>
<td>pr2_pru0 06</td>
</tr>
<tr>
<td>P8.06</td>
<td>•</td>
<td>pr2_pru0 07</td>
</tr>
<tr>
<td>P8.07</td>
<td>•</td>
<td>pr2_pru1 16</td>
</tr>
<tr>
<td>P8.08</td>
<td>•</td>
<td>pr2_pru0 20</td>
</tr>
<tr>
<td>P8.09</td>
<td>•</td>
<td>pr2_pru1 06</td>
</tr>
<tr>
<td>P8.10</td>
<td>•</td>
<td>pr2_pru1 15</td>
</tr>
<tr>
<td>P8.11</td>
<td>pr1_pru0 15 (Out)</td>
<td>pr1_pru0 04</td>
</tr>
<tr>
<td>P8.12</td>
<td>pr1_pru0 14 (Out)</td>
<td>pr1_pru0 03</td>
</tr>
<tr>
<td>P8.13</td>
<td>•</td>
<td>pr1_pru1 07</td>
</tr>
<tr>
<td>P8.14</td>
<td>•</td>
<td>pr1_pru1 09</td>
</tr>
<tr>
<td>P8.15</td>
<td>pr1_pru0 15 (In)</td>
<td>pr1_pru1 16</td>
</tr>
<tr>
<td>P8.16</td>
<td>pr1_pru0 14 (In)</td>
<td>pr1_pru1 18</td>
</tr>
<tr>
<td>P8.17</td>
<td>•</td>
<td>pr2_pru0 15</td>
</tr>
<tr>
<td>P8.18</td>
<td>•</td>
<td>pr1_pru1 05</td>
</tr>
<tr>
<td>P8.19</td>
<td>•</td>
<td>pr1_pru1 06</td>
</tr>
<tr>
<td>P8.20</td>
<td>•</td>
<td>pr2_pru0 03</td>
</tr>
<tr>
<td>P8.21</td>
<td>•</td>
<td>pr2_pru0 02</td>
</tr>
<tr>
<td>P8.22</td>
<td>•</td>
<td>pr2_pru0 09</td>
</tr>
<tr>
<td>P8.23</td>
<td>•</td>
<td>pr2_pru0 08</td>
</tr>
<tr>
<td>P8.24</td>
<td>•</td>
<td>pr2_pru0 05</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Header Pin</th>
<th>Black</th>
<th>AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.25</td>
<td>•</td>
<td>pr2_pru0 04</td>
</tr>
<tr>
<td>P8.26</td>
<td>•</td>
<td>pr1_pru1 17</td>
</tr>
<tr>
<td>P8.27</td>
<td>•</td>
<td>pr2_pru1 17</td>
</tr>
<tr>
<td>P8.28</td>
<td>•</td>
<td>pr2_pru0 17</td>
</tr>
<tr>
<td>P8.29</td>
<td>•</td>
<td>pr2_pru0 18</td>
</tr>
<tr>
<td>P8.30</td>
<td>•</td>
<td>pr2_pru0 19</td>
</tr>
<tr>
<td>P8.31</td>
<td>•</td>
<td>pr2_pru0 11</td>
</tr>
<tr>
<td>P8.32</td>
<td>•</td>
<td>pr2_pru1 00</td>
</tr>
<tr>
<td>P8.33</td>
<td>•</td>
<td>pr2_pru0 10</td>
</tr>
<tr>
<td>P8.34</td>
<td>•</td>
<td>pr2_pru0 08</td>
</tr>
<tr>
<td>P8.35</td>
<td>•</td>
<td>pr2_pru0 09</td>
</tr>
<tr>
<td>P8.36</td>
<td>•</td>
<td>pr2_pru0 07</td>
</tr>
<tr>
<td>P8.37</td>
<td>•</td>
<td>pr2_pru0 05</td>
</tr>
<tr>
<td>P8.38</td>
<td>•</td>
<td>pr2_pru0 06</td>
</tr>
<tr>
<td>P8.39</td>
<td>•</td>
<td>pr2_pru0 03</td>
</tr>
<tr>
<td>P8.40</td>
<td>•</td>
<td>pr2_pru0 04</td>
</tr>
<tr>
<td>P8.41</td>
<td>•</td>
<td>pr2_pru0 01</td>
</tr>
<tr>
<td>P8.42</td>
<td>•</td>
<td>pr2_pru0 02</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Header Pin</th>
<th>Black</th>
<th>AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8.43</td>
<td></td>
<td>pr2_pru1 20</td>
</tr>
<tr>
<td>P8.44</td>
<td></td>
<td>pr2_pru0 00</td>
</tr>
<tr>
<td>P8.45</td>
<td></td>
<td>pr2_pru1 18</td>
</tr>
<tr>
<td>P8.46</td>
<td></td>
<td>pr2_pru1 19</td>
</tr>
<tr>
<td>P9.11</td>
<td></td>
<td>pr2_pru0 14</td>
</tr>
<tr>
<td>P9.13</td>
<td></td>
<td>pr2_pru0 15</td>
</tr>
<tr>
<td>P9.14</td>
<td></td>
<td>pr1_pru1 14</td>
</tr>
<tr>
<td>P9.15</td>
<td></td>
<td>pr1_pru0 5</td>
</tr>
<tr>
<td>P9.16</td>
<td></td>
<td>pr1_pru1 15</td>
</tr>
<tr>
<td>P9.17</td>
<td></td>
<td>pr2_pru0 09</td>
</tr>
<tr>
<td>P9.18</td>
<td></td>
<td>pr2_pru0 08</td>
</tr>
<tr>
<td>P9.19</td>
<td></td>
<td>pr1_pru0 02</td>
</tr>
<tr>
<td>P9.20</td>
<td></td>
<td>pr1_pru0 01</td>
</tr>
<tr>
<td>P9.24</td>
<td>pr1_pru0 16 (In)</td>
<td></td>
</tr>
<tr>
<td>P9.25</td>
<td>pr1_pru0 07</td>
<td>pr2_pru0 05</td>
</tr>
<tr>
<td>P9.26</td>
<td>pr1_pru1 16 (In)</td>
<td>pr1_pru0 17</td>
</tr>
<tr>
<td>P9.27</td>
<td>pr1_pru0 05</td>
<td>pr1_pru1 11</td>
</tr>
<tr>
<td>P9.28</td>
<td>pr1_pru0 03</td>
<td>pr2_pru0 13</td>
</tr>
<tr>
<td>P9.29</td>
<td>pr1_pru0 01</td>
<td>pr2_pru1 11</td>
</tr>
<tr>
<td>P9.30</td>
<td>pr1_pru0 02</td>
<td>pr2_pru1 12</td>
</tr>
<tr>
<td>P9.31</td>
<td>pr1_pru0 00</td>
<td>pr2_pru1 10</td>
</tr>
<tr>
<td>P9.41</td>
<td>pr1_pru0 06</td>
<td>pr1_pru0 03</td>
</tr>
<tr>
<td>P9.42</td>
<td>pr1_pru0 04</td>
<td>pr1_pru0 10</td>
</tr>
</tbody>
</table>

**GPIO**  TODO<br>For each of the pins with a GPIO, there should be a symlink that comes from the names *
Methodology

The methodology 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.

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.

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

2.7.2 BeagleBoard.org BeagleBone Relay Cape

Relay Cape, as the name suggests, is a simple Cape with a relay 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.

### Installation

No special configuration is required. When you plug Cape into your BeagleBoard, it is automatically recognized by the Cape Universal function.

You can check to see if 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.kern`el 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 `/boot/uEnv.txt`).

```
uboot_overlay_addr0=BBORG_RELAY-00A2.dtbo
```

### Usage

```
ls /sys/class/leds
```

The directory “relay*” exists in the following directory. The LEDs can be controlled by modifying the files in this directory.

```
echo 1 > relay1/brightness
```
This allows you to adjust the brightness; entering 1 for brightness turns it ON, and entering 0 for OFF.
The four relays can be changed individually by changing the number after “relay.

2.8 BeagleConnect

**Important:** Currently under development

**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 boards-terms-and-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.

BeagleConnect™ is a revolutionary technology virtually eliminating low-level software development for IoT and IIoT applications, such as building automation, factory automation, home automation, and scientific data acquisition. While numerous IoT and IIoT solutions available today provide massive software libraries for microcontrollers supporting a limited body of sensors, actuators and indicators as well as libraries for communicating over various networks, BeagleConnect simply eliminates the need for these libraries by shifting the burden into the most massive and collaborative software project of all time, the Linux kernel.

These are the tools used to automate things in scientific data collection, data science, mechatronics, and IoT.
BeagleConnect™ technology solves:

- The need to write software to add a large set of diverse devices to your system,
• The need to maintain the software with security updates,
• The need to rapidly prototype using off-the-shelf software and hardware without wiring,
• The need to connect to devices using long-range, low-power wireless, and
• The need to produce high-volume custom hardware cost-optimized for your requirements.

2.8.1 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.
Background

BeagleConnect™ uses Greybus and updated Click Boards with identifiers to eliminate the need to add and manually configure devices added onto the Linux system.

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.
Software architecture

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
• Click Board plug-ins for node-red for the same 100 or so Click Boards
• BeagleConnect™ Freedom System Reference Manual and FAQs

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.
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

2.8.2 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. sensor test-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 wpansusb 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 #1b  
very Thu, 1 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:  
      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:  

(continues on next page)```
attr 0: data_available value: 0
attr 1: watermark value: 1

iio:device1: hdc2010
3 channels found:
humidity relative: (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

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: CMP 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
[ 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: 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 the USB and appears at 192.168.7.2:

```bash
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.

```bash
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:

```bash
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
```

(continues on next page)
Performing mass erase
Erasing all main bank flash sectors
Erase done
Writing 360448 bytes starting at address 0x00000000
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**
   
   ```bash
   sudo apt install -y tio
   ```

2. **Run tio**
   
   ```bash
   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.

```bash
uart:~$ *** Booting Zephyr OS build 9c8588c863223 ***
[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 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...

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,
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)>
dissociate ;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 on-going work. The instructions here only cover checking out the demo branch which should stay in a tested state. On-going 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/\n--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
```

(continues on next page)
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
```

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:
          [1] 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,
```

(continues on next page)
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: 2,4,5,6,7,8
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

```bash
sudo iwpan phy phy0 set channel 0 26
```

1. Then, set the PAN identifier for the 802.15.4 device on the Linux machine

```bash
sudo iwpan dev wpan0 set pan_id 0xabcd
```

2. Associate the wpan0 device to a new, 6lowpan network interface

```bash
sudo ip link add link wpan0 name lowpan0 type lowpan
```

1. Finally, set the links up for both wpan0 and lowpan0

```bash
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.

```bash
$ ping6 -I lowpan0 ff02::1
PING ff02::1 (ff02::1) from fe80::9c0b:a4e8:d3:4553%lowpan0 64 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::9c0b:a4e8:d3:4553%lowpan0: icmp_seq=3 ttl=64 time=17.3 ms (DUP!,
→ operator is 'or')
64 bytes from fe80::9c0b:a4e8:d3:4553%lowpan0: icmp_seq=4 ttl=64 time=14.9 ms (DUP!,
→ operator is 'or')
```

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::c9f9:a11c:4b:1200
PING fe80::c9f9:a11c:4b:1200(fe80::c9f9:a11c:4b:1200) from fe80::9c0b:a4e8:d3:4553
→%lowpan0 lowpan0: 56 data bytes
64 bytes from fe80::c9f9:a11c:4b:1200%lowpan0: icmp_seq=1 ttl=64 time=16.0 ms
64 bytes from fe80::c9f9:a11c:4b:1200%lowpan0: icmp_seq=2 ttl=64 time=13.8 ms
64 bytes from fe80::c9f9:a11c:4b:1200%lowpan0: icmp_seq=3 ttl=64 time=9.77 ms
64 bytes from fe80::c9f9: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.

```bash
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
PING fe80::9c0b:a4e8:d3:4553 8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::c9f9:a11c:4b:1200: icmp_seq=0 ttl=64
        rssi=110 time=11 ms
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::c9f9:a11c:4b:1200: icmp_seq=1 ttl=64
        rssi=126 time=9 ms
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::c9f9:a11c:4b:1200: icmp_seq=2 ttl=64
        rssi=128 time=13 ms
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::c9f9:a11c:4b:1200: icmp_seq=3 ttl=64
        rssi=126 time=10 ms
8 bytes from fe80::9c0b:a4e8:d3:4553 to fe80::c9f9: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

```bash
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.

```bash
$ ip a show lowpan0
37: lowpan0@wpan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1280 qdisc noqueue state...
    UNITS 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.

```bash
$ 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:

```bash
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:

```bash
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.

```bash
#!/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 ((i=0; i<$NPINS; i++)); do
  for j in ${!PINS[@]}; do
    # (continues on next page)
```
# 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
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...
  (continues on next page)
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/greybus1/1-2/1-2.1/1-2.2/1-2.2.2/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 straightforward 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.

2.8.3 BeagleConnect™ Story

There are many stories behind BeagleConnect™, mine is just one of them. It begins with my mom teaching me about computers. She told me I could anything I wanted with ours, as long as I didn’t open the case. This was the late-70s/early-80s, so all she needed to do was put her floppy disk away and there wasn’t risk of me damaging the family photo album or her ability to do her work the next day. I listened and learned from her the basics of programming, but it wasn’t long before I wanted to take the computer apart.

Initially exploring Getting Started in Electronics satisfied my itch for quite a while. Eventually, I got a Commodore 64 and began connecting voice synthesizer ICs to it. My interest in computers and electronics flourished into an electrical engineering degree and a long career in the semiconductor industry.

Over this time, I’ve become more and more alarmed with the progress of technology. Now, to be clear, I love technology. I love innovation and invention. It is just that some things have evolved in a sort of tunnel-vision, without bringing everyone along.

But, what about keyboard users? As graphical user interfaces and mice took over computers, they rapidly became almost unusable by my mom. She typed well, but the dexterity to move a mouse eluded her. To satisfy the need to interact with locations on the screen, she adopted using a joystick and her productivity came to a crawl. How is it that such assumptions could be made impacting all computer users without any thoughtful provisions for what already worked?

2.8.4 BeagleConnect Experience

BeagleConnect™ provides a scalable experience for interacting with the physical world.

Note: The term BeagleConnect™ refers to a technology comprising of a family of boards, a collection of Linux kernel drivers, microcontroller firmware, a communication protocol, and system-level integration to automation software tools. More specific terms will be applied in the architecture details. The term is also used here to represent the experience introduced to users through the initial BeagleConnect™ Freedom product consisting of a board and case which ships programmed and ready to be used.

For scientists, we are integrating Jupyter Notebook with the data streams from any of hundreds of sensor options, including vibration, gas detection, biometrics and more. These data streams can be stored in simple data files <https://en.wikipedia.org/wiki/Comma-separated_values> or processed and visualized.

#TODO: provide images demonstrating Jupyter Notebook visualization

For embedded systems developers, data is easily extracted using the standard IIO interface provided by the Linux kernel running on the gateway using any of hundreds of programming languages and environments, without writing a line of microcontroller firmware. The Linux environment provides opportunities for high-level remote management using tools like Balena with applications deployed in Docker containers.
#TODO: provide image illustrating remote management
The hardware and software are fully open source, providing for scalability and a lack of vendor lock-in.
For DevOps…
For home automaters, integration into WebThings…
#TODO: think a bit more about this section with some feedback from Cathy.

## 2.8.5 BeagleConnect Boards

Get started using your BeagleConnect.

**BeagleConnect Freedom**

**Important:** Currently under development
The initial BeagleConnect™ Freedom production release will:

- Support at least 100 mikroBUS-based Click boards from Mikroelectronika
- Work with Bluetooth Low Energy (BLE)-enabled Linux computers at 2.4GHz
- Work with long-range sub-1GHz IEEE 802.15.4 wireless connections at 500 meters with data rates of 1kbps, and
- Work with a low-cost BeagleBoard.org Linux single-board computer (SBC) as a BeagleConnect™ gateway device and work with at least 10 other BeagleConnect™ node devices each supporting 2 add-on sensor, actuator or indicator devices.

Future releases will be collaborated with the community, evolve dynamically, and contain additional functionality. The goal is to support over 500 add-on devices within the first year after the initial release.

**Important:** BeagleConnect™ Freedom enables wirelessly adding new device nodes and is targeted to cost initially around US$20 with a roadmap to variants as low as US$1.

**BeagleConnect™ Freedom** BeagleConnect™ Freedom is based on a TI Arm Cortex-M4 wireless-enabled microcontroller and is the first available BeagleConnect™ solution. It implements:

- BeagleConnect™ gateway device function for Sub-GHz 802.15.4 long-range wireless
- BeagleConnect™ node device function for Bluetooth Low-Energy (BLE) and Sub-GHz 802.15.4 long range wireless
- USB-based serial console and firmware updates
- 2x mikroBUS sockets with BeagleConnect™ protocol support

#TODO: provide image of BeagleConnect™ Freedom in a case with a hand for size perspective
BeagleConnect™ Freedom beta kit  A small number of beta kits have been assembled with BeagleConnect™ Freedom rev C5 boards, which is the version that should be taken to production.

The kit includes:

- 1x Seeed BeagleBone® Green Gateway (board, USB cable)
- 3x BeagleConnect™ Freedom (board, antenna, USB cable)
- 1x Mikroelectronika Click ID Board

To get started with this kit, see [demo-1].

What makes BeagleConnect™ new and different?

**Important:** BeagleConnect™ solves IoT in a different and better way than any previous solution.

The device interface software is already done  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:** Bottom line: 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.

On-going maintenance  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, relatively 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).
Full customization possible  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 key pairs that can be replaced to secure your own network, and
- Fully FCC/CE certified.

Getting Started

- Typical BeagleConnect Freedom usage with a Linux host
- Programming BeagleConnect Freedom with Zephyr

BeagleConnect Freedom Usage  This section describes the usage model we are developing. To use the current code in development, please refer to the [development] section.

BeagleConnect wireless user experience

**The User Experience**

Step 1 – Gateway login

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

---

**The User Experience**

**Step 2 – Connect with button push**

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

BeagleConnect Freedom & Zephyr

Develop for BeagleConnect Freedom with 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 let's get started with one recommended set for the beta users.

Required

- **beta-kit**
  - Seeed Studio BeagleBone® Green Gateway
  - 3x BeagleConnect™ Freedom board, antenna, U.FL to SMA cable, SMA antenna and USB Type-A to Type-C cable
  - 1x MikroE ID Click
- microSD card (6GB or larger)
- microSD card programmer

Recommended

- 12V power brick
- USB to TTL 3.3V UART adapter
- Ethernet cable and Internet connection
- 2x USB power adapters
- BME280-based Weather Click
- iAQ-Core-based Air Quality 2 Click
Optional

• x86_64 computer running Ubuntu 20.04.3 LTS

Install the latest software image for BeagleBone Green Gateway

Download and install the Debian Linux operating system image for the Seeed BeagleBone® Green Gateway host.

1. Download the special mikroBUS/Greybus BeagleBoard.org Debian image from here. Pick the most recent directory and select the file beginning with bone- and ending with .img.xz. Today that file is bone-debian-11.2-iot-mikrobus-armhf-2022-03-04-4gb.img.xz.

2. Load this image to a microSD card using a tool like Etcher.

3. Insert the microSD card into the Green Gateway.

4. Power BeagleBone Green Gateway via the 12V barrel jack.

#TODO: describe how to know it is working

Other systems

Important: If you are using the image above, none of the instructions in this section are required.

1. Update the system.

   ```bash
   sudo apt update
   ```

2. Install all BeagleConnect™ management software.

   ```bash
   sudo apt install -y
   beagleconnect beagleconnect-msp430
   git vim
   build-essential
   cmake ninja-build gperf
   ccache dfu-util device-tree-compiler
   make gcc libstdc++-dev
   libxml2-dev libxslt-dev libssl-dev
   libtool
   ```

   ```bash
   echo "export PATH=$PATH:
   $HOME/.local/bin" >> $HOME/.bashrc
   ```

   ```bash
   source $HOME/.bashrc
   ```

3. Reboot

   ```bash
   sudo reboot
   ```

4. Install BeagleConnect™ flashing software

   ```bash
   pip3 install -U west
   ```

5. Reboot

   ```bash
   sudo reboot
   ```

Log into BeagleBone Green Gateway

These instructions assume an x86_64 computer running Ubuntu 20.04.3 LTS, but any computer can be used to connect to your BeagleBone Green Gateway.

1. Log onto the Seeed BeagleBone® Green Gateway using ssh.
We need IP address, Username, and Password to connect to the device.

- The default IP for the BeagleBone hardware is 192.168.7.2
- The default Username is `debian` and Password is `temppwd`
- To connect you can simply type `$ ssh debian@192.168.7.2` and when asked for password just type `temppwd`
- Congratulations, You are now connected to the device!

2. Connect to the WiFi

- Execute `sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan0.conf` and provide the password `temppwd` to edit the configuration file for the WiFi connection.

- Now edit the file (shown below) under the `network= {...}` section you can set you `ssid` (WiFi name) and `psk` (WiFi Password).

```plaintext
ctrl_interface=DIR=/run/wpa_supplicant GROUP=netdev
update_config=1
#country=IN
network={
    ssid="WiFi Name"
    psk="WiFi Password"
}
```

- Now save the file with CTRL+O and exit with CTRL+X.

- Check if the connection is established by executing `$ ping 8.8.8.8` you should see something like shown below.

```
debian@BeagleBone:~$ 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=10.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=5.72 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=6.13 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=6.11 ms
...```

- If everything goes well, you are ready to update your system and install new applications for beagleconnect.

### Install Zephyr development tools on BeagleBone Green Gateway

1. Download and setup Zephyr for BeagleConnect™

```plaintext
cd
west init -m https://github.com/jadonk/zephyr --mr bcf-sdk-3.1.0-rebase_ --bcf-zephyr
cd $HOME/bcf-zephyr
west update
west zephyr-export
pip3 install -r zephyr/scripts/requirements-base.txt
echo "export CROSS_COMPILE=/usr/bin/arm-none-eabi-" >> $HOME/.bashrc
echo "export ZEPHYR_BASE=$HOME/bcf-zephyr/zephyr" >> $HOME/.bashrc
echo "export PATH=$HOME/bcf-zephyr/zephyr/scripts:SPATH" >> $HOME/.bashrc
echo "export BOARD=beagleconnect_freedom" >> $HOME/.bashrc
source $HOME/.bashrc
```

### Build applications for BeagleConnect Freedom on BeagleBone Green Gateway

Now you can build various Zephyr applications

1. Change directory to BeagleConnect Freedom zephyr repository.
cd $HOME/bcf-zephyr

2. Build blinky example

west build -d build/blinky zephyr/samples/basic/blinky

3. TODO

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

4. TODO

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

5. TODO

west build -d build/bcfserial modules/lib/wpanusb_bc -- -DOVERLAY_
  → CONFIG=overlay-bcfserial.conf -DDTC_OVERLAY_FILE=bcfserial.overlay

6. TODO

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

Flash applications to BeagleConnect Freedom from BeagleBone Green Gateway

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#

2.9 BeagleBoard (all)

BeagleBoard boards are low-cost, ARM-based development boards suitable for rapid prototyping and open-hardware to enable professionals to develop production systems.

Note: Make sure to read and accept all the terms & condition provided in the boards-terms-and-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 PDF-formatted System Reference Manual for each BeagleBoard board is linked below.

- BeagleBoard
- BeagleBoard-xM
- BeagleBoard-X15
Chapter 3

Projects

This is a collection of reasonably well-supported projects useful to Beagle developers.

3.1 simpPRU

3.1.1 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.

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 typesafe, and data types of variables are decided during compilation.
- simpPRU codes have a .sim extension.
simpPRU provides a console app to use Remoteproc functionality.

### 3.1.2 Build from source

#### Dependencies

- flex
- bison
- gcc
- gcc-pru
- gnuprumcu
- cmake

#### Build

```bash
git clone https://github.com/VedantParanjape/simpPRU.git
cd simpPRU
mkdir build
cd build
cmake ..
maked
```

#### Install

```bash
sudo maked install
```

#### Generate debian package

```bash
sudo maked package
```

### 3.1.3 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
- gnuprumercu
- 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**

```bash
wget https://github.com/VedantParanjape/simpPRU/releases/download/1.4/simppru-1.4-→amd64.deb
sudo dpkg -i simppru-1.4-amd64.deb
```

**armhf**

```bash
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

### 3.1.4 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 typesafe, 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:**
```plaintext
bool out := 5 > 6;
```

**Wrong:**
```plaintext
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. ints and chars can be assigned to each other.
- Variables can be assigned expressions whose output is of same datatype.

Declaration
```plaintext
int var;
char char_var;
bool test_var;
```
Assignment during Declaration

```c
int var := 99;
char char_var := 'a';
uint8 short_var := 255;
bool test_var := false;
```

Assignment

```c
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.

```c
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:

```c
int a := arr[4]; /* Copy the 5th element of arr to a */
```

Changing elements of an array:

```c
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 */
arr[i+j] := 1; /* The 9th element of arr is now 1 */
```

(continues on next page)
/* 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

Reserved keywords

``true``
``false``
``int``
``bool``
``void``
``if``
``else``
``for``
``return``
``delay``
``digital_read``
``digital_write``
``start_counter``
``stop_counter``
``pwm``
``def``
``or``
``and``
``not``
``continue``
``break``
``while``
``in``
``init_message_channel``
``send_message``
``print``
``println``
``receive_message``

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


Expressions

Arithmetic expressions

=> (9 + 8) * 2 + -1;
33
=> 11 % 3;
2

(continues on next page)
Boolean expressions

```c
=> 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`, statement 1 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**

```c
if : boolean_expression { 
  statement 1
  ...
  ...
}
elif : boolean_expression { 
  statement 2
  ...
  ...
}
else { 
  statement 3
  ...
  ...
}
```

**Examples**

```c
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

```plaintext
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:

```plaintext
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.

```plaintext
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

```plaintext
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 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**

```java
while : boolean_expression {  
  statement 1  
  ...  
  ...  
}
```

**Examples**

- **Infinite loop**
  ```java
  while true {  
    do_something..  
    ...  
  }
  ```

- **Normal loop, will repeat 30 times, before exiting**
  ```java
  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**

```java
break;
```

**Examples**

```java
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**

```java
continue;
```

**Examples**

```java
for : j in 9:19 {  
  if : i == 12 {  
    continue;  
  }  
  else {  
    break;  
  }
}
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**

```python
def <function_name> : <data_type> : <data_type> <param_name>, <data_type> <param_> ... { 
    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.

- **Wrong**: `return` inside a compound statement, this syntax is not allowed.

```python
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

```python
def test : int : int a {
    int gf := 8;
    if : a < 4 {
        gf := 4;
    }
    return gf;
}
```

**Examples** Examples according to return types

- **Integer**

```python
def test_func : int : int a, int b {
    int aa := a + 5;
    if : aa < 3 {
        aa := 0;
    }
    return aa + b;
}
```

- **Character**
```python
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.
3.1.5 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, similiarly 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

```c
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

```c
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();
```

3.1. simpPRU
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 emp := receive_message();
if : emp >= 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”.

3.1. simpPRU
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

```c
init_message_channel();

if : digital_read(P1_29) {
    send_bool(true);
}
else {
    send_int(0);
}
```

3.1.6 Usage(simppru)

```
simppru [OPTION...] FILE

- -d --device=<device_name> Select for which BeagleBoard to compile
  (pocketbeagle, bbb, bbbwireless, bbai)
- -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
- -v, --verbose Enable verbose mode (dump symbol table and ast
  graph)
- -p, --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...
- -u, --usage Give this help list
- -v, --version Print program version
```

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.
simprru 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:

```
simprru test.sim --load
```

If we only want to generate binary for pru0

```
simprru test.sim -o test_firmware -p 0
```

this will generate a file named test_firmware.pru0

### 3.1.7 Usage(simprru-console)

simprru-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 rpsmg 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 simprru-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 an 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

3.1.8 simpPRU Examples

These are the examples which have been tested on simpPRU. These examples will serve as a guide for the users to implement.

simpPRU

Intuitive language for PRU which compiles down to PRU C.
Delay example

This codesnippet 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.

**Explanation**

Following code works on PocketBeagle, to use on other boards, please change the pins accordingly.

code

digital.Write(P1_31, true);
delay(2000);
digital.Write(P1_31, false);
 delays(5000);
digital.Write(P1_31, true);
Digital read example

![Digital read example diagram]

**Code**

```c
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

```
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)

```
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)
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.

Explaination

Ultrasonic range sensor example

Code

```python
def measure : int : {
  bool timeout := false;
  ...
```

(continues on next page)
int echo := 0;

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

```c
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.

The `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

```c
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
```c
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

```c
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

```c
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

![BeagleBoard](image)

**Code**

```c
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**

```c
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 upto 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**

```c
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**

3.1. simpPRU
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.

3.2 BB-Config

3.2.1 BB-Config Detail

Configure your beagle devices easily.

Github
What is BB-Config

BB-Config is a software that makes the most common low-level configuration changes in beagle devices easily and provides a terminal UI.

BB-Config is using FTXUI (C++ Functional Terminal User Interface) which have simple and elegant UI looking.
3.2.2 Build from Source

Dependencies

- g++
- cmake
- glib-2.0
- libnm

Build

```bash
git clone https://git.beagleboard.org/gsoc/bb-config
cd bb-config
mkdir build
cd build
cmake ..
make -j$(nproc)
```

Install

```bash
sudo make install
```

3.2.3 Features

BB-Config v1.x

PRU Enable/Disable

- Enable/Disable PRU
GPIO

- Turn On/Off gpio
### GPIO Setting

#### EMMC and MicroSD Stats

- Storage stats & grow partition

```
/rmcb0:21337 MB  21337 MB  21337 MB
/home/usb0:270845.53 MB  29277.34 MB  25764.68 MB
```

### 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:

```
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 Reference

Pin Config

Overlay (dts)
- Enable/Disable Device Tree Overlay in Boot option
- Select dtb0 file and automate update in uEnv.txt
WiFi (D-Bus)

- Connect to WiFi with wpa_supplicant
- Support for Debian 11

3.2.4 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/](https://satacker.github.io/gsoc-log/)
• **Kanban:** [https://github.com/SAtacker/beagle-config/projects/1](https://github.com/SAtacker/beagle-config/projects/1)
• **Initial Video:** [https://youtu.be/vFUWCzqE6xI](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](https://git.beagleboard.org/gsoc/bb-config)
• **Project Page:** [https://summerofcode.withgoogle.com/programs/2022/projects/2DbiYPlY](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](https://forum.beagleboard.org/t/weekly-progress-report-bb-config-improvements-gpio-benchmark/32357/2)
• **Initial Video:** [https://youtu.be/V_Euk5uWY1o](https://youtu.be/V_Euk5uWY1o)
Chapter 4

Books

This is a collection of open-source books written to help Beagle developers.

:ref:`BeagleBone Cookbook <beaglebone-cookbook>` is a great introduction to programming a BeagleBone using Linux from userspace, mostly using Python or JavaScript.

:ref:`PRU Cookbook <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.

4.1 BeagleBone Cookbook

Contributors

- Author: Mark A. Yoder
- Book revision: v2.0 beta

A cookbook for programming Beagles

4.1.1 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*.

![Plugging BeagleBone Black into a USB port](image)

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.


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:

```
bone$ sudo halt
```

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.
Fig. 4.2: The Bone appears as a USB drive
Fig. 4.3: Visual Studio Code
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 2022-07-01) 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 /ID.txt
BeagleBoard.org Debian Bullseye IoT Image 2022-07-01
```

I'm running the 2022-07-01 version.

Running the Python and JavaScript Examples

Problem  You'd like to learn Python and 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 Studio 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.

**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.

Change to the directory that contains your file, make it executable, and then run it:

```
bone$ cd ~/examples/BeagleBone/Black/
bone$ <strong>./seqLEDs.py</strong>
```

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.
Fig. 4.4: Visual Studio Code showing bash terminal
Fig. 4.5: Latest Debian images
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). 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 on board flash, or flashing a 4 GB or bigger microSD 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 500MB, so it might take a while.

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 XXXX 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.

#### 4.1.2 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 www.github.com. 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 I2C buses.
Fig. 4.7: The USB 2.0 host port

Fig. 4.8: Cape Headers P8 and P9
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.

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).
- 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
  ```
Fig. 4.10: Entering commands in the VSC bash tab
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 www.github.com. 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:

```
root@beaglebone:/boneSensors# ./pushbutton.js
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.

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](#).

The code in *Monitoring a pushbutton (pushbutton.js)* reads GPIO port `P9_42`, which is attached to the pushbutton.

```
Listing 4.1: Monitoring a pushbutton (pushbutton.py)

```python
#!/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 os
ms = 500  # Read time in ms
pin = '7'  # P9_42 is gpio 7
```

(continues on next page)
GPIOPATH="/sys/class/gpio"

# 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()

    f = open(GPIOPATH+"/gpio"+pin+"/value", "r")

while True:
    f.seek(0)
    data = f.read()[:-1]
    print("data = " + data)
    time.sleep(ms/1000)

pushbutton.py

Listing 4.2: Monitoring a pushbutton (pushbutton.js)
const fs = require("fs");

const ms = 500; // Read time in ms
const pin = "7"; // P9_42 is gpio 7
const GPIOPATH="/sys/class/gpio/";

// Make sure pin is exported
if(!fs.existsSync(GPIOPATH+"gpio"+pin)) {
    fs.writeFileSync(GPIOPATH+"export", pin);
}

// Make it an input pin
fs.writeFileSync(GPIOPATH+"gpio"+pin+"/direction", "in");

// Read every ms
setInterval(readPin, ms);

function readPin() {
    var data = fs.readFileSync(GPIOPATH+"gpio"+pin+"/value").slice(0, -1);
    console.log('data = ' + data);
}

pushbutton.js

Put this code in a file called pushbutton.js 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:

topushbutton.js
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 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's 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.

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

```
<table>
<thead>
<tr>
<th>gpiochip0</th>
<th>32 lines:</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 14: &quot;P9_26 [uart1_rxd]&quot; &quot;P9_26&quot; input active-high [used]</td>
<td></td>
</tr>
</tbody>
</table>
```

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.

To make this recipe, you will need:

- Breadboard and jumper wires.
- 10k trimpot or
- Flex resistor (optional)
- 22k resistor

A variable resistor with three terminals
Wiring a 10k 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).

Fig. 4.13: Wiring a 10k variable resistor (trimpot) to an ADC port

Reading an analog voltage (analogIn.js) shows the BoneScript code used to read the variable resistor. Add the code to a file called _analogIn.js_ and run it; then change the resistor and run it again. The voltage read will change.

Listing 4.3: Reading an analog voltage (analogIn.py)

```python
#!/usr/bin/env python3
#//////////////////////////////////////
# analogin.py
# Reads the analog value of the light sensor.
#//////////////////////////////////////
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 = '')
    time.sleep(0.1)

# // Bone | Pocket | AIN
# // ----- | ------ | ---
# // P9_39 | P1_19 | 0
```

(continues on next page)
Listing 4.4: Reading an analog voltage (analogIn.js)

```javascript
#!/usr/bin/env node
/
/// 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);
}
```

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 22k resistor to give a voltage to measure across the flex sensor.

The code in *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.

*Reading an analog voltage (ultrasonicRange.js)* shows the code that reads the sensor at a fixed interval.
Listing 4.5: Reading an analog voltage (ultrasonicRange.py)

```python
#!/usr/bin/env python
# // ultrasonicRange.js
# // Reads the analog value of the sensor.
# // ultrasonicRange.js

ms = 250;  # Time in milliseconds
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
```

Listing 4.6: Reading an analog voltage (ultrasonicRange.js)

```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');

setInterval(readPin, ms);

function readPin() {
    var data = fs.readFileSync(IIOPATH);
    console.log('data= ' + data);
}
```

(continues on next page)
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.

To make this recipe, you will need:

- Breadboard and jumper wires.
- 10 k and 20 k resistors
- HC-SR04 Ultrasonic 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 4.7: Driving a HC-SR04 ultrasound sensor (hc-sr04-ultraSonic.js)

```javascript
#!/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
ms = 250; // Trigger period in ms

var startTime, pulseTime;

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(echo, b.INPUT, 7, 'pulldown', 'fast', doAttach);

function ping() {
  b.digitalWrite(trigger, 1); // Unit triggers on a falling edge.
  b.digitalWrite(trigger, 0); // Set trigger to high so we can pull it low later
}

setInterval(ping, ms);
```

(continues on next page)
// 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) {
        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. 4.18: Wiring a rotary encoder using eQEP2
Table 4.1: On the BeagleBone and PocketBeage the three encoders are:

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>eQEP0</td>
<td>P9.27 and P9.42 OR P1_33 and P2_34</td>
</tr>
<tr>
<td>eQEP</td>
<td>P9.33 and P9.35</td>
</tr>
<tr>
<td>eQEP2</td>
<td>P8.11 and P8.12 OR P2_24 and P2_33</td>
</tr>
</tbody>
</table>

Table 4.2: On the AI it’s:

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>eQEP1</td>
<td>P8.33 and P8.35</td>
</tr>
<tr>
<td>eQEP2</td>
<td>P8.11 and P8.12 or P9.19 and P9.41</td>
</tr>
<tr>
<td>eQEP3</td>
<td>P8.24 and P8.25 or P9.27 and P9.42</td>
</tr>
</tbody>
</table>

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:

```bash
bone$ config-pin P8_11 eqep
bone$ config-pin P8_12 eqep
bone$ show-pins | grep eqep
```

This will enable *eQEP2* on pins P8_11 and P8_12. The 2 after the *eqep* returned by *show-pins* shows it’s *eQEP2*.

Finally, add the code in *Reading a rotary encoder (rotaryEncoder.js)* to a file named *rotaryEncoder.js* and run it.

**Listing 4.8: Reading a rotary encoder (rotaryEncoder.py)**

```python
#!/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
# // bone$ show-pins | grep eqep
import time

eQEP = '2'
COUNTERPATH = '/dev/bone/counter/counter'+'eQEP'+'/'+count0'
ms = 100  # Time between samples in ms
maxCount = '1000000'

# Set the eQEP maximum count
f = open(COUNTERPATH+'ceiling', 'w')
f.write(maxCount)
f.close()

# Enable
f = open(COUNTERPATH+'enable', 'w')
f.write('1')
```

(continues on next page)
f.close()

f = open(COUNTERPATH + '/count', 'r')

olddata = -1

while True:
    f.seek(0)
    data = f.read()[: -1]
    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 abd P8.25 or P9.27 and P9.42

rotaryEncoder.py

Listing 4.9: Reading a rotary encoder (rotaryEncoder.js)
rotaryEncoder.js

Try rotating the encoder clockwise and counter-clockwise. You’ll see an output like this:

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 10k 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:

```shell
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 4.10: Talking to a GPS with UART 4 (GPS.js)**

```javascript
#!/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');
var nmea = require('nmea');

var port = '/dev/ttyO4';
var options = {
  baudrate: 9600,
  parser: b.serialParsers.readline("\n")
};
```

(continues on next page)
```javascript
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`

```javascript
exports.serialParsers = m.module.parsers;
```

**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 I²C-based serial protocol.
To make this recipe, you will need:

- Breadboard and jumper wires.
- Two 4.7 k resistors.
- TMP101 temperature sensor.

Wire the TMP101, as shown in *Wiring an I2C TMP101 temperature sensor*.

There are two I2C buses brought out to the headers. *Table of I2C outputs* shows that you have wired your device to I2C bus 2.

Once the I2C device is wired up, you can use a couple handy I2C 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 I2C devices are on the bus. The -r flag indicates which bus to use. Our TMP101 is appearing at address 0x498. 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.

**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:  
  50:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 49  
  60:  
  70:  
```

(continues on next page)
2 I2C ports

Fig. 4.21: Wiring an I2C TMP101 temperature sensor

Fig. 4.22: Table of I2C outputs
Reading the temperature via the kernel driver

The cleanest way to read the temperature from a 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 busses (bus 0 is internal)
bone$ cd i2c-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--- 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--- 1 root gpio 4096 Dec 31 1999 uevent
```

Assuming the TMP101 is at address 0x48:

```
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 2-0048/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 from BoneScript.
Listing 4.11: Reading an I²C device (i2cTemp.py)

```python
#!/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-2ibus-00addr-00hwmon/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)
```

Listing 4.12: Reading an I²C device (i2cTemp.js)

```javascript
#!/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-2ibus-00addr-00hwmon/hwmon0';
// Read every ms
setInterval(readTMP, ms);

function readTMP() {
    var data = fs.readFileSync(I2CPATH + '/temp1_input').slice(0, -1);
    console.log('data (C) = ' + data/1000);
}
```

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
```

4.1. BeagleBone Cookbook 425
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 4.13: Reading an I²C device (i2cTemp.py)

```python
#!/usr/bin/env python

# // i2ctmp101.py
# // Read at TMP101 sensor on i2c bus 2, address 0x49
# // Wiring: Attach to i2c as shown in text.
# // Setup: pip install smbus
# // See:

import smbus
import time

ms = 1000  # Read time in ms
bus = smbus.SMBus(2)  # Using i2c bus 2
addr = 0x49  # TMP101 is at address 0x49

while True:
    data = bus.read_byte_data(addr, 0)
    print("temp (°C) = " + str(data))
    time.sleep(ms/1000)
```

This gets only 8 bits for the temperature. See the TMP101 datasheet 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 k resistor
- DS18B20 1-wire temperature sensor.

Wire up as shown in *Wiring a Dallas 1-Wire temperature sensor*.


Edit the file `/boot/utEnt.txt`. Go to about line 19 and edit as shown:
BeagleBoard Docs, Release 0.0.20230323

Fig. 4.23: Wiring a Dallas 1-Wire temperature sensor

```plaintext
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.js) 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.

Listing 4.14: Reading a temperature with a DS18B20 (w1.py)

```python
#!/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
```

(continues on next page)
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 4.15: Reading a temperature with a DS18B20 (w1.js)

#!/usr/bin/env node

const fs = require("fs");

const ms = 500  // Read time in ms

const addr = '28-00000d459c2c';  // Must be changed for your device.
const W1PATH = '/sys/bus/w1/devices/' + addr;

const ms = 500  // Read time in ms

function readW1() {
    var data = fs.readFileSync(W1PATH + '/temperature').slice(0, -1);
    console.log("temp (C) = " + data / 1000);
}

w1.js

bone$ ./w1.js

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.*
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**

```bash
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

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` to it.

4.1. BeagleBone Cookbook  429
Listing 4.16: Change the default audio out by putting this in ~/.asoundrc
(audio.asoundrc)

```plaintext
pcm.!default {
    type plug
    slave {
        pcm "hw:1,0"
    }
}
ctl.!default {
    type hw
    card 1
}
```

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.

4.1.3 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 www.github.com. 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).

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.js) in a file called internLED.js. 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).
Fig. 4.25: The P8 and P9 GPIO headers

Fig. 4.26: The four USER LEDs
Listing 4.17: Using an internal LED (internLED.py)

```python
#!/usr/bin/env python
# /////////////////////////////////////////////////////////////////////////////////
#                   internLED.py
# Blinks A USR LED.
# Wiring:
# Setup:
# See:
# /////////////////////////////////////////////////////////////////////////////////
import time
ms = 250  # Blink time in ms
LED = 'usr0';  # LED to blink
LEDPATH = '/sys/class/leds/beaglebone:green:LED+LED+/brightness'
state = '1'  # Initial state
f = open(LEDPATH, "w")
while True:
    f.seek(0)
    f.write(state)
    if (state == '1'):
        state = '0'
    else:
        state = '1'
        time.sleep(ms/1000)
```

Listing 4.18: Using an internal LED (internLED.js)

```javascript
#!/usr/bin/env node
// //________________________________________________________________________
//                   internalLED.js
// Blinks the USR LEDs.
// Wiring:
// Setup:
// See:
// //________________________________________________________________________
const fs = require('fs');
const ms = 250;  // Blink time in ms
const LED = 'usr0';  // LED to blink
const LEDPATH = '/sys/class/leds/beaglebone:green:LED+LED+/brightness';
var state = '1';  // Initial state
setInterval(flash, ms);  // Change state every ms

function flash() {
    fs.writeFileSync(LEDPATH, state)
    if(state == '1') {
        state = '0';
    } else {
        state = '1';
    }
}
```

In the **bash** command window, enter the following commands:
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.
- 220R to 470R 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 330R or 470R 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.

![Diagram for using an external LED](image)

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).
Listing 4.19: Code for using an external LED (externLED.py)

```python
#!/usr/bin/env python
# ////////////////////////////////////////
# // externalLED.py
# // 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.
# // Setup:
# // See:
# ////////////////////////////////////////
import time
import os
ms = 250  # Time to blink in ms
# Look up P9.14 using gpioinfo | grep -e chip -e P9.14. chip 1, line 18 maps to 50 pin = '50'
GPIOPATH = '/sys/class/gpio/
# 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 output pin
f = open(GPIOPATH + 'gpio' + pin + '/direction', "w")
    f.write("out")
    f.close()
    f = open(GPIOPATH + 'gpio' + pin + '/value', "w")
    # Blink
    while True:
        f.seek(0)
        f.write("1")
        time.sleep(ms/1000)
        f.seek(0)
        f.write("0")
        time.sleep(ms/1000)
    f.close()
```

Listing 4.20: Code for using an external LED (externLED.js)

```javascript
#!/usr/bin/env node
////////////////////////////////////////
// externalLED.js
// Blinks the P9_14 pin
// Wiring:
// Setup:
// See:
////////////////////////////////////////
const fs = require("fs");
// Look up P9.14 using gpioinfo | grep -e chip -e P9.14. chip 1, line 18 maps to...
pin="50";
```

(continues on next page)
externLED.js

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.

Fig. 4.28: Diagram for wiring PowerSwitch Tail II
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.

**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.

Listing 4.21: Code for using an external LED (fadeLED.py)

```python
#!/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')
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.js

The Bone has several outputs that can be used 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.

The pwm’s are accessed through `/dev/bone/pwm`

```
bone$ cd /dev/bone/pwm
bone$ ls
0 1 2
```

Here we see six pwm chips that can be used, each has two channels. Explore one.

```
bone$ cd 1
```
Fig. 4.29: Table of PWM outputs

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.

<table>
<thead>
<tr>
<th>Pin</th>
<th>PWM Channel Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>P9_31</td>
<td>0</td>
</tr>
<tr>
<td>P9_29</td>
<td>0</td>
</tr>
<tr>
<td>P9_14</td>
<td>1</td>
</tr>
<tr>
<td>P9_16</td>
<td>1</td>
</tr>
<tr>
<td>P8_19</td>
<td>2</td>
</tr>
<tr>
<td>P8_13</td>
<td>2</td>
</tr>
</tbody>
</table>

Headers to pwm channel mapping are the mapping I've figured out so far. I don't know how to get to the timers.
Writing to an LED Matrix

Problem  You have an I²C-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 R resistors.
• I²C 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.

Measuring a Temperature shows how to use `i2cdetect` to discover the address of an I²C device.

Run the `i2cdetect -y -r 2` command to discover the address of the display on I²C 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

```bash
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: -- -- -- -- -- -- -- -- -- -- -- --
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 0.70.

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)**

```python
include:../code/03displays/matrixLEDi2c.py
```

1. This line states which bus to use. The last digit gives the BoneScript 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 I²C 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. 4.31: Wiring a PCA9306 level translator to an LED matrix

Fig. 4.32: 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:

```bash
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 4.23: A program that talks (speak.js)

```javascript
#!/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.

4.1.4 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° (deg). 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 www.github.com. 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.

```bash
bone$ cd ~/beaglebone-cookbook-code/04motors
bone$ config-pin P9_16 pwm
bone$ ./servoMotor.py
```
Fig. 4.33: Driving a servo motor with the 3.3 V power supply

Listing 4.24: Code for driving a servo motor (servoMotor.py)

```python
#!/usr/bin/env python
# // //////////////////////////////////////////////////////////////////////
# // servoMotor.py
# // 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')
```

(continues on next page)
f.write(pwmPeriod)
f.close()
f = open(PWMPATH+’/enable’, ’w’)
f.write(’1’)
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 4.25: Code for driving a servo motor (servoMotor.js)
BeagleBoard Docs, Release 0.0.20230323

var dutyCycle = parseInt(pos * 1000000);  // Convert ms to ns
// console.log('pos = ' + pos + ' duty cycle = ' + dutyCycle);
fs.writeFileSync(PWMPATH + '/duty_cycle', dutyCycle);
}

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$ <strong>config-pin P9_16 pwm</strong>
bone$ <strong>config-pin P8_11 eqep</strong>
bone$ <strong>config-pin P8_12 eqep</strong>
bone$ <strong>./servoEncoder.py</strong>
```

Listing 4.26: Code for driving a servo motor with a rotary encoder(servoEncoder.py)

```python
#!/usr/bin/env python
# ///////////////////////////////////////
# // servoEncoder.py
# // Drive a simple servo motor using rotary encoder via 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
```
(continues on next page)
# Set the eEQP maximum count
fQEP = open(COUNTERPATH+"/ceiling", 'w')
fQEP.write(maxCount)
fQEP.close()

# Enable
fQEP = open(COUNTERPATH+"/enable", 'w')
fQEP.write('1')
fQEP.close()

fQEP = open(COUNTERPATH+"/count", 'r')

# Set up servo
pwmPeriod = '20000000'  # Period in ns, (20 ms)
pwm = '1'  # pwm to use
channel = 'b'  # channel to use
PWMPATH='/dev/bone/pwm/\pw]+
low = 0.6  # Smallest angle (in ms)
hi = 2.5  # 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()
s.exit(0)
signal.signal(signal.SIGINT, signal_handler)
f = open(PWMPATH+"/period", 'w')
f.write(pwmPeriod)
f.close()
f = open(PWMPATH+"/duty_cycle", 'w')
f.write(str(round(int(pwmPeriod)/2)))
f.close()
f = open(PWMPATH+"/enable", 'w')
f.write('1')
f.close()

print('Hit ^C to stop')

olddata = -1
while True:
    fQEP.seek(0)
    data = fQEP.read()[:1]
    if data != olddata:
        olddata = data
        # print("data = " + data)
        # map 0-180 to low-hi
        duty_cycle = -1*int(data)*(hi-low)/180.0 + hi
        duty_cycle = str(int(duty_cycle*1000000))  # Convert from ms...

        # print('duty_cycle = ' + duty_cycle)
f = open(PWMPATH+"/duty_cycle", 'w')
f.write(duty_cycle)
f.close()
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

<table>
<thead>
<tr>
<th>Pin</th>
<th>pwm</th>
<th>channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>P9_31</td>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>P9_29</td>
<td>0</td>
<td>b</td>
</tr>
<tr>
<td>P9_14</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>P9_16</td>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>P8_19</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>P8_13</td>
<td>2</td>
<td>b</td>
</tr>
</tbody>
</table>

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 ma) 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.js)* (dcMotor.js) to run the motor.

Listing 4.27: Driving a DC motor in one direction (dcMotor.py)
import time
import signal
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=1'+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()
```javascript
f = open(PWMPATH+'/duty_cycle', 'w')

while True:
    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)

f.close()

```

dcMotor.js

Listing 4.28: Driving a DC motor in one direction (dcMotor.js)

```javascript
#!/usr/bin/env node

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
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.

![Driving a DC motor with an H-bridge](image)

The code in Code for driving a DC motor with an H-bridge (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 4.29: Code for driving a DC motor with an H-bridge (h-bridgeMotor.js)

```javascript
#!/usr/bin/env node

// This example uses an H-bridge to drive a DC motor in two directions
```

(continues on next page)
```javascript
var b = require('bonescript');

var enable = 'P9_21'; // Pin to use for PWM speed control
var in1 = 'P9_15',
in2 = 'P9_16',
step = 0.05, // Change in speed
min = 0.05, // Min duty cycle
max = 1.0, // Max duty cycle
ms = 100, // Update time, in ms
speed = min; // Current speed;

b.pinMode(enable, b.ANALOG_OUTPUT, 6, 0, 0, doInterval);
b.pinMode(in1, b.OUTPUT);
b.pinMode(in2, b.OUTPUT);

function doInterval(x) {
  if(x.err) {
    console.log('x.err = ' + x.err);
    return;
  }
  timer = setInterval(sweep, ms);
}

clockwise(); // Start by going clockwise

function sweep() {
  speed += step;
  if(speed > max || speed < min) {
    step *= -1;
    step>0 ? clockwise() : counterClockwise();
  }
  b.analogWrite(enable, speed);
  console.log('speed = ' + speed);
}

function clockwise() {
  b.digitalWrite(in1, b.HIGH);
  b.digitalWrite(in2, b.LOW);
}

function counterClockwise() {
  b.digitalWrite(in1, b.LOW);
  b.digitalWrite(in2, b.HIGH);
}

process.on('SIGINT', function() {
  console.log('Got SIGINT, turning motor off');
  clearInterval(timer); // Stop the timer
  b.analogWrite(enable, 0); // Turn motor off
});
```

**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.

Use the code in Driving a bipolar stepper motor (bipolarStepperMotor.py) to drive the motor.

Listing 4.30: Driving a bipolar stepper motor (bipolarStepperMotor.py)

```python
#!/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]]
statesHalfStep = [[1,0,0,0], [1,1,0,0], [0,1,0,0], [0,1,1,0],
[0,0,1,0], [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 # current position and direction
direction = CW
GPIOPATH="/sys/class/gpio"
```

(continues on next page)
```python
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(str(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)
# Rotate

```python
while True:
    move()
    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.

![Unipolar stepper motor wiring](image)

**Fig. 4.37: 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.
Listings 4.31 and 4.32: Changes to bipolar code to drive a unipolar stepper motor

```python
# 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
```

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.

### 4.1.5 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.*

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:

```bash
### 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.

4.1. BeagleBone Cookbook 457
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 20.04 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 pass:[<span class="keep-together">computer</span>].

**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):

```bash
host$ ssh debian@192.168.7.2
Warning: Permanently added 'bone,192.168.7.2' (ECDSA) to the list of known hosts.
Last login: Mon Dec 22 07:53:06 2014 from yoder-linux.local
bone$
```

`debian` has the default password *tempped* It’s best to change the password:

```bash
bone$ passwd
Changing password for debian.
(new) UNIX password: 
Enter new UNIX password:
Retype new UNIX password: 
passwd: password updated successfully
```
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.)

First, check to ensure that the serial port is there. On the host computer, run the following command:

```bash
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:

```bash
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:

```bash
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:

```bash
host$ sudo apt install screen
host$ screen /dev/ttyACM0 115200
Debian GNU/Linux 7 beaglebone ttyGS0
default username:password is [debian:temppwd]
```


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 tty00

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:
Fig. 4.40: FTDI connector

Fig. 4.41: FTDI pins for the FTDI connector
Note: Your screen might initially be blank. Press Enter a couple times to see the login prompt.

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 2022-07-01
```

Verifying You Have the Latest Version of the OS on Your Bone shows how to open the ID.txt file to see the OS version. The /etc/dogtag file has the same contents and is easier to find if you already have a command prompt. 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 Run the installed Virtual Network Computing (VNC) server:

```
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.

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.
Fig. 4.42: Creating a new remote desktop file in Remmina Remote Desktop Client

Fig. 4.43: Configuring the Remmina Remote Desktop Client
Fig. 4.44: The Remmina Remote Desktop Client showing the BeagleBone desktop
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>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwd</td>
<td>show current directory</td>
</tr>
<tr>
<td>cd</td>
<td>change current directory</td>
</tr>
<tr>
<td>ls</td>
<td>list directory contents</td>
</tr>
<tr>
<td>chmod</td>
<td>change file permissions</td>
</tr>
<tr>
<td>chown</td>
<td>change file ownership</td>
</tr>
<tr>
<td>cp</td>
<td>copy files</td>
</tr>
<tr>
<td>mv</td>
<td>move files</td>
</tr>
<tr>
<td>rm</td>
<td>remove files</td>
</tr>
<tr>
<td>mkdir</td>
<td>make directory</td>
</tr>
<tr>
<td>rmdir</td>
<td>remove directory</td>
</tr>
<tr>
<td>cat</td>
<td>dump file contents</td>
</tr>
<tr>
<td>less</td>
<td>progressively dump file</td>
</tr>
<tr>
<td>vi</td>
<td>edit file (complex)</td>
</tr>
<tr>
<td>nano</td>
<td>edit file (simple)</td>
</tr>
<tr>
<td>head</td>
<td>trim dump to top</td>
</tr>
<tr>
<td>tail</td>
<td>trim dump to bottom</td>
</tr>
<tr>
<td>echo</td>
<td>print/dump value</td>
</tr>
<tr>
<td>env</td>
<td>dump environment variables</td>
</tr>
<tr>
<td>export</td>
<td>set environment variable</td>
</tr>
<tr>
<td>history</td>
<td>dump command history</td>
</tr>
<tr>
<td>grep</td>
<td>search dump for strings</td>
</tr>
<tr>
<td>man</td>
<td>get help on command</td>
</tr>
<tr>
<td>apropos</td>
<td>show list of man pages</td>
</tr>
<tr>
<td>find</td>
<td>search for files</td>
</tr>
<tr>
<td>tar</td>
<td>create/extract file archives</td>
</tr>
<tr>
<td>gzip</td>
<td>compress a file</td>
</tr>
<tr>
<td>gunzip</td>
<td>decompress a file</td>
</tr>
<tr>
<td>du</td>
<td>show disk usage</td>
</tr>
<tr>
<td>df</td>
<td>show disk free space</td>
</tr>
<tr>
<td>mount</td>
<td>mount disks</td>
</tr>
<tr>
<td>tee</td>
<td>write dump to file in parallel</td>
</tr>
<tr>
<td>hexdump</td>
<td>readable binary dumps</td>
</tr>
<tr>
<td>whereis</td>
<td>locates binary and source files</td>
</tr>
</tbody>
</table>

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.

**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.

**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:
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 e8: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 [debian has the default password tempped It’s best to change the password](https://forum.beagleboard.org/t/8847) and [Setting Up a Firewall](https://www.debian.org/doc/faq/security/ch02s02.html).

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](https://www.beagleboard.org/images/) and click on the image you are using.

I’m running Debian 11.x (Bullseye), the middle one.

![Latested BeagleImages](https://forum.beagleboard.org/t/8847)

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](https://www.beagleboard.org/wifi-adapters) for the latest list.
Fig. 4.48: Instructions for setting up your network.
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`.

```
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
```

(continues on next page)
Next edit the configuration file */etc/wpa_supplicant/wpa_supplicant-wlan0.conf*.

```
bones$ sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

In the file you’ll see:

```bash
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:

```
bones$ sudo systemctl restart systemd-networkd
bones$ ip a
bones$ 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
```

```
--- 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 4.33: Code for IP Masquerading (ipMasquerade.sh)**

```bash
#!/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.
```

(continues on next page)
# 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 4.34: Code for setting the DNS on the Bone (setDNS.sh)
# 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

```
if [ $? -ne 0 ]; then
    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
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$ ifconfig
eth0 Link encap:Ethernet  HWaddr 00:e0:4e:00:22:51
    inet addr:137.112.41.35  Bcast:137.112.41.255  Mask:255.255.255.0
    inet6 addr: fe80::2e0:4eff:fe00:2251/64 Scope:Link
    UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
    RX packets:5371019 errors:0 dropped:0 overruns:0 frame:0
    TX packets:4720856 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1000
    RX bytes:1667916614 (1.6 GB)  TX bytes:597909671 (597.9 MB)
eth1 Link encap:Ethernet  HWaddr 00:1d:60:40:58:e6 ...
```

It's the number following `inet addr:`, which in my case is 137.112.41.35.
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 137.112.41.35 --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](https://www.howtogeek.com/126379/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 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
(continues on next page)
BeagleBoard Docs, Release 0.0.20230323

(continued from previous page)

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
  67303   67303 1348342 /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` 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:
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:

```bash
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:

```bash
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
0: 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 $K$ postixes 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 /usr/*
4.0K /var/backups
76M /var/cache
93M /var/lib
4.0K /var/local
0 /var/lock
751M /var/log
4.0K /var/mail
0 /var/opt
16K /var/spool
93M /var/swap
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
e . 0.0 B [ ] /sys
e . 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 4.35: Use C to blink an LED (blinkLED.c)

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define MAXSTR 100

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.

<table>
<thead>
<tr>
<th>P9</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGND</td>
<td>DGND</td>
</tr>
<tr>
<td>VDD_3V3</td>
<td>VDD_3V3</td>
</tr>
<tr>
<td>VDD_5V</td>
<td>VDD_5V</td>
</tr>
<tr>
<td>SYS_5V</td>
<td>SYS_5V</td>
</tr>
<tr>
<td>PWR_BUT</td>
<td>PWR_BUT</td>
</tr>
<tr>
<td>GPIO_30</td>
<td>GPIO_60</td>
</tr>
<tr>
<td>GPIO_31</td>
<td>GPIO_50</td>
</tr>
<tr>
<td>GPIO_48</td>
<td>GPIO_51</td>
</tr>
<tr>
<td>GPIO_5</td>
<td>GPIO_4</td>
</tr>
<tr>
<td>I2C_SCL</td>
<td>I2C_SCL</td>
</tr>
<tr>
<td>GPIO_3</td>
<td>GPIO_2</td>
</tr>
<tr>
<td>GPIO_49</td>
<td>GPIO_15</td>
</tr>
<tr>
<td>GPIO_117</td>
<td>GPIO_14</td>
</tr>
<tr>
<td>GPIO_115</td>
<td>GPIO_113</td>
</tr>
<tr>
<td>GPIO_111</td>
<td>GPIO_112</td>
</tr>
<tr>
<td>GPIO_110</td>
<td>VDD_ADC</td>
</tr>
<tr>
<td>AIN4</td>
<td>GNDA_ADC</td>
</tr>
<tr>
<td>AIN6</td>
<td>AIN5</td>
</tr>
<tr>
<td>AIN2</td>
<td>AIN3</td>
</tr>
<tr>
<td>AIN0</td>
<td>AIN1</td>
</tr>
<tr>
<td>GPIO_20</td>
<td>GPIO_7</td>
</tr>
<tr>
<td>DGND</td>
<td>DGND</td>
</tr>
<tr>
<td>DGND</td>
<td>DGND</td>
</tr>
</tbody>
</table>

Fig. 4.49: Mapping from header pin to internal GPIO number

Compile and run the code:

```
bone$ gcc -o blinkLED blinkLED.c
bone$ ./blinkLED
^C
```

Hit ^C to stop the blinking.

### 4.1.6 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).

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`):

```bash
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`:

```bash
host$ sudo apt install sshfs
```

and then access:

```bash
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:

```bash
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 4.36: A sample web page (test.html)

```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`.
Fig. 4.50: 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 is in the Cookbook repo:

```
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**

```
#!/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`:
Fig. 4.51: Test page served by our custom flask server
Listing 4.38: index1.html

```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:

Listing 4.39: app1.py

```python
#!/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/index...
'''

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 index.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” change it every second.

4.1. BeagleBone Cookbook
Hello, World!

The date and time on the server is: 2022-07-12 14:32

Fig. 4.52: Test page served by app1.py
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*.

Listing 4.40: A simple Flask-based web server to read a GPIO (app2.py)

```python
#!/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*

Look that 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:

4.1. BeagleBone Cookbook
Listing 4.41: A simple Flask-based web server to read a GPIO (index2.html)

```html
<!DOCTYPE html>
<head>
<title>{{ title }}</title>
<link rel="stylesheet" href='../static/style.css'/>
</head>
<body>
<h1>{{ title }}</h1>
<h2>Button pressed: {{ button }}</h2>
</body>
</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.

![GPIO input Status!](image)

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.

The python script Let’s create a new Python script and named it app3.py.

A simple Flask-based web server to read a GPIO (app3.py)

include::../code/06iot/flask/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: GPIO.output(actuator, GPIO.HIGH)

• Update the status of each actuator

• Update the variable library

• Return the data to index.html

Let’s now create an index.html to show the GPIO status of each actuator and more important, create “buttons” to send the commands:

Listing 4.42: A simple Flask-based web server to write a GPIO (index3.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.

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.
Here you see the files used to read the one shot values. Look in `scan_elements` to see how to enable continuous input.

```
bones ls scan_elements
```

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:

```
bones echo 1 > scan_elements/in_voltage1_en
```

Next set the buffer size.

```
bones ls buffer
```

Let’s use a 512 sample buffer. You might need to experiment with this.

```
bones echo 512 > buffer/length
```

Then start it running.

```
bones echo 1 > buffer/enable
```

Now, just read from `*/dev/iio:device0*`.

An example Python program that does the above and the reads and plot the buffer is here: `analogInContinuous.py`

```
# analogInContinuous.py
# Read analog data via IIO continuous mode and plots it.
# From: https://stackoverflow.com/questions/20295646/python-ascii-plots-in-terminal
# https://github.com/dkogan/gnuplotlib
# https://github.com/dkogan/gnuplotlib/blob/master/guide/guide.org
# sudo apt install gnuplot (10 minute to install)
# sudo apt install libatlas-base-dev
# pip3 install gnuplotlib
# This uses X11, so when connecting to the bone from the host use: ssh -X bone
# See https://elinux.org/index.php?title=EBC_Exercise_10a_Analog_In#Analog_in_-_Continuous.2C_Change_the_sampling_rate
# for instructions on changing the sampling rate. Can go up to 200KHz.
```

(continues on next page)
Fig. 4.54: 1KHz sine wave sampled at 8KHz
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 Continuous 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
    file1 = open(IIOPATH + '/buffer/enable', 'w')
    file1.write('0')

(continues on next page)
analogInContinuous.py

Be sure to read the installation 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**

```
// For each step, number of adc clock cycles to wait between setting up...
//muxes and sampling.
// range: 0 .. 262143
// optional, default is 152 (XXX but why?)
ti,chan-step-opendelay = <152 152 152 152 152 152 152>
```

```
// For each step, how many times it should sample to average.
```

(continues on next page)
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 get as fast as a possible.

```c
// optional, default is 16
ti,chan-step-avg = <16 16 16 16 16 16 16 16>
```

Now compile it.

```bash
bone$ make 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.

```bash
bone$ sudo make install ...
'src/arm/AM335X-PRU-UlO-00A0.dtbo' -> '/lib/firmware/AM335X-PRU-UlO-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.

```bash
bone$ cd beaglebone-cookbook-code/06iot
bone$ ./analogInContinuous.py
Hit ^C to stop
```

Here’s the output of a 10KHz sine wave.

It’s still a good idea to disable the buffer when done.

```bash
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 (emailTest.py) to a file named emailTest.py. Substitute your own Gmail username. For the password:

- Go to: https://myaccount.google.com/security
- 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.

**Listing 4.44: Sending email using nodemailer (emailTest.py)**

```python
#!/usr/bin/env python
# From: https://realpython.com/python-send-email/
import smtplib, ssl
```
Fig. 4.55: 10KHz triangle wave sampled at 200KHz
BeagleBoard Docs, Release 0.0.20230323

(continued from previous page)

4.1. BeagleBone Cookbook

```python
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:

```bash
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/](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:

```bash
bone$ npm install -g twilio
```

Finally, add the code in *Sending SMS messages using Twilio (twilio-test.js)* to a file named *twilio-test.js* and run it. Your text will be sent.

**Listing 4.45: Sending SMS messages using Twilio (twilio-test.js)**

```javascript
#!/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: '+4260551212',
  from: '+2605551212',
  body: 'This is a test',
}, function(err, message) {
  console.log(message.sid);
});

The MIT License (MIT)
Copyright (c) 2010 Stephen Walters
Copyright (c) 2012 Twilio Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.

twilio-test.js nodemailer-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.js.
- Run the python script.
Listing 4.46: 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']) # ☀
        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]) # ☀
        # print("weather: ", weather) # ☀
        # print("icon: ", weather['current']['weather'][0]['icon'])
        # print()  
    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.

Run this by using the following commands:

```
bone$ chmod +x weather.py
bone$ ./weather.js
Getting weather
```

(continues on next page)
Temp: 85.1
Humid: 50
Low: 62.02
High: 85.1
sunrise: 2022-07-14 14:32:46

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 an app.
- Be sure to give your app Read and Write permission.
- Then go to the developer portal and select your 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.

```bash
export API_KEY='XXX'
export API_SECRET_KEY='XXX'
export BEARER_TOKEN='XXX'
export TOKEN='4XXX'
export TOKEN_SECRET='XXX'
```

- Next, source the file so the values will appear in your bash session.

```bash
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 4.47: Create a Tweet (twitter_create_tweet.py)**

```python
#!/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...
```
8  # export 'API_KEY'='<your_consumer_key>'
9  # export 'API_SECRET_KEY'='<your_consumer_secret>'
10
11 consumer_key = os.environ.get("API_KEY")
12 consumer_secret = os.environ.get("API_SECRET_KEY")
13
14 # Be sure to add replace the text of the with the text you wish to Tweet. You can...
15 # also add parameters to post polls, quote Tweets, Tweet with reply settings, and...
16 # Tweet to Super Followers in addition to other features.
17 payload = {"text": "Hello world!
18
19 # Get request token
20 request_token_url = "https://api.twitter.com/oauth/request_token?oauth__
21 --callback=oob&x_auth_access_type=write"
22 oauth = OAuth1Session(consumer_key, client_secret=consumer_secret)
23 try:
24     fetch_response = oauth.fetch_request_token(request_token_url)
25 except ValueError:
26     print("There may have been an issue with the consumer_key or consumer_secret you...
27 # Get authorization
28 base_authorization_url = "https://api.twitter.com/oauth/authorize"
29 authorization_url = oauth.authorization_url(base_authorization_url)
30     print("Please go here and authorize: %s" % authorization_url)
31     verifier = input("Paste the PIN here: ")
32
33 # Get the access token
34 access_token_url = "https://api.twitter.com/oauth/access_token"
35 oauth = OAuth1Session(
36     consumer_key,
37     client_secret=consumer_secret,
38     resource_owner_key=resource_owner_key,
39     resource_owner_secret=resource_owner_secret,
40     verifier=verifier,
41 )
42 oauth_tokens = oauth.fetch_access_token(access_token_url)
43 access_token = oauth_tokens["oauth_token"]
44 access_token_secret = oauth_tokens["oauth_token_secret"]
45
46 # Make the request
47 oauth = OAuth1Session(
48     consumer_key,
49     client_secret=consumer_secret,
50     resource_owner_key=access_token,
51     resource_owner_secret=access_token_secret,
52 )
53
54 # Making the request
55 response = oauth.post(  
56         "https://api.twitter.com/2/tweets",
57         json=payload,
if response.status_code != 201:
    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.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: tWBldQAolAAWJgjAAABggJt7qq
Please go here and authorize: https://api.twitter.com/oauth/authorize?oauth_token=tWBldQAolAAWJgjAAABggJt7qq
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 4.48: 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,
    client_secret=consumer_secret,
    resource_owner_key=access_token,
    resource_owner_secret=access_token_secret,
)

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 4.49: 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);
      }
    });

  // 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 iStrategyLabs node-twitter GitHub page.

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.
Installing Node-RED

To install Node-RED, run the following commands:

```bash
bone$ cd          # Change to home directory
bone$ git clone https://github.com/node-red/node-red.git
bone$ cd node-red/
bone$ npm install --production         # almost 6 minutes
bone$ cd nodes
bone$ git clone https://github.com/node-red/node-red-nodes.git # 2 seconds
```

To run Node-RED, use the following commands:

```bash
bone$ cd ~/node-red
bone$ node red.js
Welcome to Node-RED
```

- 18 Aug 16:31:43 - [red] Loading palette nodes
- 18 Aug 16:31:56 - ———————————————————
- 18 Aug 16:31:56 - [red] Failed to register 44 node types
- 18 Aug 16:31:56 - [red] Run with -v for details
- 18 Aug 16:31:56 - ———————————————————

The second-to-last line informs you that Node-RED is listening on port 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 canvas- (*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. 4.56: The Node-RED web page
Fig. 4.57: Node-RED twitter node
Fig. 4.58: Node-RED Twitter authorization, step 1
Fig. 4.59: Node-RED twitter authorization, step 2
Authorize Node RED to use your account?

This application will be able to:
- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages.

This application will not be able to:
- See your Twitter password.

You can revoke access to any application at any time from the Applications tab of your Settings page.
By authorizing an application you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.

Fig. 4.60: Node-RED Twitter site authorization
Fig. 4.61: Node-RED adding the #BeagleBone hashtag
Fig. 4.62: 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).

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. 4.64: 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.

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 4.50: LaunchPad code for communicating via the UART (launchPad.ino)

```c
// 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)
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 4.51: 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) {
    console.log(x.event); // ⑤
    if (x.err) {
        console.log('***ERROR*** ' + JSON.stringify(x));
    }
    if (x.event == 'open') {
        // (continues on next page)
```
```
20  console.log('***OPENED***');
21  setInterval(sendCommand, 1000);  // ⑥
22  }
23  if (x.event == 'data') {
24     console.log(String(x.data));
25  }
26  }
27
28  var command = ['r', 'g'];  // ⑦
29  var commIdx = 1;
30
31  function sendCommand() {
32      // console.log('Command: ' + command[commIdx]);
33      b.serialWrite(port, command[commIdx++]); // ⑧
34      if(commIdx >= command.length) {  // ⑨
35          commIdx = 0;
36      }
37  }
```

**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 baud rate 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 and, when it arrives, responds by toggling the corresponding LED.

### 4.1.7 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.

**Note:** We assume here that you are logged on to your Bone as root and superuser privileges. You also need to be logged in to your Linux host computer as a nonsuperuser.
Updating the Kernel

**Problem**  You have an out-of-date kernel and want to want to make it current.

**Solution**  Use the following command to determine which kernel you are running:

```
bone$ uname -a
Linux beaglebone 3.8.13-bone67 #1 SMP Wed Sep 24 21:30:03 UTC 2014 armv7l
GNU/Linux
```

The `3.8.13-bone67` 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-3.15.8-armv7-x5
linux-image-3.15.8-bone5
linux-image-3.15.8-bone6

... linux-image-3.16.0-rc7-bone1
...
linux-image-3.8.13-bone60
linux-image-3.8.13-bone61
linux-image-3.8.13-bone62
bone$ sudo apt install linux-image-3.14.23-ti-r35
bone$ sudo reboot
bone$ uname -a
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.

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`.

```
#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...
    printk(KERN_INFO "Hello, World!
    return 0;
}

static void __exit hello_end(void)
{
    printk(KERN_INFO "Goodbye Boris"
}

module_init(hello_start);
module_exit(hello_end);

MODULE_AUTHOR("Boris Houndleroy");
MODULE_DESCRIPTION("Hello World Example");
MODULE_LICENSE("GPL");
```

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.

Next, add the code in *Simple Kernel Module (Makefile)* to a file called `Makefile`. 
Listing 4.53: Simple Kernel Module (Makefile)

1. obj-m := hello.o
2. KDIR := /lib/modules/$(shell uname -r)/build
3. all:
4. <TAB>make -C $(KDIR) M=\$\$PWD
5. clean:
6. <TAB>rm hello.mod.c hello.o modules.order hello.mod.o Module.symvers

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/3.8.13-bone67/build \
   SUBDIRS=/home/debian/beaglebone-cookbook-code/07kernel/hello modules
make[1]: Entering directory `/usr/src/linux-Headers-3.8.13-bone67'
CC [M] /home/debian/beaglebone-cookbook-code/07kernel/hello/hello.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/debian/beaglebone-cookbook-code/07kernel/hello/hello.mod.o
LD [M] /home/debian/beaglebone-cookbook-code/07kernel/hello/hello.ko
make[1]: Leaving directory `/usr/src/linux-Headers-3.8.13-bone67'
bone$ ls
Makefile hello.c hello.mod.c hello.o Module.symvers

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/beaglebone-cookbook-code/07kernel/hello/hello.ko
srcversion: 87C6AEED7791B4B90C3B50C
depends: vermagic: 3.8.13-bone67 SMP mod_unload modversions ARMv7 thumb2 p2v8
bone$ sudo insmod hello.ko
bone$ dmesg | tail -4
[419313.320052] bone-iio-helper helper.15: ready
[419313.322776] bone-capemgr bone_capemgr.9: slot #8: Applied #1 overlays.
[491540.999431] Loading hello module...
[491540.999476] 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!

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:
What you are seeing are four directories, one for each onboard LED. Now try this:

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

The first command changes into the directory for LED \textit{usr0}, which is the LED closest to the edge of the board. The \textit{[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 \textit{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** \textit{Controlling LEDs by Using SYSFS Entries} introduces the \textit{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 \textit{P9_42} GPIO pin. \textit{(Reading the Status of a Pushbutton or Magnetic Switch (Passive OnOff Sensor)} shows how to wire a switch to \textit{P9_42}. First, you need to map the \textit{P9} header location to GPIO number using \textit{Mapping \textit{P9_42} header position to GPIO 7}, which shows that \textit{P9_42} maps to GPIO 7.

Next, change to the GPIO \textit{sysfs} directory:

```
bone$ cd /sys/class/gpio/
bone$ ls
export gpiochip0 gpiochip32 gpiochip64 gpiochip96 unexport
```

The \textit{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 \textit{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:
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.

### 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://github.com/RobertCNelson/bb-kernel.git
host$ cd bb-kernel
host$ git tag
host$ git checkout 3.8.13-bone60 # v3.8.13-bone60
host$ ./build_kernel.sh
```

1. The first command clones a repository with the tools to build the kernel for the Bone.
2. This command lists all the different versions of the kernel that you can build. You’ll need to pick one of these. How do you know which one to pick? A good first step is to choose the one you are currently running. `uname -a` will reveal which one that is. When you are able to reproduce the current kernel, go to Linux Kernel Newbies to see what features are available in other kernels. LinuxChanges shows the features in the newest kernel and LinuxVersions links to features of previous kernels.
3. When you know which kernel to try, use `git checkout` to check it out. This command checks out at tag 3.8.13-bone60 and creates a new branch, v3.8.13-bone60.
4. `build_kernel` is the master builder. If needed, it will download the cross compilers needed to compile the kernel (linaro is the current cross compiler). If there is a kernel at ~/linux-dev, it will use it; otherwise, it will download a copy to bb-kernel/ignore/linux-src. It will then patch the kernel so that it will run on the Bone.

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.
Fig. 4.68: Kernel configuration menu
The `bb-kernel/KERNEL` directory contains the source code for the kernel. The `bb-kernel/deploy` directory contains the compiled kernel and the files needed to run it.

### Installing the Kernel on the Bone

To copy the new kernel and all its files to the microSD card, you need to halt the Bone, and then pull the microSD card out and put it in an microSD card reader on your host computer. Run Disk (see [Verifying You Have the Latest Version of the OS on Your Bone](https://beagleboard.org/docs/0.0.20230323)) to learn where the microSD card appears on your host (mine appears in `/dev/sdb`). Then open the `bb-kernel/system.sh` file and find this line near the end:

```
MMC=/dev/sde
```

Change that line to look like this (where `/dev/sdb` is the path to your device):

```
MMC=/dev/sdb
```

Now, while in the `bb-kernel` directory, run the following command:

```bash
host$ tools/install_kernel.sh
```

I see...

```
I see... 
```

```
fdisk -l:
```

```
Disk /dev/sda: 160.0 GB, 160041885696 bytes
Disk /dev/sdb: 3951 MB, 3951034368 bytes
Disk /dev/sdc: 100 MB, 100663296 bytes
```

```
lsblk:
```

```
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 0 149.1G 0 disk
├─sda1 8:1 0 141.1G 0 part /
├─sda2 8:2 0 1K 0 part
└─sda5 8:5 0 8G 0 part [SWAP]
```

```
sdb 8:16 1 3.7G 0 disk
├─sdb1 8:17 1 16M 0 part
└─sdb2 8:18 1 3.7G 0 part
```

```
sdc 8:32 1 96M 0 disk
```

```
Are you 100% sure, on selecting [/dev/sdb] (y/n)? y
```

The script lists the partitions it sees and asks if you have the correct one. If you are sure, press Y, and the script will uncompress and copy the files to the correct locations on your card. When this is finished, eject your card, plug it into the Bone, and boot it up. Run `uname -a`, and you will see that you are running your compiled kernel.

### Using the Installed Cross Compiler

**Problem** You have followed the instructions in [Compiling the Kernel](https://beagleboard.org/docs/0.0.20230323) and want to use the cross compiler it has downloaded.

**Tip** You can cross-compile without installing the entire kernel source by running the following:

```bash
host$ sudo apt install gcc-arm-linux-gnueabihf
```

Then skip down to [Setting Up Variables](https://beagleboard.org/docs/0.0.20230323).

**Solution** Compiling the Kernel installs a cross compiler, but you need to set up a couple of things so that it can be found. Compiling the Kernel installed the kernel and other tools in a directory called `bb-kernel`. Run the following commands to find the path to the cross compiler:
host$ cd bb-kernel/dl
host$ ls
gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux
gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux.tar.xz

Here, the path to the cross compiler contains the version number of the compiler. Yours might be different from mine. cd into it:

host$ cd gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux
host$ ls
20130415-gcc-linaro-arm-linux-gnueabihf bin libexec
arm-linux-gnueabihf lib share

At this point, we are interested in what’s in bin:

host$ cd bin
host$ ls
arm-linux-gnueabihf-addr2line arm-linux-gnueabihf-gfortran
arm-linux-gnueabihf-ar arm-linux-gnueabihf-gprof
arm-linux-gnueabihf-as arm-linux-gnueabihf-ld
arm-linux-gnueabihf-c++ arm-linux-gnueabihf-ld.bfd
arm-linux-gnueabihf-c++filt arm-linux-gnueabihf-ldd
arm-linux-gnueabihf-cpp arm-linux-gnueabihf-ld.gold
arm-linux-gnueabihf-ct-ng.config arm-linux-gnueabihf-nm
arm-linux-gnueabihf-elfedit arm-linux-gnueabihf-objcopy
arm-linux-gnueabihf-g* arm-linux-gnueabihf-objdump
arm-linux-gnueabihf-gcc arm-linux-gnueabihf-pkg-config
arm-linux-gnueabihf-gcc-4.7.3 arm-linux-gnueabihf-pkg-config-real
arm-linux-gnueabihf-gcc-ar arm-linux-gnueabihf-ranlib
arm-linux-gnueabihf-gcc-nm arm-linux-gnueabihf-readelf
arm-linux-gnueabihf-gcc-ranlib arm-linux-gnueabihf-size
arm-linux-gnueabihf-gcov arm-linux-gnueabihf-strings
arm-linux-gnueabihf-gdb arm-linux-gnueabihf-strip

What you see are all the cross-development tools. You need to add this directory to the $PATH the shell uses to find the commands it runs:

host$ pwd
/home/yoder/BeagleBoard/bb-kernel/dl/
gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux/bin

host$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games

The first command displays the path to the directory where the cross-development tools are located. The second shows which directories are searched to find commands to be run. Currently, the cross-development tools are not in the $PATH. Let’s add it:

host$ export PATH=`pwd`:$PATH
host$ echo $PATH
/home/yoder/BeagleBoard/bb-kernel/dl/
gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux/bin:
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games

Note: Those are backtick characters (left of the “1” key on your keyboard) around pwd.

The second line shows the $PATH now contains the directory with the cross-development tools.
Setting Up Variables

Now, setup a couple of variables to know which compiler you are using:

```bash
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 4.54: Simple helloWorld.c to test cross compiling (helloWorld.c)

```c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello, World! \n");
}
```

You can then cross-compile by using the following commands:

```bash
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]=0x10182364352b9f3cb15d1aa61395aede11a52ad, 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 4.55: Simple kernel patch file (hello.patch)

```diff
--- 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)
+KDIR := $(PWD)/..
```

(continues on next page)
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 bb-kernel/KERNEL+).
- Add Simple kernel patch file (hello.patch) to a file named hello.patch in the bb-kernel/KERNEL directory.
- Run the following commands:

  host$ cd bb-kernel/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 bb-kernel/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 bb-kernel/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 > hello1.patch
```
4.1.8 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 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.js) 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.

Listing 4.56: Monitoring a pushbutton (pushLED.py)

```python
#!/usr/bin/env python
# // pushLED.py
# // Blinks an LED attached to P9_12 when the button at P9_42 is pressed
# // Wiring:
# // Setup:
# // 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")
f.close()

# Read every ms
fin = open(GPIOPATH+"gpio"+button+"/value", "r")
fout = open(GPIOPATH+"gpio"+LED+"/value", "w")

(continues on next page)
```python
while True:
    fin.seek(0)
    fout.seek(0)
    fout.write(fin.read())
    time.sleep(ms/1000)
```

Listing 4.57: Monitoring a pushbutton (pushLED.js)

```javascript
#!/usr/bin/env node

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 maps to 7
GPIOPATH="/sys/class/gpio/";

// Make sure LED is exported
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);
}
```

Add the code to a file named `pushLED.js` and run it by using the following commands:

```
bone$ chmod +x pushLED.js
bone$ ./pushLED.js
```

(continues on next page)
Press ^C (Ctrl-C) to stop the code.

**I/O with C**

**Problem** You want to use the C language to process inputs in real time, or Python/JavaScript isn’t fast enough.

**Solution** *I/O with JavaScript* shows how to control an LED with a pushbutton using JavaScript. This recipe accomplishes the same thing using C. It does it in the same way, opening the correct /sys/class/gpio files and reading an writing them.

Wire up the pushbutton and LED as shown in *Diagram for wiring a pushbutton and LED with the LED attached to P9_14*. Then add the code in *Code for reading a switch and blinking an LED (pushLED.c)* to a file named `pushLED.c`.

Listing 4.58: Code for reading a switch and blinking an LED (pushLED.c)

```c
#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, GPIOPATH, "gpio", LED);
    if (!access(path, F_OK) == 0) {
        snprintf(path, MAXSTR, GPIOPATH, "export");
        fpLED = fopen(path, "w");
        fprintf(fpLED, "%s", LED);
        fclose(fpLED);
    }

    // Make sure button is exported
    snprintf(path, MAXSTR, GPIOPATH, "gpio", button);
    if (!access(path, F_OK) == 0) {
        snprintf(path, MAXSTR, GPIOPATH, "export");
        fpbutton = fopen(path, "w");
        fprintf(fpbutton, "%s", button);
        fclose(fpbutton);
    }

    // Wiring:
    // Setup:
    // See:

    // Make it an output LED
    snprintf(path, MAXSTR, GPIOPATH, "direction");
    fpLED = fopen(path, "w");
    fprintf(fpLED, "out");
    fclose(fpLED);

    // Make it an output LED
    snprintf(path, MAXSTR, GPIOPATH, "direction");
    fpLED = fopen(path, "w");
    fprintf(fpLED, "out");
    fclose(fpLED);

    // Make sure button is exported
    snprintf(path, MAXSTR, GPIOPATH, "gpio", button);
    if (!access(path, F_OK) == 0) {
        snprintf(path, MAXSTR, GPIOPATH, "export");
        fpbutton = fopen(path, "w");
        fprintf(fpbutton, "%s", button);
        fclose(fpbutton);
    }
}
```
BeagleBoard Docs, Release 0.0.20230323

```c
fprintf(fpbutton, "%s", button);
fclose(fpbutton);
}

// 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
}
```

```bash
pushLED.c

Compile and run the code:

```
bone$ gcc -o pushLED pushLED.c
bone$ ./pushLED
state: 1
state: 1
state: 0
state: 0
state: 0
state: 1
```

The code responds quickly to the pushbutton. If you need more speed, comment-out the `printf()` and the `sleep()`.

**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`:

```
```
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.

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 GPIO31, export GPIO 31 and make it an output:

```
bone$ 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
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 to 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 4.59: Memory address definitions (pushLEDmmap.h)
```

```
#ifndef _BEAGLEBONE_GPIO_H_
#define _BEAGLEBONE_GPIO_H_

// 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)

#undef _BEAGLEBONE_GPIO_H_
#define _BEAGLEBONE_GPIO_H_
```

(continues on next page)
#define GPIO0_START_ADDR 0x44e07000
#define GPIO0_END_ADDR 0x44e08000
#define GPIO0_SIZE (GPIO0_END_ADDR - GPIO0_START_ADDR)

#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
#endif

pushLEDmmap.h
Add the code in Code for directly reading memory addresses (pushLEDmmap.c) to a file named pushLEDmmap.c.

Listing 4.60: Code for directly reading memory addresses (pushLEDmmap.c)

```c
#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;

// Set the signal callback for Ctrl-C
signal(SIGINT, signal_handler);

int fd = open("/dev/mem", O_RDWR);

printf("Mapping $X - $X (size: $X)\n", GPIO0_START_ADDR, GPIO0_END_ADDR, GPIO0_SIZE);
gpio_addr = mmap(0, GPIO0_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, GPIO0_START_ADDR);

gpio_datain = gpio_addr + GPIO_DATAIN;
gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;

if(gpio_addr == MAP_FAILED) {
    printf("Unable to map GPIO\n");
    exit(1);
}

printf("GPIO mapped to %p\n", gpio_addr);
printf("GPIO SETDATAOUT ADDR mapped to %p\n", gpio_setdataout_addr);
printf("GPIO CLEARDATAOUT mapped to %p\n", gpio_cleardataout_addr);

printf("Start copying GPIO_07 to GPIO_31\n");
while(keepgoing) {
    if(!gpio_datain & GPIO_07) {
        *gpio_setdataout_addr = GPIO_31;
    } else {
        *gpio_cleardataout_addr = GPIO_31;
    }
    //usleep(1);
}

munmap((void *)gpio_addr, GPIO0_SIZE);
close(fd);
return 0;
}

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 SETDATAOUT ADDR 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:

  ```bash
  bone$ uname -a
  Linux breadboard-home 5.10.120-ti-r47 #bullseye 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:**

  ```bash
  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:**

  ```bash
  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. 4.71: The regular and RT kernels

  In *The regular and RT kernels* you see the regular kernel on top and the RT below.

  - We want the RT one.

  ```bash
  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:

  ```bash
  #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.

Cyclic test

cyclic test is one tool for measuring the latency from when a thread is scheduled 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 4.61: rt/install.sh

    1. sudo apt install rt-tests
    2. # You can run gnuplot on the host
    3. 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 4.62: hist.gen

    1.#!/bin/sh
    2. # This code is from Julia Cartwright julia@kernel.org
    3. cyclic test -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 you 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:

![Histogram of Non-RT and RT kernels running cyclic test](cyclictest.png)

Fig. 4.72: Histogram of Non-RT and RT kernels running cyclic test

Notice the NON-RT data have much longer latencies. 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 150s.
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

4.1.9 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 from Bone-sized (Using a 128 x 128-Pixel LCD Cape) to much larger than the Bone (Using a Seven-Inch LCD Cape).

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.

Using a Seven-Inch LCD Cape

Problem You want to display the Bone’s desktop on a portable LCD.

Solution

Note: #TODO# The 4D Systems LCD capes would make a better example. CircuitCo is out of business.

A number of LCD capes are built for the Bone, ranging in size from three to seven inches. This recipe attaches a seven-inch BeagleBone LCD7 from CircuitCo (shown in 7” LCD) to the Bone.

7” LCD

Note: Seven-inch LCD from CircuitCo, 7” LCD was originally posted by CircuitCo at http://elinux.org/File: BeagleBone-LCD7-Front.jpg under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

To make this recipe, you will need:

• Seven-inch LCD cape
• A 5 V power supply

Just attach the Bone to the back of the LCD, making sure pin 1 of P9 lines up with pin 1 of +P9+ on the LCD. Apply a 5 V power supply, and the desktop will appear on your LCD, as shown in Seven-inch LCD desktop.

Attach a USB keyboard and mouse, and you have a portable Bone. Wireless keyboard and mouse combinations make a nice solution to avoid the need to add a USB hub.
Fig. 4.73: Seven-inch LCD desktop
Using a 128 x 128-Pixel LCD Cape

Problem You want to use a small LCD to display things other than the desktop.

Solution The MiniDisplay is a 128 x 128 full-color LCD cape that just fits on the Bone, as shown in MiniDisplay 128 x 128-pixel LCD from CircuitCo.

![MiniDisplay 128x128-pixel LCD from CircuitCo](image)

Fig. 4.74: MiniDisplay 128 x 128-pixel LCD from CircuitCo

To make this recipe, you will need:

- MiniDisplay LCD cape

Attach to the Bone and apply power. Then run the following commands:

```bash
# From http://elinux.org/CircuitCo:MiniDisplay_Cape
bone$ wget http://elinux.org/images/e/e4/Minidisplay-example.tar.gz
bone$ tar zmxvf Minidisplay-example.tar.gz
bone$ cd minidisplay-example
bone$ make
bone$ ./minidisplay-test
Unable to initialize SPI: No such file or directory
Aborted
```
Warning: You might get a compiler warning, but the code should run fine.

The MiniDisplay uses the Serial Peripheral Interface (SPI) interface, and it's not initialized. The manufacturer's website suggests enabling SPI0 by using the following commands:

```
bone$ export SLOTS=/sys/devices/bone_capemgr.*/slots
bone$ echo BB-SPIDEV0 > $SLOTS
```

Hmmm, something isn’t working here. Here’s how to see what happened:

```
bone$ dmesg | tail
[ 625.334497] bone_capemgr.9: part_number 'BB-SPIDEV0', version 'N/A'
[ 625.334673] bone_capemgr.9: slot #11: generic override
[ 625.334720] bone_capemgr.9: bone: Using override eeprom data at slot 11
[ 625.334769] bone_capemgr.9: slot #11: 'Override Board Name,00A0,Override Manuf,BB-SPIDEV0'
[ 625.335026] bone_capemgr.9: slot #11: \Requesting part number/version based \ 'BB-SPIDEV0-00A0.dtbo'
[ 625.335076] bone_capemgr.9: slot #11: Requesting firmware \ 'BB-SPIDEV0-00A0.dtbo' \ for board-name 'Override Board Name', version '00A0'
[ 625.335144] bone_capemgr.9: slot #11: dtbo 'BB-SPIDEV0-00A0.dtbo' loaded; \ converting to live tree
[ 625.341842] bone_capemgr.9: slot #11: BB-SPIDEV0 conflict P9.21 \ (#10:bspwm_P9_21_b) # ①
[ 625.351296] bone_capemgr.9: slot #11: Failed verification
```

① Shows there is a conflict for pin P9_21: it’s already configured for pulse width modulation (PWM).

Here’s how to see what’s already configured:

```
bone$ cat $SLOTS
0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
7: ff:P-O-L Override Board Name,00A0,Override Manuf,bspm_P9_42_27
8: ff:P-O-L Override Board Name,00A0,Override Manuf,bspm_P9_41_27
9: ff:P-O-L Override Board Name,00A0,Override Manuf,am33xx_pwm
10: ff:P-O-L Override Board Name,00A0,Override Manuf,bspm_P9_21_b # ①
```

① You can see the eMMC, HDMI, and three PWMs are already using some of the pins. Slot 10 shows P9_21 is in use by a PWM.

You can unconfigure it by using the following commands:

```
bone$ echo -10 > $SLOTS
bone$ cat $SLOTS
0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
7: ff:P-O-L Override Board Name,00A0,Override Manuf,bspm_P9_42_27
8: ff:P-O-L Override Board Name,00A0,Override Manuf,bspm_P9_41_27
9: ff:P-O-L Override Board Name,00A0,Override Manuf,am33xx_pwm
```

Now P9_21 is free for the MiniDisplay to use.
Note: In future Bone images, all of the pins will already be allocated as part of the main device tree using runtime pinmux helpers and configured at runtime using the config-pin utility. This would eliminate the need for device tree overlays in most cases.

Now, configure it for the MiniDisplay and run a test:

```
bone$ echo BB-SPIDEV0 > $SLOTS
bone$ ./minidisplay-test
```

You then see Boris, as shown in *Mini display Boris*.

**Mini display Boris**

Note: MiniDisplay showing Boris, *Mini display Boris* was originally posted by David Anders at http://elinux.org/File:Minidisplay-boris.jpg under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

---

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. 4.75: 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).

• 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 +VD*_5V+ p*n). 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**
  - I²C 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. Beaglebread shows the BeagleBone Breadboard, which is just one protoboard option.

**Beaglebread**

---

**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).

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).
Fig. 4.76: A simple robot controller diagram (quickBot.fzz)
Automatically generated schematic shows the schematic automatically generated by Fritzing. It’s a mess. It’s up to you to fix it.

Fig. 4.77: Automatically generated schematic

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?

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)

```javascript
#!/usr/bin/env node
var b = require('bonescript');
var M1_SPEED = 'P9_14'; //
var M1_FORWARD = 'P8_9';
var M1_BACKWARD = 'P8_11';
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; //
```

(continues on next page)
Fig. 4.78: Cleaned-up schematic

4.1. BeagleBone Cookbook
```javascript
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:
        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);  // ☑
}
```

(continues from previous page)
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;
    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) {
    speed = (speed > 1) ? 1 : speed; // 
    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); // 
}

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.

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.

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.

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.
Fig. 4.80: quickBot motor test showing kickback
Fig. 4.81: quickBot motor test code under scope

Fig. 4.82: Simple robot PCB
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 4.63: Outline SVG for BeagleBone cape (beaglebone_cape_boardoutline.svg)

```xml
<?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'>
<g id='board'>
<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'/>
</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.

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.

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.
Figure 70. Cape Board Dimensions

Fig. 4.83: Cape dimensions
4.1. BeagleBone Cookbook

Fig. 4.84: Rendered cape outline in Chrome

Fig. 4.85: PCB with original board, without notch for Ethernet connector
Fig. 4.86: PCB with all but the Board Layer hidden

Fig. 4.87: Clicking :load image file: with PCB1 selected
4.1. BeagleBone Cookbook

Fig. 4.88: Selecting the .svg file

Fig. 4.89: PCB Inspector
PCB Design Alternatives

There are other free PCB design programs. Here are a few.

TO PROD: The headings I’ve marked as bold lines really should be subheadings of “PCB Design Alternatives,” but AsciiDoc won’t let me go that deep (to the level). Is what I’ve done the best solution, or is there a way to create another heading level?

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 (Using a 128 x 128-Pixel LCD Cape) has the schematic shown in Schematic for the MiniDisplay cape and PCB shown in PCB for MiniDisplay cape.

![Schematic for the MiniDisplay cape](image1)

![PCB for MiniDisplay cape](image2)

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

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):

4.1. BeagleBone Cookbook
Fig. 4.93: The Eagle installation destination directory
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*.

![Eagle start screen](image)

**Fig. 4.94: 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 PCB 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.
BeagleBoard Docs, Release 0.0.20230323

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?

**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.
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
```

(continues on next page)
host$ cd ..
host$ python -m upconvert.upconverter -h

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:

<table>
<thead>
<tr>
<th>File type</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>openjson</td>
<td>i/o</td>
</tr>
<tr>
<td>kicad</td>
<td>i/o</td>
</tr>
<tr>
<td>geda</td>
<td>i/o</td>
</tr>
<tr>
<td>eagle</td>
<td>i/o</td>
</tr>
<tr>
<td>eaglexml</td>
<td>i/o</td>
</tr>
<tr>
<td>fritzing</td>
<td>in only schematic only</td>
</tr>
<tr>
<td>gerber</td>
<td>i/o</td>
</tr>
<tr>
<td>spectra</td>
<td>i/o</td>
</tr>
<tr>
<td>image</td>
<td>out only</td>
</tr>
<tr>
<td>ncdrill</td>
<td>out only</td>
</tr>
<tr>
<td>bom (csv)</td>
<td>out only</td>
</tr>
<tr>
<td>netlist (csv)</td>
<td>out only</td>
</tr>
</tbody>
</table>

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 ubuntu 63914 Nov 25 19:47 quickBot-eaglexml.sch
  -rw-r--r-- 1 ubuntu ubuntu 122193 Nov 25 19:43 quickBot.fzz
drw-rwrxr-x 9 ubuntu ubuntu 4096 Nov 25 19:42 schematic-file-converter

Output of Upverter conversion shows the output of the 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.
Fig. 4.95: Output of Upverter conversion
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* [http://oshpark.com](http://oshpark.com) 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. 4.96: The OSH Park QuickBot Cape shared project page](image)

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.
Fig. 4.97: Choosing “Extended Gerber” in Fritzing
Fig. 4.98: QuickBot PCB
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.

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. 4.99: QuickBot motors under test](image)

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 I²C 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.

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.

![CircuitHub part matching](image)

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 4.5: CircuitHub price examples (all prices USD) |
|-------------------------------|----------------|----------------|----------------|----------------|
| Quantity | PCB | Parts | Assembly | Per unit |
| 1        | $208.68 | $11.56 | $249.84 | $470.09 |
| 10       | $21.75  | $2.55  | $30.69  | $54.99  |
| 100      | $3.30   | $1.54  | $7.40   | $12.25  |
| 1000     | $0.98   | $0.92  | $2.79   | $4.79   |
| 10,000   | $0.90   | $0.92  | $2.32   | $4.16   |

Checking the Crystalfontz web page for the LCD, you can find the prices for the LCDs as well, as shown in LCD pricing (USD).
To enable more cape developers to launch their designs to the market, CircuitHub has launched a [http://campaign.circuithub.com](http://campaign.circuithub.com) 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, 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.

### 4.1.10 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 4.7: United States suppliers

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Website</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit</td>
<td><a href="http://www.adafruit.com">http://www.adafruit.com</a></td>
<td>Good for modules and parts</td>
</tr>
<tr>
<td>Amazon</td>
<td><a href="http://www.amazon.com">http://www.amazon.com</a></td>
<td>Carries everything</td>
</tr>
<tr>
<td>Digikey</td>
<td><a href="http://www.digikey.com/">http://www.digikey.com/</a></td>
<td>Wide range of components</td>
</tr>
<tr>
<td>MakerShed</td>
<td><a href="http://www.makershed.com/">http://www.makershed.com/</a></td>
<td>Good for modules, kits, and tools</td>
</tr>
<tr>
<td>RadioShack</td>
<td><a href="http://www.radioshack.com/">http://www.radioshack.com/</a></td>
<td>Walk-in stores</td>
</tr>
<tr>
<td>SeeedStudio</td>
<td><a href="http://www.seeedstudio.com/depot/">http://www.seeedstudio.com/depot/</a></td>
<td>Low-cost modules</td>
</tr>
<tr>
<td>SparkFun</td>
<td><a href="http://www.sparkfun.com">http://www.sparkfun.com</a></td>
<td>Good for modules and parts</td>
</tr>
</tbody>
</table>

### Table 4.8: Other suppliers

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Website</th>
<th>Notes</th>
</tr>
</thead>
</table>

### 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. `parts_jumper` lists places with jumper wires, and `parts_breadboard` shows where you can get breadboards.

### Table 4.9: Jumper wires

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td><a href="http://www.amazon.com/Elenco-Piece-Pre-formed-Jumper-Wire/dp/B0002H7AIG">http://www.amazon.com/Elenco-Piece-Pre-formed-Jumper-Wire/dp/B0002H7AIG</a></td>
</tr>
<tr>
<td>RadioShack</td>
<td><a href="http://www.radioshack.com/solderless-breadboard-jumper-wire-kit/2760173.html#VG51IPnF8fA">http://www.radioshack.com/solderless-breadboard-jumper-wire-kit/2760173.html#VG51IPnF8fA</a></td>
</tr>
<tr>
<td>SparkFun</td>
<td><a href="https://www.sparkfun.com/products/124">https://www.sparkfun.com/products/124</a></td>
</tr>
</tbody>
</table>

### Table 4.10: Breadboards

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>RadioShack</td>
<td><a href="http://www.radioshack.com/search?q=breadboard">http://www.radioshack.com/search?q=breadboard</a></td>
</tr>
<tr>
<td>SparkFun</td>
<td><a href="https://www.sparkfun.com/search/results?term=breadboard">https://www.sparkfun.com/search/results?term=breadboard</a></td>
</tr>
<tr>
<td>CircuitCo</td>
<td><a href="http://elinux.org/CircuitCo/BeagleBone_Breadboard">http://elinux.org/CircuitCo/BeagleBone_Breadboard</a></td>
</tr>
</tbody>
</table>

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 22k 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. RadioShack’s 500-piece Resistor Assortment is a bit more expensive, but it has a wider variety of resistors.
If you don’t need an entire kit of resistors, you can order a la carte from a number of places. RadioShack has 5-packs, and 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 k trimpot (or variable resistor) at SparkFun 10k POT, Adafruit 10k POT, or RadioShack 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 transistor has it. Adafruit NPN transistor has a nice 10-pack. SparkFun NPN transistor lets you buy them one at a time. DigiKey NPN transistor will gladly sell you 100,000.

The 1N4001 is a popular 1A diode. Buy one at SparkFun diode, 10 at Adafruit diode, 25 at RadioShack diode, or 40,000 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 L393D, Adafruit L393D, and DigiKey L393D 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, Adafruit red LED, RadioShack 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. eLinux.org BeagleBoard.org capes page keeps a current list.

Miscellaneous

Here are some things that don’t fit in the other categories.

<table>
<thead>
<tr>
<th>Table 4.11: Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3 V FTDI cable</td>
</tr>
<tr>
<td>SparkFun FTDI cable, Adafruit FTDI cable</td>
</tr>
<tr>
<td>USB WiFi adapter</td>
</tr>
<tr>
<td>Adafruit WiFi adapter</td>
</tr>
</tbody>
</table>

continues on next page
Table 4.11 – continued from previous page

<table>
<thead>
<tr>
<th>Product Description</th>
<th>Manufacturer/Supplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female HDMI to male microHDMI adapter</td>
<td>Amazon HDMI to microHDMI adapter</td>
</tr>
<tr>
<td>HDMI cable</td>
<td>SparkFun HDMI cable</td>
</tr>
<tr>
<td>Micro HDMI to HDMI cable</td>
<td>Adafruit HDMI to microHDMI cable</td>
</tr>
<tr>
<td>HDMI to DVI Cable</td>
<td>SparkFun HDMI to DVI cable</td>
</tr>
<tr>
<td>HDMI monitor</td>
<td>Amazon HDMI monitor</td>
</tr>
<tr>
<td>Powered USB hub</td>
<td>Amazon power USB hub, Adafruit power USB hub</td>
</tr>
<tr>
<td>Keyboard with USB hub</td>
<td>Amazon keyboard with USB hub</td>
</tr>
<tr>
<td>Soldering iron</td>
<td>SparkFun soldering iron, Adafruit soldering iron</td>
</tr>
<tr>
<td>Oscilloscope</td>
<td>Adafruit oscilloscope</td>
</tr>
<tr>
<td>Multimeter</td>
<td>SparkFun multimeter, Adafruit multimeter</td>
</tr>
<tr>
<td>PowerSwitch Tail II</td>
<td>SparkFun PowerSwitch Tail II, Adafruit PowerSwitch Tail II</td>
</tr>
<tr>
<td>Servo motor</td>
<td>SparkFun servo motor, Adafruit servo motor</td>
</tr>
<tr>
<td>5 V power supply</td>
<td>SparkFun 5V power supply, Adafruit 5V power supply</td>
</tr>
<tr>
<td>3 V to 5 V motor</td>
<td>SparkFun 3V-5V motor, Adafruit 3V-5V motor</td>
</tr>
<tr>
<td>3 V to 5 V bipolar stepper motor</td>
<td>SparkFun 3V-5V bipolar stepper motor, Adafruit 3V-5V bipolar stepper motor</td>
</tr>
<tr>
<td>3 V to 5 V unipolar stepper motor</td>
<td>Adafruit 3V-5V unipolar stepper motor</td>
</tr>
<tr>
<td>Pushbutton switch</td>
<td>SparkFun pushbutton switch, Adafruit pushbutton switch</td>
</tr>
<tr>
<td>Magnetic reed switch</td>
<td>SparkFun magnetic reed switch</td>
</tr>
<tr>
<td>LV-MaxSonar-EZ1 Sonar Range Finder</td>
<td>SparkFun LV-MaxSonar-EZ1, Amazon LV-MaxSonar-EZ1</td>
</tr>
<tr>
<td>HC-SR04 Ultrasonic Range Sensor</td>
<td>Amazon HC-SR04</td>
</tr>
<tr>
<td>Rotary encoder</td>
<td>SparkFun rotary encoder, Adafruit rotary encoder</td>
</tr>
<tr>
<td>GPS receiver</td>
<td>SparkFun GPS, Adafruit GPS</td>
</tr>
<tr>
<td>BLE USB dongle</td>
<td>Adafruit BLE USB dongle</td>
</tr>
<tr>
<td>SensorTag</td>
<td>DigiKey SensorTag, Amazon SensorTag, TI SensorTag</td>
</tr>
<tr>
<td>Syba SD-CM-UAUD USB Stereo Audio Adapter</td>
<td>Amazon USB audio adapter</td>
</tr>
<tr>
<td>Sabrent External Sound Box USB-SBCV</td>
<td>Amazon USB audio adapter (alt)</td>
</tr>
<tr>
<td>Vantec USB External 7.1 Channel Audio Adapter</td>
<td>Amazon USB audio adapter (alt2)</td>
</tr>
<tr>
<td>Nokia 5110 LCD</td>
<td>Adafruit 5110 LCD, SparkFun 5110 LCD</td>
</tr>
<tr>
<td>BeagleBone LCD7</td>
<td>eLinux LCD7</td>
</tr>
<tr>
<td>MiniDisplay Cape</td>
<td>eLinux minidisplay</td>
</tr>
</tbody>
</table>

### 4.2 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

#### 4.2.1 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 processor running Linux and two\(^1\) Programmable Real-Time Units (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**

- Pocket Beagle System Reference Manual
- **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.

\(^1\) Four if you are on the BeagleBone AI
Fig. 4.102: Blue balancing
**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 multiplexer that lets you select what I/O appears on a given pin. The Blue has the mux already configured to 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:

```bash
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.

```bash
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.
- `-r {ch}`  Use DSM radio channel (ch) to control servo.
- `-h`  Print this help message.

Sample use to center servo channel 1:

```bash
rc_test_servo -c 1 -p 0.0
```

**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*.

<table>
<thead>
<tr>
<th>PRU pin</th>
<th>Blue pin</th>
<th>Black pin</th>
<th>Pocket pin</th>
<th>AI pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>pru1_r30_8</td>
<td>P8_27</td>
<td>P2.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pru1_r30_10</td>
<td>P8_28</td>
<td>P1.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pru1_r30_9</td>
<td>P8_29</td>
<td>P1.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pru1_r30_11</td>
<td>P8_30</td>
<td>P1.04</td>
<td>P9_27</td>
<td></td>
</tr>
<tr>
<td>pru1_r30_6</td>
<td>P8_39</td>
<td></td>
<td></td>
<td>P8_19</td>
</tr>
<tr>
<td>pru1_r30_7</td>
<td>P8_40</td>
<td></td>
<td></td>
<td>P8_13</td>
</tr>
<tr>
<td>pru1_r30_4</td>
<td>P8_41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pru1_r30_5</td>
<td>P8_42</td>
<td></td>
<td></td>
<td>P8_18</td>
</tr>
</tbody>
</table>

You can find these details in the

- Pocket Beagle 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.
You can modify `rc_test_servos.c`. You'll find it on the bone online at https://github.com/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 reprogramming 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.

<table>
<thead>
<tr>
<th>Raw encoder positions</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>^C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>^C</td>
</tr>
</tbody>
</table>

You can also access these hardware encoders on the Black and Pocket using the pins shown in *eEQP to pin mapping*.

**eEQP to pin mapping**

<table>
<thead>
<tr>
<th>eEQP</th>
<th>Blue pin</th>
<th>Black pin A</th>
<th>Black pin B</th>
<th>Al pin A</th>
<th>Al pin B</th>
<th>Pocket pin A</th>
<th>Pocket pin B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>E1</td>
<td>P9_42B</td>
<td>P9_27</td>
<td></td>
<td></td>
<td>P1.31</td>
<td>P2.24</td>
</tr>
<tr>
<td>1</td>
<td>E2</td>
<td>P8_35</td>
<td>P8_33</td>
<td>P8_35</td>
<td>P8_33</td>
<td>P2.10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>E3</td>
<td>P8_12</td>
<td>P8_11</td>
<td>P8_12</td>
<td>P8_11</td>
<td>P2.24</td>
<td>P2.33</td>
</tr>
<tr>
<td>2</td>
<td>E4</td>
<td>P8_16</td>
<td>P8_15</td>
<td></td>
<td></td>
<td></td>
<td>P2.09</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>P8_25</td>
<td>P8_24</td>
<td></td>
<td></td>
<td>P2.18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>P9_42</td>
<td>P9_27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The I/O pins on the Beagles have a multiplexer that lets you select what I/O appears on a given pin. The Blue has the mux already configured to 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.

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 100Mps, 14-channel logic analyzer

Solution  BeagleLogic documentation is a 100Mps, 14-channel logic analyzer that runs on the Beagle.

Information

BeagleLogic turns your BeagleBone [Black] into a 14-channel, 100Mps 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 remotproc 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 4.64: 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/ →0761ef7b10822cfd4b3efdf194837f49c/raw/eb2c6f59ff5cc9b710d0d7d4a40cc0c050/ →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!
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

```bash
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/`.

```bash
bone$ *cd /sys/devices/virtual/misc/beaglelogic*
bone$ *ls*
buffers filltestpattern power state uevent
bufunit size lasterror samplerate subsystem
development memalloc sampleunit triggerflags
bone$ *cat samplerate*
1000
bone$ *cat sampleunit*
8bit
```

You can set the sample rate by simply writing to `samplerate`.

```bash
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.

```bash
bone$ *dmesg -Hw*
[Jul25 08:46] misc beaglelogic: capture started with sample rate=100000000 Hz, ...
```

(continues on next page)
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.

**NeoPixels – 5050 RGB LEDs with Integrated Drivers (Falcon Christmas)**

**Problem** You have an Adafruit NeoPixel LED string, Adafruit NeoPixel LED matrix or any other type of WS2812 LED and want to light it up.

**Solution** If you are driving just one string you can write your own code (See [WS2812 (NeoPixel) driver](https://www.adafruit.com/product/1398)). If you plan to drive multiple strings, then consider Falcon Christmas (FPP). FPP can be used to drive both LEDs with an integrated driver (neopixels) or without an integrated driver. Here we’ll show you how to set up for the integrated drive and in the next section the no driver LEDs will be show.

**Hardware** For this setup we’ll wire a single string of NeoPixels to the Beagle. I’ve attached the black wire on the string to ground on the Beagle and the red wire to a 3.3V pin on the Beagle. The yellow data in line is attached to P1.31 (I’m using a PocketBeagle.).

How did I know to attach to P1.31? The FalconChristmas git repo [https://github.com/FalconChristmas/fpp](https://github.com/FalconChristmas/fpp) has files that tell which pins attach to which port. [https://github.com/FalconChristmas/fpp/blob/master/capes/pb/strings/F8-B-20.json](https://github.com/FalconChristmas/fpp/blob/master/capes/pb/strings/F8-B-20.json) has a list of 20 ports and where they are connected. Pin P1.31 appears on line 27. It’s the 20th entry in the list. You could pick any of the others if you’d rather.

**Software Setup** 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](https://www.adafruit.com/product/1398).

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 Pixel Strings tab. Earlier we noted that P1.31 is attached to port 20. Note that at the bottom of the screen, port 20 has a PIXEL COUNT of 24. We’re telling FPP our string has 24 NeoPixels and they are attached to port 2 which in P1.31.

Be sure to check the Enable String Cape.

Next we need to test the display. Select Display Testing shown in Selecting Display Testing.

Set the End Channel to 72. (72 is 3*24) Click Enable Test Mode and your matrix should light up. Try the different testing patterns shown in Display Testing Options.

**Note:** Clicking on the -3 will subtract three from the End Channel, which should then display three fewer LEDs which is one NeoPixel. The last of your NeoPixels should go black. This is an easy way to make sure you have the correct pixel count.


Tell it you have 72 LEDs and enable the input as shown in Setting Channel Inputs.

Finally go to the Status Page as shown in Watching the status.

Now run a program on another computer that generated E1.31 packets. `e1.31-test.py -Example of generating packets to control the NeoPixels` is an example python program.
Fig. 4.104: Falcon Play Program Control
Listing 4.65: e1.31-test.py - Example of generating packets to control the NeoPixels

```python
#!/usr/bin/env python3
# Controls a NeoPixel (WS2812) string via E1.31 and FPP
# https://pypi.org/project/sacn/
# https://github.com/FalconChristmas/fpp/releases
import sacn
import time

# provide an IP-Address to bind to if you are using Windows and want to use multicast
sender = sacn.sACNsender("192.168.7.1")
sender.start() # start the sending thread
sender.activate_output(1) # start sending out data in the 1st universe
sender[1].multicast = False # set multicast to True
sender[1].destination = "192.168.7.2" # or provide unicast information.
sender.manual_flush = True # turning off the automatic sending of packets
# Keep in mind that if multicast is on, unicast is not used
LEDcount = 24 # Have green fade as it goes
data = []
for i in range(LEDcount):
    data.append(0) # Red
    data.append(i) # Green
    data.append(0) # Blue
sender[1].dmx_data = data
sender.flush()
time.sleep(0.5)
```

(continues on next page)
**4.2. PRU Cookbook**

![Channel Outputs Settings](image-url)

**Fig. 4.106: Channel Outputs Settings**

<table>
<thead>
<tr>
<th>PORT</th>
<th>DESCRIPTION</th>
<th>START CHANNEL</th>
<th>PIXEL COUNT</th>
<th>GROUP COUNT</th>
<th>END CHANNEL</th>
<th>DIRECTION</th>
<th>COLOR ORDER</th>
<th>START NULLS</th>
<th>END NULLS</th>
<th>ZDO</th>
<th>BRIGHTNESS</th>
<th>GAMMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Forward</td>
<td>RGB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Forward</td>
<td>RGB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Forward</td>
<td>RGB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Forward</td>
<td>RGB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Forward</td>
<td>RGB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td>65</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Forward</td>
<td>RGB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td>75</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Forward</td>
<td>RGB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td>85</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Forward</td>
<td>RGB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>1.0</td>
</tr>
</tbody>
</table>

---

**Enable String Caps**

- **Cape Type**: FB-PB (No Serial)
- **Pixel Timing**: Normal (vs2Red)

Press F2 to save and set the start channel on the next row.
Fig. 4.107: Selecting Display Testing

Fig. 4.108: Display Testing Options
Fig. 4.109: Going to Channel Inputs

Fig. 4.110: Setting Channel Inputs
# Turn off all LEDs

```python
data=[]
for i in range(3*LEDcount):
    data.append(0)
sender.flush()
sender[1].dmx_data = data
time.sleep(0.5)
```

# Have red fade in

```python
data = []
for i in range(LEDcount):
    data.append(i)
data.append(0)
data.append(0)
sender[1].dmx_data = data
sender.flush()
time.sleep(0.25)
```

# Make LED circle 5 times

```python
for j in range(15):
    for i in range(LEDcount-1):
        data[3*i+0] = 0
        data[3*i+1] = 0
        data[3*i+2] = 0
        data[3*i+3] = 0
        data[3*i+4] = 64
        data[3*i+5] = 0
        sender[1].dmx_data = data
        sender.flush()
time.sleep(0.02)
```

# Wrap around

```python
i = LEDcount-1
data[0] = 0
data[1] = 64
data[2] = 0
data[3*i+0] = 0
```

(continues on previous page)
```python
63 data[3*i+1] = 0
64 data[3*i+2] = 0
65 sender[1].dmx_data = data
66 sender.flush()
67 time.sleep(0.02)
68 time.sleep(2)  # send the data for 10 seconds
69 sender.stop()  # do not forget to stop the sender
```

e1.31-test.py

**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 Adafruit shown in *Adafruit LED Matrix*.

![Adafruit LED Matrix](image)

*Fig. 4.112: 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 PocketScrollr LED Panel Cape. (See to purchase.) Building and Octoscroller Matrix Display gives details for using the BeagleBone Black.

*Pocket Beagle Driving a P5 RGB LED Matrix via the PocketScrollr 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.
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 an 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.

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.

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/
Fig. 4.114: Falcon Play Program Control
xLights can be installed on your host computer (not the Beagle) by following instructions at [https://xlights.org/releases/](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.

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.
Fig. 4.116: Channel Outputs Settings
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 to 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.

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.
4.2. PRU Cookbook

Fig. 4.118: Display Testing Options
Fig. 4.119: xLights Setup

Fig. 4.120: Setting Up E1.31
Fig. 4.121: xLights setup for P5 display

Fig. 4.122: Setting up the Matrix Layout
Fig. 4.123: Layout details for P5 matrix

Fig. 4.124: Starting a new sequence
## 4.2. PRU Cookbook

**Fig. 4.125: E1.31 Inputs**

<table>
<thead>
<tr>
<th>Enable Input</th>
<th>Timeout</th>
<th>Inputs Count</th>
<th>E1.31 / ArtNet / DDP Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 1 512 1 1 512</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 513 1024 2 1 512</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 1025 1536 3 1 512</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 1537 2048 4 1 512</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 2049 2560 5 1 512</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 2561 3072 6 1 512</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 3073 3584 7 1 512</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 3585 4096 8 1 512</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 4097 4608 9 1 512</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 4609 5120 10 1 512</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 5121 5632 11 1 512</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>12</td>
<td>E1.31 - Multicas 5633 6144 12 1 512</td>
</tr>
</tbody>
</table>
Fig. 4.126: Bridge Mode
Fig. 4.127: xLights test page

Fig. 4.128: xLights Twinkle test pattern
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 typesafe, and data types of variables are decided during compilation. simpPRU 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:

```bash
echo
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
decho
echo
```

Now, suppose you wanted to run the LED blink example which is reproduced here.

```python
Listing 4.66: LED Blink (blink.sim)

1/2 From: https://simppru.readthedocs.io/en/latest/examples/led_blink/  */
 invention 07
 while : 1 == 1 {
 (continues on next page)
```
Fig. 4.130: FPP file manager
Fig. 4.131: Adding a new playlist to FPP
Fig. 4.132: Adding a new playlist to FPP
digital_write(P1_31, true);
delay(250); /**< Delay 250 ms */
digital_write(P1_31, false);
delay(250);
}

blink.sim

Just run simprru

bone$ simprru 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://simprru.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.
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/

**ArduPilot**  
ArduPilot is an open source autopilot system supporting multi-copters, traditional helicopters, fixed wing aircraft and rovers. ArduPilot runs on 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/

### 4.2.2 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 Cloud9 IDE and how to blink an LED.

If you already have your Beagle and know your way around it, you can find the code (and the whole book) on the PRU Cookbook github site: https://github.com/MarkAYoder/PRUCookbook.

**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.

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 or 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
• Charger input: 9-18V
• Motor control: 8 6V 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.

![BeagleBone Blue EduMIP Kit](https://www.hackster.io/53815/controlling-edumip-with-ni-labview-2005f8)

Fig. 4.136: BeagleBone Blue EduMIP Kit

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
t• 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)
• 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 lastest version of Debian that is available for your Bone.

**Solution**  On your host computer open a browser and go to http://beagleboard.org/latest-images. 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.

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.

Once the SD is flashed, insert it in the Beagle and power it up.

Cloud9 IDE

**Problem**  How do I manage and edit my files?
Fig. 4.139: Latest Debian images

Fig. 4.140: Etcher
Solution  The image you downloaded includes Cloud9, a web-based integrated development environment (IDE) as shown in Cloud9 IDE.

![Cloud9 IDE](image)

Fig. 4.141: Cloud9 IDE

Just point the browser on your host computer to http://192.168.7.2 and start exploring. 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 Cloud9 Showing Home files.

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*
bone$ *ls*
```

```
nfs-uEnv.txt boot etc ID.txt lost+found mnt opt root sbin sys ...
→ user
bin dev home lib media nfs-uEnv.txt proc run srv tmp ...
→ var
```

Now you can reach all the files from Cloud9.

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 (and the whole book) on the PRU Cookbook github site: <https://github.com/MarkAYoder/PRUCookbook/tree/master/docs>. Just clone it on your Beagle and then look in the docs directory.

Each chapter has its own directory and within that directory is a code directory that has all of the code.

Go and explore.
Fig. 4.142: Cloud9 Showing Home files
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 4.67: hello.pru0.c

```#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"

volatile register unsigned int __R30;
volatile register unsigned int __R31;

void main(void) {
    int i;

    uint32_t *gpio1 = (uint32_t *)GPIO1;
    /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
    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();
}
```

Later chapters will go into details of how this code works, but if you want to run it right now do the following.

```
bone$ *git clone https://github.com/MarkAYoder/PRUCookbook.git*
bone$ *cd PRUCookbook/docs/02start/code*
```

**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*
/var/lib/cloud9/common/Makefile:28: MODEL=TI_AM335x_BeagleBone_Black,TARGET=hello.
 -->pru0,COMMON=/var/lib/cloud9/common
```

(continues on next page)
Running Code on the AI

```
bone$ *make TARGET=hello.pru1_1*
/var/lib/cloud9/common/Makefile:28: MODEL=BeagleBoard.org_BeagleBone_AI,
  →TARGET=hello.pru1_1
  → Stopping PRU 1_1
CC hello.pru1_1.c
"/var/lib/cloud9/common/prugpio.h", line 4: warning #1181-D: #warning directive:
  →"Found AI"
LD /tmp/cloud9-examples/hello.pru1_1.o
  → copying firmware file /tmp/cloud9-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/cloud9-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.

4.2.3 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$ git clone https://github.com/MarkAYoder/PRUCookbook.git
```

**Note:**  #TODO#: There needs to be a code-only repo that is validated against the documentation code to be identical for specific version. The version needs to be noted in the documentation.

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](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 an error about `/dev/remoteproc` is missing.
Solution  Edit `/boot/uEnv.txt` and enable `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-PRPROC-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 PRUCookbook/docs/06io/code
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
/var/lib/cloud9/common/Makefile:28: MODEL=TI_AM335x_BeagleBone_Black,TARGET=gpio.
  -pru0,COMMON=/var/lib/cloud9/common
  /var/lib/cloud9/common/Makefile:147: GEN_DIR=/tmp/cloud9-examples,CHIP=am335x,
  -PROC=pru,PRUN=0,PRU_DIR=/sys/class/remoteproc/remoteproc1,EXE=.out
  - Stopping PRU 0
/bin/sh: 1: echo: echo: I/O error
Cannot stop 0
CC gpio.pru0.c
"/var/lib/cloud9/common/prugpio.h", line 53: warning #1181-D: #warning directive:  
  "Found am335x"
LD /tmp/cloud9-examples/gpio.pru0.o
- copying firmware file /tmp/cloud9-examples/gpio.pru0.out to /lib/firmware/
  -am335x-pru0-fw
write_init_pins.sh
writing "out" to "/sys/class/gpio/gpio30/direction"
- Starting PRU 0
MODEL  = TI_AM335x_BeagleBone_Black
PROC   = pru
PRUN   = 0
```

(continues on next page)
Congratulations, you 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_1, 2_0, 2_1 on the AI or 0 or 1 on the others).

You can override the TARGET on the command line.

```bash
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.

```bash
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 4.68: gpio_setup.sh

```bash
#!/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
done

gpio_setup.sh
```

4.2. PRU Cookbook 623
<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-5</td>
<td>Set which PRU to use and which file to compile.</td>
</tr>
<tr>
<td>7</td>
<td>Figure out which type of Beagle we have.</td>
</tr>
<tr>
<td>9-21</td>
<td>Based on the type, set the pins.</td>
</tr>
<tr>
<td>23-28</td>
<td>Configure (set the pin mux) for each of the pins.</td>
</tr>
</tbody>
</table>

**Tip:** The BeagleBone AI has its 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:

```bash
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:

```bash
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.

```bash
Listing 4.69: Local 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. Both files can be found in `/var/lib/cloud9/common`.

Listing 4.70: am335x_pru.cmd

```c
/* AM335x_PRU.cmd */

// 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 */

  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 =

  0x00000600 CREGISTER=3

  PRU_IEP     : org = 0x0002E000 len =

  0x0000031C CREGISTER=26

  PRU_INTC    : org = 0x00020000 len =

  0x00015000 CREGISTER=0

  PRU_UART    : org = 0x00028000 len =

  0x00000038 CREGISTER=7

  (continues on next page)
```
DCAN0 : org = 0x481CC000 len =
\-0x000001E8 CREGISTER=14
DCAN1 : org = 0x481D0000 len =
\-0x000001E8 CREGISTER=15
DMTMR2 : org = 0x48040000 len =
\-0x0000000C 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 = 0x4A000000 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 =
\-0x000000100 CREGISTER=8
MCSP10 : org = 0x48030000 len =
\-0x000001A4 CREGISTER=6
MCSP11 : org = 0x481A0000 len =
\-0x000001A4 CREGISTER=16
MMCHS0 : org = 0x48060000 len =
\-0x00000300 CREGISTER=5
SPINLOCK : org = 0x480CA000 len =
\-0x00000890 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

} (continued from previous page)

/* 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
.sysmem > PRU_DMEM_0_1, PAGE 1

(continues on next page)
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.

<table>
<thead>
<tr>
<th>AM335x_PRU.cmd important things</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Line</strong></td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>72</td>
</tr>
<tr>
<td>73</td>
</tr>
<tr>
<td>74</td>
</tr>
</tbody>
</table>

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:
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.

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.

```bash
#!/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
```

This script loops through each of the pins and configures it.

**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.

```bash
#!/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=""
else
  echo " Not Found"
pins=""
fi
for pin in $pins
doi
echo $pin
cfgpin-pin qep
cfgpin-q $pin
done

# Configure PRU pins
if [ $machine = "Black" ]; then
echo " Found"
pins="P8_16 P8_15"
eelif [ $machine = "Blue" ]; then
echo " Found"
pins=""
eelse
  echo " Not Found"
pins=""
efi
for pin in $pins
do碘
  echo $pin
  cfgpin-pin pruin
  cfgpin-q $pin
done
```

`encoder_setup.sh`

4.2. PRU Cookbook 629
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.

4.2.4  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.

![LED used for debugging P9_29](image)

Fig. 4.143: 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](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](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.

```
bones$ *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   0x000034000   0x000000000   0x00022000
   1   0x000038000   0x000002000   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_va [length] - Raw dump of PRU data memory (32-bit byte
<--offset from beginning of full PRU memory block - all PRUs)
DD memory_location_va [length] - Dump data memory (32-bit byte offset from
<--beginning of PRU data memory)
DI memory_location_va [length] - Dump instruction memory (32-bit byte offset
<--from beginning of PRU instruction memory)
DIS memory_location_va [length] - Disassemble instruction memory (32-bit byte
<--offset from beginning of PRU instruction memory)
G - 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_va value1 [value2 [value3 ...]] - Write a 32-bit value to a...
<--raw (offset from beginning of full PRU memory block)
WRD memory_location_va value1 [value2 [value3 ...]] - Write a 32-bit value to...
<--PRU data memory for current PRU
WRI memory_location_va 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: 0x000008003
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.

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

R0: 0x00000000  R08: 0x00000000  R16: 0x00000001  R24: 0x00000002
R01: 0x00000000  R09: 0xaf40d3c0  R17: 0x00000000  R25: 0x00000003
R02: 0x00000000  R10: 0x00000000  R18: 0x00000000  R26: 0x00000003
R03: 0x00000000  R11: 0xc50cbe82  R19: 0x00000000  R27: 0x00000002
R04: 0x00000000  R12: 0xb037c0d7  R20: 0x00000000  R28: 0x8ca9d976
(continues on next page)
R05: 0x00000009  R13: 0xf48bbe23  R21: 0x441fb678  R29: 0x00000002
R06: 0x00000000  R14: 0x000000134  R22: 0xc8cc0752  R30: 0x00000000
R07: 0x00000000  R15: 0x000000134  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 4.12: Important memory locations

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000</td>
<td>Start of the stack for PRU 0. The file <code>AM335x_PRU.cmd</code> specifies where the stack is.</td>
</tr>
<tr>
<td>0x00100</td>
<td>Start of the heap for PRU 0.</td>
</tr>
<tr>
<td>0x00200</td>
<td>Start of DRAM that your programs can use. The <code>Makefile</code> specifies the size of the stack and the heap.</td>
</tr>
<tr>
<td>0x10000</td>
<td>Start of the memory shared between the PRUs.</td>
</tr>
</tbody>
</table>

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 0x000000200 0x86e5c18 0xfeb21aca
```

Here we see some values on the heap.

PRU0> *dd 0x200*  
```
  dd 0x200  
  Absolute addr = 0x0200, offset = 0x0000, Len = 16
  [0x0200] 0x00000001 0x00000002 0x00000003 0x00000004
  [0x0210] 0x00000003 0x00000002 0x00000001 0x00000000
  [0x0220] 0x0a4fe833 0xb22e2bda 0xe575236 0xc50cbefd
  [0x0230] 0xb037c0d7 0xf48bbe23 0x88c460f0 0x15550d4
```

Data written explicity to 0x0200 of the DRAM.

PRU0> *dd 0x10000*  
```
  dd 0x10000
  Absolute addr = 0x10000, offset = 0x0000, Len = 16
  [0x10000] 0x8ca9d976 0x0ebcbe319 0x3aebce31 0x68c44d8b
  [0x10010] 0x3c70ba7e 0x2fefa993b 0x15c67fa5 0x8fbf68557
  [0x10020] 0x5add81b4f 0x4a55071a 0x48576eb7 0x104786b
  [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.

4.2. PRU Cookbook 633
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 Arduino playbook; use the UART (serial port) to output debug information. The PRU has its 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.

![FTDI cable](image)

Fig. 4.144: 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>
<thead>
<tr>
<th>FTDI pin</th>
<th>Color</th>
<th>Black pin</th>
<th>AI 1 pin</th>
<th>AI 2 pin</th>
<th>Pocket</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>black</td>
<td>P9_1</td>
<td>P8_1</td>
<td>P8_1</td>
<td>P1_16</td>
<td>ground</td>
</tr>
<tr>
<td>4</td>
<td>orange</td>
<td>P9_24</td>
<td>P8_43</td>
<td>P8_33a</td>
<td>P1_12</td>
<td>rx</td>
</tr>
<tr>
<td>5</td>
<td>yellow</td>
<td>P9_26</td>
<td>P8_44</td>
<td>P8_31a</td>
<td>P1_06</td>
<td>tx</td>
</tr>
</tbody>
</table>

\(^1\) FTDI images are from the BeagleBone Cookbook
Fig. 4.145: FTDI connector

Fig. 4.146: FTDI to BB Black
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.

```sh
# 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

```sh
DRA7XX_CORE_IOPAD(0x3610, *PIN_INPUT* | MUX_MODE10) // C6: P8.33a:
```

Listing 4.73: uart1.pru1_0.c

```c
/* 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 */
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 */
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 */
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

Set the following variables so make will know what to compile.

Listing 4.74: make

```
bones$ *make TARGET=uart1.pru0*
/var/lib/cloud9/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_Black,TARGET=uart1.
    pru0
    - Stopping PRU 0
    - copying firmware file /tmp/cloud9-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 characters (H) and then as you enter characters on the keyboard, the rest of the message will appear.

Fig. 4.147: uart1.pru0.c output

Here’s the code (uart1.pru1_0.c) that does it.

Listing 4.75: 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

int main(void)
{
    uint8_t tx;
    uint8_t rx;
    uint8_t cnt;
    char* hostBuffer;
    CT_UART.DIVISOR_REGISTER_LSB_ = 104;
    CT_UART.DIVISOR_REGISTER_MSB_ = 0;
    CT_UART.MODE_DEFINITION_REGISTER = 0x0;
    CT_UART.INTERRUPT_ENABLE_REGISTER = 0x7;
    CT_UART.INTERRUPT_IDENTIFICATION_REGISTER_FIFO_CONTROL_REGISTER = (0x8) | (0x4) | (0x2) | (0x1);
    //CT_UART.FCR = (0x80) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO trigger
    CT_UART.LINE_CONTROL_REGISTER = 3;
    CT_UART.MODEM_CONTROL_REGISTER = 0x00;
    CT_UART.POWERMANAGEMENT_AND_EMULATION_REGISTER = 0x6001;

    /* 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 */
    /* Enable Interrupts in UART module. This allows the main thread to poll_for
     * Receive Data Available and Transmit Holding Register Empty */
    /* 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 */
    /* Choose desired protocol settings by writing to LCR */
    /* 8-bit word, 1 stop bit, no parity, no break control and no divisor_latch */
    /* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x_oversample
     * 192MHz / 104 / 16 = ~115200 */
    CT_UART.DIVISOR_REGISTER_MSB_ = 0;
    CT_UART.MODE_DEFINITION_REGISTER = 0x0;
    CT_UART.INTERRUPT_ENABLE_REGISTER = 0x7;
    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;
    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 = 0x6001;

    */
    /* Priming the 'hostbuffer' with a message */
    hostBuffer = "Hello! This is a long string\r\n";
}

#include <stdint.h>
#include <pru_uart.h>
#include "resource_table_empty.h"

uart1.pru1_0.c

Note: I'm using the AI version of the code since it uses variables with more descriptive 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.
/* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily) */
#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;
}

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
(continues on next page)
/* 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++) {
      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 */

(continues on next page)
uart2.pru0.c

If you want to try `uart2.pru0.c`, run the following:

**Listing 4.77: make**

```bash
bone$ *make TARGET=uart2.pru0*
/var/lib/cloud9/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_Black,TARGET=uart2.
  → pru0
  - Stopping PRU 0
  - copying firmware file /tmp/cloud9-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:

![uart2.pru0.c output](image)

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.TEMP);` 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 4.78: uart2.pru1_0.c**

```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)
(continues on next page)
/* 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)
{
    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

    #*** INITIALIZE ***/

    /* 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;

    (continues on next page)
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) | (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;
/* 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 INITIALIZE ****/
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++) {
    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.POWERMANAGEMENT_AND_EMULATION_REGISTER = 0x0;
(continues on next page)
/* Halt PRU core */
__halt();
}

uart2.pru1_0.c

More complex examples can be built using the principles shown in these examples.

Copyright

Listing 4.79: copyright.c

/*
 * Copyright (C) 2015 Texas Instruments Incorporated - http://www.ti.com/
 */

/* Redistributions 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

4.2.5 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
Memory Allocation

Problem  I want to control where my variables are stored in memory.

Solution  Each PRU has its 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 4.80: shared.pro0.c - Examples of Using Different Memory Locations

```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
(continues on next page)*/
the same subsystem must take care **not** to allocate data to that memory.

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;
PRU_DMEM1 volatile uint32_t shared_3;
PRU_SRAM volatile uint32_t shared_1;
volatile uint32_t shared_0;

/* 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 */
    //*************************************************************************/
    /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
    CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
    /**************************************************************************/
    /* Access PRU Shared RAM using Constant Table */
    /**************************************************************************/
    /* C28 defaults to 0x00000000, we need to set bits 23:8 to 0x0100 in order...

---to have it point to 0x00010000 */
    PRU0_CTRL.CTPPR0_bit.C28_BLK_POINTER = 0x0100;
    shared_0 = 0xfeef;
    shared_1 = 0xdeadbeef;
    shared_2 = shared_2 + 0xfeed;
    shared_3 = 0xdead;
    shared_4 = 0xbeed;
    shared_5[0] = 0x1234;
    shared_6 = 0x4321;
    shared_7 = 0x9876;
    /* Halt PRU core */
    __halt();
}

Discussion  Here's the line-by-line
Table 4.14: Line-byline for shared.pru0.c

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>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.</td>
</tr>
<tr>
<td>8, 9</td>
<td>These are like the previous line except for the DMEM sections.</td>
</tr>
<tr>
<td>16</td>
<td>Variables declared outside of main() are put on the heap.</td>
</tr>
<tr>
<td>17</td>
<td>Adding PRU_SRAM has the variable stored in the shared memory.</td>
</tr>
<tr>
<td>18, 19</td>
<td>These are stored in the PRU’s local RAM.</td>
</tr>
<tr>
<td>20, 21</td>
<td>These lines are for storing in the .bss section as declared on line 74 of am335x_pru.cmd.</td>
</tr>
<tr>
<td>28-31</td>
<td>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.</td>
</tr>
<tr>
<td>36, 37</td>
<td>Variable declared inside main() go on the stack.</td>
</tr>
</tbody>
</table>

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
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*
/var/lib/cloud9/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_Black,TARGET=shared, → pru0
- Stopping PRU 0
- copying firmware file /tmp/cloud9-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/cloud9-examples/shared.pru0.map*
....
```

(continues on next page)

4.2. PRU Cookbook 649
We see, shared_0 had no directives and was placed 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 runtime.

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 AM33x
PRUSS memory address 0x4a300000
PRUSS memory length 0x00080000

offsets below are in 32-bit byte addresses (not ARM byte addresses)
<table>
<thead>
<tr>
<th>PRU</th>
<th>Instruction</th>
<th>Data</th>
<th>Ctrl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00034000</td>
<td>0x00000000 0x00022000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x00038000</td>
<td>0x00000200 0x00024000</td>
<td></td>
</tr>
</tbody>
</table>

PRU0> *d 0*
Absolute addr = 0x0000, offset = 0x0000, Len = 16
[0x0000] 0x00000000 0x00000001 0x00000000 0x00000000
[0x0001] 0x00000000 0x00000000 0x00000000 0x00000000
[0x0002] 0x00000000 0x00000000 0x00000000 0x00000000
[0x0003] 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 0x00000000 0x00000000 0x00000000
[0x0101] 0x00000000 0x00000000 0x00000000 0x00000000
[0x0102] 0x03ec71de3 0x1a013e1a 0xbf2a01a0 0x00000000
[0x0103] 0x0f811111 0x55555555 0xbfc55555 0x00000000

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
[0x00c1] 0x00000000 0x00000000 0x00000000 0x00000000
[0x00c2] 0x00000000 0x00000000 0x00000000 0x00000000
[0x00c3] 0x0004321 0x00009876

There they are; the stack grows from the top. (The heap grows from the bottom.)
PRU> dd *0x2000*
Absolute addr = 0x2000, offset = 0x0000, Len = 16
    [0x2000] 0x0000deed 0x00000001 0x00000000 0x557fcfb5
    [0x2010] 0xce97bd0f 0x6afb2c8f 0xc7f35df4 0x5af6dcb
    [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.

PRU> *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-shiftlock kbd-ctrllock kbd-ctrllock kbd-cursorlock *[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-shiftlock kbd-ctrllock kbd-cursorlock kbd-cursorlock 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.

---

**4.2. PRU Cookbook**

**651**
BeagleBoard Docs, Release 0.0.20230323

1
2
3
4
5

#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";

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 4.81: write_init_pins.sh
1
2

3
4
5
6
7
8
9

#!/bin/bash
init_pins=$(readelf -x .init_pins $1 | grep 0x000 | cut -d' ' -f4-7 | xxd -r -p |␣
,→tr '\0' '\n' | paste - -)
while read -a line; do
if [ ${#line[@]} == 2 ]; then
echo writing \"${line[1]}\" to \"${line[0]}\"
echo ${line[1]} > ${line[0]}
sleep 0.1
fi
done <<< "$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.

652

Chapter 4. Books


Listing 4.82: pwm1.pru0.c

```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);
    }
}
```

To run this code you need to configure the pin muxes to output the PRU. If you are on the Black run

```bash
bone$ config-pin P9_31 pruout
```

On the Pocket run

```bash
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

```bash
bone$ export TARGET=pwm1.pru0
```

Now you are ready to compile

```bash
bone$ make
```

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 4.83: pwm1.pru0.c

```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 expansion of the c code.

Table 4.15: Line-by-line of pwm1.pru0.c

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Standard c-header include</td>
</tr>
<tr>
<td>2</td>
<td>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</td>
</tr>
<tr>
<td>3</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>This include has addresses for the GPIO ports and some bit positions for some of the headers.</td>
</tr>
</tbody>
</table>

Here’s what’s in resource_table_empty.h

Listing 4.84: resource_table_empty.c

```c
/*
 * define the resource table entries for all PRU cores. This will be
 * incorporated into corresponding base images, and used by the remotproc
 * on the host-side to allocated/reserve resources. Note the remotproc
 * 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_

(continues on next page)
```c
#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, /* reserved, must be zero */
    0, /* offset[0]*/
};

#endif /* _RSC_TABLE_PRU_H_ */
```

resource_table_empty.c

Table 4.16: Line-by-line (continued)

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-7</td>
<td>__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.</td>
</tr>
<tr>
<td>11</td>
<td>This line selects which GPIO pin to toggle. The table below shows which bits in __R30 map to which pins</td>
</tr>
<tr>
<td>14</td>
<td>CT_CFG.SYSCFG_bit.STANDBY_INIT is set to 0 to enable the OCP master port. More details on this and thousands of other registers see the TI AM335x TRM. Section 4 is on the PRU and section 4.5 gives details for all the registers.</td>
</tr>
</tbody>
</table>

Bit 0 is the LSB.
Table 4.17: Mapping bit positions to pin names

<table>
<thead>
<tr>
<th>PRU</th>
<th>Bit</th>
<th>Black pin</th>
<th>Pocket pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>P9_31</td>
<td>P1.36</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>P9_29</td>
<td>P1.33</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>P9_30</td>
<td>P2.32</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>P9_28</td>
<td>P2.30</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>P9_42b</td>
<td>P1.31</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>P9_27</td>
<td>P2.34</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>P9_41b</td>
<td>P2.28</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>P9_25</td>
<td>P1.29</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>P8_12(out) P8_16(in)</td>
<td>P2.24</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>P8_11(out) P8_15(in)</td>
<td>P2.33</td>
</tr>
</tbody>
</table>

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
1  9  P8_29
1  10 P8_28
1  11 P8_30
1  12 P8_21
1  13 P8_20
1  14 __R30
1  15 __R30
1  16 P9_26(in)

Table 4.18: Line-by-line (continued again)

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>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.</td>
</tr>
<tr>
<td>18</td>
<td>__delay_cycles is an ((instrinsic 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.</td>
</tr>
<tr>
<td>19</td>
<td>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.</td>
</tr>
</tbody>
</table>

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^!.
Fig. 4.150: Output of pwm1.pru0.c with 100,000,000 delays cycles giving a 1s period
You can see how fast the PRU can run by setting both of the `__delay_cycles` to 0. This results in the next figure.

![Output of pwm1.pru0c with 0 delay cycles](image)

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.

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.

```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;
```

(continues on next page)
Clear SYSCFG[STANDBY_INIT] to enable OCP master port

```c
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);
}
```

The output now looks like:

![Waveform](image)

**Fig. 4.152: Output of pwm2.pru0.c corrected delay**

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 its 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 0xA300000 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

```c
#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

```c
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

```c
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 4.86: pwm4.pru0.c

```c
#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.
#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.
```

(continues on next page)
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 {
            onCount[ch] = pru0_dram[2 * ch];  // Read from DRAM0
            offCount[ch] = pru0_dram[2 * ch + 1];
        }
    }
}

\textbf{Listing 4.87: pwm-test.c}

Here is code that runs on the ARM side to set the on and off time values.

\begin{verbatim}
/*
 * 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
#define PRU_LEN 0x80000  // Length of PRU memory
#define PRU0_DRAM 0x00000  // Offset to DRAM
#define PRU1_DRAM 0x02000
#define PRU0_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
\end{verbatim}

(continues on next page)
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.

---

```
unsigned pruDRAM_32int_ptr[2*(ch)+0] = countOn;  // On time
pruDRAM_32int_ptr[2*(ch)+1] = countOff;  // Off time
return 0;
}

int main(int argc, char *argv[])
{
    unsigned int  *pru;  // Points to start of PRU memory.
    int fd;
    printf("Servo tester\n");
    fd = open("/dev/mem", O_RDWR | O_SYNC);
    if (fd == -1) {
        printf("ERROR: could not open /dev/mem.\n\n");
        return 1;
    }
    pru = mmap (0, PRU_LEN, PROT_READ | PROT_WRITE, MAP_SHARED, fd, PRU_ADDR);
    if (pru == MAP_FAILED) {
        printf("ERROR: could not map memory.\n\n");
        return 1;
    }
    close(fd);
    printf("Using /dev/mem.\n");
    pru0DRAM_32int_ptr = pru + PRU0_DRAM/4 + 0x200/4;  // Points to...
    pru1DRAM_32int_ptr = pru + PRU1_DRAM/4 + 0x200/4;  // Points to...
    prusharedMem_32int_ptr = pru + PRU_SHAREDMEM/4;  // Points to start...
    int i;
    for(i=0; i<MAXCH; i++) {
        start_pwm_count(i, i+1, 20-(i+1));
    }
    if(munmap(pru, PRU_LEN)) {
        printf("munmap failed\n");
    } else {
        printf("munmap succeeded\n");
    }
}
pwm-test.c
```
Fig. 4.153: Four Channel PWM with ARM control
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.

---

```c
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"

#define PRU0_DRAM 0x00000 // Offset to DRAM
#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)
  }
```

(continues on next page)
The output of `pwm5.pru0.c` is in the figure below.

![Figure 4.154: pwm5.pru0.c Unrolled version of pwm4.pru0.c](image)

It's running about 6 times faster than `pwm4.pru0.c`.

<table>
<thead>
<tr>
<th>Measure</th>
<th>pwm4.pru0.c time</th>
<th>pwm5.pru0.c time</th>
<th>Speedup</th>
<th>pwm5.pru0.c w/o UNROLL</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period</td>
<td>3.06μs</td>
<td>510ns</td>
<td>6x</td>
<td>1.81μs</td>
<td>~1.7x</td>
</tr>
<tr>
<td>Width+</td>
<td>450ns</td>
<td>70ns</td>
<td>~6x</td>
<td>1.56μs</td>
<td>~3x</td>
</tr>
</tbody>
</table>

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 and 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 multichannel 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.

![Agilent Technologies](image)

Fig. 4.155: pwm5.pru0 Zoomed In

The solution is to declare `Rtmp` (line 35) which holds the value for `__R30`.

```c
// This code does MAXCH parallel PWM channels.
// All channels start at the same time. It's period is 510ns
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
(continues on next page)
```
```c
#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]--;
    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]; // Copy to DRAM0 so the...
    pru0_dram[2*ch+1] = off[ch]; // Interleave the on and off...
    onCount[ch] = on[ch];
    offCount[ch] = off[ch];
    Rtmp = __R30;
  }
  while (1) {
    update(0)
    update(1)
    update(2)
    update(3)
    __R30 = Rtmp;
  }
}
```

pwm6.pru0.c Sync'ed Version of pwm5.pru0.c

Each channel writes its 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.
Fig. 4.156: pwm6.pru0 Synchronized Channels
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*)

Listing 4.90: pwm7.pru0.c Using Both PRUs

```c
1 #include <stdint.h>
2 #include <pru_cfg.h>
3 #include "resource_table_empty.h"

4 #define PRUMAN 0
5
6 #define PRU0_DRAM 0x00000 // Offset to DRAM
7 // Skip the first 0x200 byte of DRAM since the Makefile allocates
8 // 0x100 for the STACK and 0x100 for the HEAP.
9 volatile unsigned int *pru0_dram = (unsigned int *) (PRU0_DRAM + 0x200);
10 #define MAXCH 2 // Maximum number of channels per PRU
11
12 #define update(ch) 
13 /*
14 if(onCount[ch]) {
15    onCount[ch]--;
16    Rtmp |= 0x1<<ch;
17 } else if(offCount[ch]) {
18    offCount[ch]--;
19    Rtmp &= ~(0x1<<(ch));
20 } else {
21    onCount[ch] = pru0_dram[2*ch];
22    offCount[ch]= pru0_dram[2*ch+1];
23 }
24 */
25 volatile register uint32_t __R30;
26 volatile register uint32_t __R31;

27 void main(void)
28 {
29    uint32_t ch;
30    uint32_t on[] = {1, 2, 3, 4};
31    uint32_t off[] = {4, 3, 2, 1};
32    uint32_t onCount[MAXCH], offCount[MAXCH];
33    register uint32_t Rtmp;
34
35    /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
36    CT_CFG.SYSCFG.Bit.STANDBY_INIT = 0;
37
38    #pragma UNROLL(MAXCH)
39    for(ch=0; ch<MAXCH; ch++)
40        pru0_dram[2*ch ] = on [ch+PRUMAN*MAXCH]; // Copy to DRAM0...
```

(continues on next page)
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;
   }
}

pwm7.pru0.c Using Both PRUs

Be sure to run pwm7_setup.sh to get the correct pins configured.

Listing 4.91: 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

pwm7_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*
/var/lib/cloud9/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_Black,TARGET=pwm7.
   -pru0
   - Stopping PRU 0

(continues on next page)
copying firmware file /tmp/cloud9-examples/pwm7.pru0.out to /lib/firmware/
write_init_pins.sh
- Starting PRU 0
  MODEL = TI_AM335x_BeagleBone_Black
  PROC = pru
  PRUN = 0
  PRU_DIR = /sys/class/remoteproc/remoteproc1
- pru1
- Stopping PRU 1
- copying firmware file /tmp/cloud9-examples/pwm7.pru1.out to /lib/firmware/
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.

```c
pru0_dram[2*ch ] = on [ch+PRUNUN*MAXCH];  // Copy to DRAM0 so the ARM can...
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 optimized 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 4.92: 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
```

(continues on next page)
Fig. 4.157: pwm7.pru0 Two PRUs running
Fig. 4.158: pwm7.pru0 Two PRUs stopped
```c
#include <stdint.h>
#include <pru_cfg.h>
#include <pru_intc.h>
#include <pru_ctrl.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;

// Initialize interrupts so the PRUs can be synchronized.
// 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_INT.CMR4.bit.CH_MAP_16 = 1; // Map event 16 to channel 1
    CT_INT.HMR0.bit.HINT_MAP_1 = 1; // Map channel 1 to host 1
    CT_INT.SICR = 16; // Ensure event 16 is cleared
    CT_INT.EISR = 16; // Enable event 16
    CT_INT.HIESR |= (1 << 0); // Enable Host interrupt 1
    CT_INT.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.GPFCFG0 = 0x00000; // Configure GPI and GPO as Mode 0 (Direct Connect)
    configIntc(); // Configure INTC
}
```

(continues on next page)
/* 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 DRAM
    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) {
    __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);
}

Listing 4.93: pwm8.pru1.c PRU 1 waiting for INTC from PRU 0

pwm8.pru0.c PRU 0 using INTC to send a signal to PRU 1

PRU 2's code waits for PRU 0 before going.

// This code does MAXCH parallel PWM channels on both PRU 0 and PRU 1
// 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) \n    if(onCount[ch]) { \n        onCount[ch]--; \n        Rtmp |= 0x1<<ch; \n    } else if(offCount[ch]) { \n        offCount[ch]--; \n        Rtmp &= ~0x1<<ch); \n    } else { \n        onCount[ch] = pru0_dram[2*ch]; \n        offCount[ch]= pru0_dram[2*ch+1]; \n    }
volatile register uint32_t __R30;
volatile register uint32_t __R31;
// Initialize interrupts so the PRUs can be synchronized.
// PRU1 is started first and then waits for PRU0
// PRU0 is then started and tells PRU1 when to start going
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) {
        while((__R31 & (0x1<<31))==0) { // Wait for PRU 0
            CT_INTC.SICR = 16; // Clear...
            --event 16
            __R30 = Rtmp;
            update(0);
            update(1);
        }
    }
}

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 4.20: pwm8.pru0.c changes from pwm7.pru0.c

<table>
<thead>
<tr>
<th>PRU</th>
<th>Line</th>
<th>Change</th>
</tr>
</thead>
</table>
| 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. |
Fig. 4.159: pwm8.pru0 PRUs sync

4.2. PRU Cookbook
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.

**Solution** You can use the __R31 register to read an input pin. Let’s use the following pins.

<table>
<thead>
<tr>
<th>Table 4.21: Input/Output pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>in</td>
</tr>
</tbody>
</table>

These values came from *Mapping bit positions to pin names.*

Configure the pins with `input_setup.sh`.

```bash
#!/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.

```c
//include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"

volatile register uint32_t __R30;
volatile register uint32_t __R31;
```

(continues on next page)
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
    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 4.96: 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

(continues on next page)
// Generate a periodic wave in an array of MAXT values

#define SAWTOOTH
for(i=0; i<MAXT; i++) {
    waveform[i] = i*100/MAXT;
}
#endif

#define TRIANGLE
for(i=0; i<MAXT/2; i++) {
    waveform[i] = 2*i*100/MAXT;
    waveform[MAXT-i-1] = 2*i*100/MAXT;
}
#endif

#define SINE
float gain = 50.0f;
float bias = 50.0f;
float freq = 2.0f * 3.14159f / MAXT;
for (i=0; i<MAXT; i++){
    waveform[i] = (uint32_t)(bias+gain*sin(i*freq));
}
#endif

/* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
CT_CFG.SYSCFG.bit.STANDBY_INIT = 0;

while (1) {
    // Generate a PWM signal whose duty cycle matches
    // the amplitude of the signal.
    for(i=0; i<MAXT; i++) {
        onCount = waveform[i];
        offCount = 100 - onCount;
        while(onCount--) {
            __R30 |= 0x1;                  // Set the GPIO pin to 1
        }
        while(offCount--) {
            __R30 &= ~(0x1);             // Clear the GPIO pin
        }
    }
}

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. 4.160: Continuous Sawtooth Waveform

Fig. 4.161: Sampled Sawtooth Waveform
Table 4.22: Line-by-line of sine.pru0.c

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-5</td>
<td>Standard c-header includes</td>
</tr>
<tr>
<td>7</td>
<td>Number for samples in one cycle of the analog waveform</td>
</tr>
<tr>
<td>8</td>
<td>Which waveform to use. We’ve defined SAWTOOTH, TRIANGLE and SINE, but you can define your own too.</td>
</tr>
<tr>
<td>10-11</td>
<td>Declaring registers pass:[__R30] and pass:[__R31].</td>
</tr>
<tr>
<td>15-16</td>
<td>onCount counts how many cycles the PWM should be 1 and offCount counts how many it should be off.</td>
</tr>
<tr>
<td>18</td>
<td>waveform[] stores the analog waveform being output.</td>
</tr>
<tr>
<td>21-24</td>
<td>SAWTOOTH is the simplest of the waveforms. Each sample is the duty cycle at that time and must therefore be between 0 and 100.</td>
</tr>
<tr>
<td>26-31</td>
<td>TRIANGLE is also a simple waveform.</td>
</tr>
<tr>
<td>32-39</td>
<td>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.</td>
</tr>
<tr>
<td>47</td>
<td>Here the for loop looks up each value of the generated waveform.</td>
</tr>
<tr>
<td>48,49</td>
<td>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.</td>
</tr>
<tr>
<td>50-52</td>
<td>Stay on for onCount cycles.</td>
</tr>
<tr>
<td>53-55</td>
<td>Now turn off for offCount cycles, then loop back and look up the next cycle count.</td>
</tr>
</tbody>
</table>

Unfiltered Sawtooth Waveform shows the output of the code.

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.

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 frequency superimposed on it. We are only using a simple first-order filter. You could lower the cutoff frequency 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.
Fig. 4.162: Unfiltered Sawtooth Waveform

Fig. 4.163: Low-Pass Filter Wiring Diagram
Fig. 4.164: Reconstructed Sawtooth Waveform

- Period(1): 163.5us
- Width(1): 71.5us
- Freq(1): 6.12kHz

FRI JUN 29 11:41:11 2018
1.00V/ 2.83V
0.0s 50.00%/
More FFT Settings Menu

Window Hanning
Span 200kHz
Center 6.10kHz
Scale 20dB/
Offset 30.0dBV

f(t) = FFT(Ch1)
FFT Sample Rate = 2.00MSa/s
Fig. 4.165: Reconstructed Sawtooth Waveform with Lower Cutoff Frequency
Fig. 4.166: Reconstructed Sawtooth Waveform with Higher Cutoff Frequency
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. \textit{Reconstructed Triangle Waveform} shows the output signal.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{reconstructed_triangle_waveform.png}
\caption{Reconstructed Triangle Waveform}
\end{figure}

And also the sine wave as shown in \textit{Reconstructed Sinusoid Waveform}.

Notice on the bottom plot the harmonics are much more suppressed.

Generating the sine waveform uses \texttt{floats}. This requires much more code. You can look in \texttt{/tmp/cloud9-examples/sine.pru0.map} to see how much memory is being used. \texttt{/tmp/cloud9-examples/sine.pru0.map} for Sine Wave shows the first few lines for the sine wave.

\begin{lstlisting}[language=C]
1 ****************************************************************************
2  PRU Linker Unix v2.1.5
3  ****************************************************************************
4  >> Linked Fri Jun 29 13:58:08 2018
5  OUTPUT FILE NAME:  </tmp/pru0-gen/sine1.out>
6  ENTRY POINT SYMBOL: "_c_int00_noinit_noargs_noexit" address: 00000000
7  MEMORY CONFIGURATION
8  (continues on next page)
\end{lstlisting}

\section*{4.2. PRU Cookbook}
Fig. 4.168: Reconstructed Sinusoid Waveform

Period(1): 163.5us
Width(1): 83.0us
Freq(1): 6.12kHz
SECTION ALLOCATION MAP

<table>
<thead>
<tr>
<th>output attributes/</th>
<th>section page</th>
<th>origin</th>
<th>length</th>
<th>input sections</th>
</tr>
</thead>
</table>
| .text:.c_int00*   | 0             | 00000000     | 00000014    | rspruv3_le.lib : boot_special.obj (.
|                   |               | 00000000     | 00000014    | text:.c_int00_noinit_noargs_noexit) |
| .text             | 0             | 00000014     | 000018ac    | rspruv3_le.lib : sin.obj (.text:sin) |
|                   |               | 00000388     | 00000314    | : frcmpyd.obj (.text:__
|                   |               | 0000069c     | 00000258    | -TI_frcmpyd) |
|                   |               | 000008f4     | 00000254    | : frcaddd.obj (.text:__
|                   |               | 00000b48     | 00000248    | -pruabi_mpyd) |
|                   |               |              |             | : addd.obj (.text:__
(continues on next page)
(continued from previous page)

```c

--pruab(add)
00000d90 000001c8 : mpyf.obj (.text:__
--pruabi_mpyf)
00000f58 00000100 : modf.obj (.text:modf)
--pruabi_gtd)
000110c 000000b0 : ged.obj (.text:__
--pruabi_ged)
00011bc 000000b0 : ltd.obj (.text:__
--pruabi_ltd)
000126c 000000b0 sine1.obj (.text:main)
--pruabi_rtspruv3_le.lib : frcmpyf.obj (.text:__
--pruabi_fixdu)
000131c 000000a8 rtspruv3_le.lib : frcmpyf.obj (.text:__
--pruabi_eqd)
0001500 000000a0 : round.obj (.text:__
--pruabi_renorm)
0000008c : renormd.obj (.text:__
--pruabi_fixdi)
00016a8 00000084 : fltid.obj (.text:__
--pruabi_cvtfd)
0001672c 00000078 : cvtfd.obj (.text:__
--pruabi_fltu)
00017a4 00000050 : fltu.obj (.text:__
--pruabi_mpyi)
0001820 0000002c : asri.obj (.text:__
--pruabi_subd)
000184c 00000024 : mpyi.obj (.text:__
--pruabi_mpyi)
0001870 00000020 : negd.obj (.text:__
--pruabi_trunc)
0001890 00000020 : trunc.obj (.text:__
--text:loader_exit)
00018b0 00000008 : exit.obj (.text:abort)
00018b8 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 rtspruv3_le.lib : sin.obj (.fardata:R$1)
00000100 00000040 --HOLE--

.resource_table *
1 00000140 00000014 sine1.obj (.resource_table:retain)
00000140 00000014

.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)
110  * 2 00026044 00000000 UNINITIALIZED
111 .creg.PRU_CFG.noload.far
112  * 2 00026044 00000000 NOLOAD SECTION
113 .creg.PRU_CFG.far
114  * 2 00026044 00000000 UNINITIALIZED
115
116 SEGMENT ATTRIBUTES
117
118 id tag seg value
119 --- --- --- -----
120 0 PHA_PAGE 1 1
121 1 PHA_PAGE 2 1
122
123 GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name
124
125 page address name
126 ---- ------- ----
127 0 000018b8 C$$EXIT
128 2 00026000 CT_CFG
129 abs 481cc000 __PRU_CREG_BASE_DCAN0
130 abs 481d0000 __PRU_CREG_BASE_DCAN1
131 abs 80000000 __PRU_CREG_BASE_DDR
132 abs 48040000 __PRU_CREG_BASE_DMTIMER2
133 abs 4a100000 __PRU_CREG_BASE_GEMAC
134 abs 4802a000 __PRU_CREG_BASE_I2C1
135 abs 4819c000 __PRU_CREG_BASE_I2C2
136 abs 40000000 __PRU_CREG_BASE_L3OCMC
137 abs 480c8000 __PRU_CREG_BASE_MBX0
138 abs 46000000 __PRU_CREG_BASE_MCASPD7_DMA
139 abs 48030000 __PRU_CREG_BASE_MCSP10
140 abs 481a0000 __PRU_CREG_BASE_MCSP11
141 abs 48060000 __PRU_CREG_BASE_MMXCHS0
142 abs 00026000 __PRU_CREG_BASE_PRU_CFG
143 abs 00000000 __PRU_CREG_BASE_PRU_DMEM_0_1
144 abs 00000200 __PRU_CREG_BASE_PRU_DMEM_1_0
145 abs 00030000 __PRU_CREG_BASE_PRU_IEP
146 abs 0002e000 __PRU_CREG_BASE_PRU_INTS
147 abs 00010000 __PRU_CREG_BASE_PRU_SHAREDMEM
148 abs 00028000 __PRU_CREG_BASE_PRU_UART
149 abs 48300000 __PRU_CREG_BASE_PWSMS0
150 abs 48302000 __PRU_CREG_BASE_PWSMS1
151 abs 48304000 __PRU_CREG_BASE_PWSMS2
152 abs 48318000 __PRU_CREG_BASE_RSPV0
153 abs 48310000 __PRU_CREG_BASE_RSPV13
154 abs 00032400 __PRU_CREG_BASE_RSPV21
155 abs 00032200 __PRU_CREG_BASE_RSPV27
156 abs 480ca000 __PRU_CREG_BASE_SPINLOCK
157 abs 49000000 __PRU_CREG_BASE_TPCC
158 abs 48022000 __PRU_CREG_BASE_UART1
159 abs 48024000 __PRU_CREG_BASE_UART2
160 abs 00000000 __PRU_CREG_DCAN0
161 abs 0000000e __PRU_CREG_DCAN1
162 abs 0000001f __PRU_CREG_DDR
163 abs 00000001 __PRU_CREG_DMTIMER2
164 abs 00000009 __PRU_CREG_GEMAC
165 abs 00000002 __PRU_CREG_I2C1
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>_c_int00_noinit_noargs_noexit</td>
<td></td>
</tr>
<tr>
<td>0x00000001</td>
<td>_stack</td>
<td></td>
</tr>
<tr>
<td>0x00000002</td>
<td>abort</td>
<td></td>
</tr>
<tr>
<td>0x00000003</td>
<td>main</td>
<td></td>
</tr>
<tr>
<td>0x00000004</td>
<td>pru_remoteproc_ResourceTable</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
<table>
<thead>
<tr>
<th>page address</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00000000</td>
<td>__c_int00_noinit_noargs_noexit</td>
</tr>
<tr>
<td>0 00000014</td>
<td>sin</td>
</tr>
<tr>
<td>0 00000388</td>
<td>__TI_frcmpyd</td>
</tr>
<tr>
<td>0 0000069c</td>
<td>__TI_frcaddd</td>
</tr>
<tr>
<td>0 000008f4</td>
<td>__pruabi_mpyd</td>
</tr>
<tr>
<td>0 00000b48</td>
<td>__pruabi_addd</td>
</tr>
<tr>
<td>0 00000d90</td>
<td>__pruabi_mpyf</td>
</tr>
<tr>
<td>0 00000f58</td>
<td>modf</td>
</tr>
<tr>
<td>0 00001058</td>
<td>__pruabi_gtd</td>
</tr>
<tr>
<td>0 0000110c</td>
<td>__pruabi ged</td>
</tr>
<tr>
<td>0 000011bc</td>
<td>__pruabi_ltd</td>
</tr>
<tr>
<td>0 0000126c</td>
<td>main</td>
</tr>
<tr>
<td>0 0000131c</td>
<td>__TI_frcmpyf</td>
</tr>
<tr>
<td>0 000013c4</td>
<td>__pruabi_fixdu</td>
</tr>
<tr>
<td>0 00001464</td>
<td>__pruabi_nround</td>
</tr>
<tr>
<td>0 00001500</td>
<td>__pruabi_eqd</td>
</tr>
<tr>
<td>0 00001590</td>
<td>__TI_renormd</td>
</tr>
<tr>
<td>0 0000161c</td>
<td>__pruabi_fixdi</td>
</tr>
<tr>
<td>0 000016a8</td>
<td>__pruabi_fltd</td>
</tr>
<tr>
<td>0 0000172c</td>
<td>__pruabi_cvtfd</td>
</tr>
<tr>
<td>0 000017a4</td>
<td>__pruabi_fltuf</td>
</tr>
<tr>
<td>0 000017f4</td>
<td>__pruabi_asri</td>
</tr>
<tr>
<td>0 00001820</td>
<td>__pruabi_subbd</td>
</tr>
<tr>
<td>0 0000184c</td>
<td>__pruabi_mpyi</td>
</tr>
<tr>
<td>0 00001870</td>
<td>__pruabi_negd</td>
</tr>
<tr>
<td>0 00001990</td>
<td>__pruabi_trunc</td>
</tr>
<tr>
<td>0 000018b0</td>
<td>abort</td>
</tr>
<tr>
<td>0 000018b8</td>
<td>C$$EXIT</td>
</tr>
<tr>
<td>1 00000000</td>
<td>_stack</td>
</tr>
<tr>
<td>1 00000100</td>
<td>__TI_STACK_END</td>
</tr>
<tr>
<td>1 00000140</td>
<td>pru_remoteproc_ResourceTable</td>
</tr>
<tr>
<td>2 00026000</td>
<td>CT_CFG</td>
</tr>
<tr>
<td>abs 00000000</td>
<td>__PRU_CREG_BASE_PRU_DMEM_0_1</td>
</tr>
<tr>
<td>abs 00000000</td>
<td>__PRU_CREG_PRU_INTC</td>
</tr>
<tr>
<td>abs 00000001</td>
<td>__PRU_CREG_DMTIMER2</td>
</tr>
<tr>
<td>abs 00000002</td>
<td>__PRU_CREG_I2C1</td>
</tr>
<tr>
<td>abs 00000003</td>
<td>__PRU_CREG_PRU_ECAP</td>
</tr>
<tr>
<td>abs 00000004</td>
<td>__PRU_CREG_PRU_CFG</td>
</tr>
<tr>
<td>abs 00000005</td>
<td>__PRU_CREG_MMCBS0</td>
</tr>
<tr>
<td>abs 00000006</td>
<td>__PRU_CREG_MCSPI0</td>
</tr>
<tr>
<td>abs 00000007</td>
<td>__PRU_CREG_PRU_UART</td>
</tr>
<tr>
<td>abs 00000008</td>
<td>__PRU_CREG_MCASPO_DMA</td>
</tr>
<tr>
<td>abs 00000009</td>
<td>__PRU_CREG_GEMAC</td>
</tr>
<tr>
<td>abs 0000000a</td>
<td>__PRU_CREG_RSVD10</td>
</tr>
<tr>
<td>abs 0000000b</td>
<td>__PRU_CREG_UART1</td>
</tr>
<tr>
<td>abs 0000000c</td>
<td>__PRU_CREG_UART2</td>
</tr>
<tr>
<td>abs 0000000d</td>
<td>__PRU_CREG_RSVD13</td>
</tr>
<tr>
<td>abs 0000000e</td>
<td>__PRU_CREG_DCAN0</td>
</tr>
<tr>
<td>abs 0000000f</td>
<td>__PRU_CREG_DCANI</td>
</tr>
<tr>
<td>abs 00000010</td>
<td>__PRU_CREG_MCSPI1</td>
</tr>
<tr>
<td>abs 00000011</td>
<td>__PRU_CREG_I2C2</td>
</tr>
<tr>
<td>abs 00000012</td>
<td>__PRU_CREG_PWMS0</td>
</tr>
<tr>
<td>abs 00000013</td>
<td>__PRU_CREG_PWMS1</td>
</tr>
<tr>
<td>abs 00000014</td>
<td>__PRU_CREG_PWMS2</td>
</tr>
<tr>
<td>abs 00000015</td>
<td>__PRU_CREG_RSVD21</td>
</tr>
<tr>
<td>abs 00000016</td>
<td>__PRU_CREG_MBX0</td>
</tr>
</tbody>
</table>

(continues on next page)
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 bytes 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 sensitive, 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.

![Image of NeoPixel Wiring](image)

Fig. 4.169: NeoPixel Wiring

Test your wiring with the simple code in neo1.pru0.c - Code to turn all NeoPixels's white which turns all pixels white.

Listing 4.98: neo1.pru0.c - Code to turn all NeoPixels’s white

```c
1 #include <stdint.h>
2 #include <pru_cfg.h>
3 #include "resource_table_empty.h"
4 #include "prugpio.h"
5
6 #define STR_LEN 24
7 #define oneCyclesOn 700/5 // Stay on 700ns
8 #define oneCyclesOff 800/5
9 #define zeroCyclesOn 350/5
10 #define zeroCyclesOff 600/5
11 #define resetCycles 60000/5 // Must be at least 50u, use 60u
12 #define gpio P9_29 // output pin
13
14 #define ONE
15 volatile register uint32_t __R30;
16 volatile register uint32_t __R31;
17
18 void main(void)
19 {
20     /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
21     CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
22 }
```

(continues on next page)
```c
uint32_t i;
for(i=0; i<STR_LEN*3*8; i++) {
  #ifdef ONE
    __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);
  #endif
  // Send Reset
    __R30 &= ~gpio; // Clear the GPIO pin
    __delay_cycles(resetCycles);
    __halt();
}
```

Discussion  NeoPixel bit sequence (taken from WS2812 Data Sheet) shows the following waveforms are used to send a bit of data.

![Sequence chart](image)

<table>
<thead>
<tr>
<th>Label</th>
<th>Time in ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0H</td>
<td>350</td>
</tr>
<tr>
<td>T0L</td>
<td>800</td>
</tr>
<tr>
<td>T1H</td>
<td>700</td>
</tr>
<tr>
<td>T1L</td>
<td>600</td>
</tr>
<tr>
<td>Treset</td>
<td>&gt;50,000</td>
</tr>
</tbody>
</table>

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.

![Agilent Technologies](image)

**Fig. 4.171: 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 4.99: neo2.pru0.c - Code to turn on green, red, blue

```c
// Control a ws2812 (neo pixel) display, green, red, blue, green, ...
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"
```

(continues on next page)
```c
#define STR_LEN 3
#define oneCyclesOn 700/5 // Stay on 700ns
#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)) { // Set the GPIO pin
                __R30 |= gpio;  // Set the GPIO pin
                __delay_cycles(oneCyclesOn-1);
                __R30 &= ~gpio; // Clear the GPIO pin
                __delay_cycles(oneCyclesOff-2);
            } else { // Set the GPIO pin
                __R30 |= gpio;  // Set the GPIO pin
                __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();
}
```

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.

**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 4.24: Line-by-line for neo2.pru0.c

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Define the string of colors to be output. Here the ordering of the bits is the same as NeoPixel data sequence, GRB.</td>
</tr>
<tr>
<td>26</td>
<td>Loop for each color to output.</td>
</tr>
<tr>
<td>27</td>
<td>Loop for each bit in an GRB color.</td>
</tr>
<tr>
<td>28</td>
<td>Get the i(^{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 (&amp;) 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.</td>
</tr>
<tr>
<td>29-32</td>
<td>Send a 1.</td>
</tr>
<tr>
<td>34-37</td>
<td>Send a 0.</td>
</tr>
<tr>
<td>42-43</td>
<td>Send a reset pulse once all the colors have been sent.</td>
</tr>
</tbody>
</table>

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 4.100: neo3.pru0.c - Code to animate a red pixel running around a ring of blue

```c
#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 SPEED 20000000/5 // Time to wait between updates

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
    uint32_t background = 0x00000f;
    uint32_t foreground = 0x000f00;
```

(continues on next 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
            } else {
               __R30 |= gpio;       // Set the GPIO pin to 1
               __delay_cycles(oneCyclesOff-2);
            }

            __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 4.25: Here’s the highlights.

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>32,33</td>
<td>Initialize the array of colors.</td>
</tr>
<tr>
<td>38-41</td>
<td>Update the array.</td>
</tr>
<tr>
<td>44-58</td>
<td>Send the array to the LEDs.</td>
</tr>
<tr>
<td>60-61</td>
<td>Send a reset.</td>
</tr>
<tr>
<td>64</td>
<td>Wait a bit.</td>
</tr>
</tbody>
</table>

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 4.101: `neo4.pru0.c - Code to talk to the PRU via rpmsg_pru`

```c
// 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)
```c
#define STR_LEN 24
#define oneCyclesOn 700/5 // Stay on for 700ns
#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, &resourceTable.rpmsg_vring0, &resourceTable.rpmsg_vring1, TO_ARM_HOST, FROM_ARM_HOST) != PRU_RPMSG_SUCCESS);
    /* 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;
        #else
        CT_INTC.SICR_bit.STS_CLR_IDX = FROM_ARM_HOST;
        #endif
    }
}
```

(continues on next page)
```c
#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
        r = strtol(&ret[1], NULL, 0);
        ret = strchr(&ret[1], ' '); // Skip
        g = strtol(&ret[1], NULL, 0);
        ret = strchr(&ret[1], ' '); // Skip
        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;
                    __delay_cycles(zeroCyclesOn-1);
                    __R30 &= ~out; // Clear the GPIO pin
                    __delay_cycles(zeroCyclesOff-14);
                }
            }
        }
    }
}
```

(continues on next page)
Run the code as usual.

```
bone$ make TARGET=neo4.pru0
   pru0
   - Stopping PRU 0
   - copying firmware file /tmp/cloud9-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\textsuperscript{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\textsuperscript{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>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>The \texttt{CHAN_NAME} of \texttt{rpmsg_pru} matches that \texttt{rpmsg_pru} driver that is already installed. This connects this PRU to the driver.</td>
</tr>
<tr>
<td>32</td>
<td>The \texttt{CHAN_PORT} tells it to use port 30. That’s why we use \texttt{/dev/rpmsg_pru30}</td>
</tr>
<tr>
<td>40</td>
<td>\texttt{payload[]} is the buffer that receives the data from the ARM.</td>
</tr>
<tr>
<td>42-48</td>
<td>Same as the previous NeoPixel examples.</td>
</tr>
<tr>
<td>52</td>
<td>\texttt{color[]} is the state to be sent to the LEDs.</td>
</tr>
<tr>
<td>66-68</td>
<td>\texttt{color[]} is initialized.</td>
</tr>
<tr>
<td>70-85</td>
<td>Here are a number of details needed to set up the channel between the PRU and the ARM.</td>
</tr>
<tr>
<td>88</td>
<td>Here we wait until the ARM sends us some numbers.</td>
</tr>
<tr>
<td>99</td>
<td>Receive all the data from the ARM, store it in \texttt{payload[]}.</td>
</tr>
<tr>
<td>101-111</td>
<td>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.</td>
</tr>
<tr>
<td>113</td>
<td>The NeoPixels want the data in GRB order. Shift and OR everything together.</td>
</tr>
<tr>
<td>116-133</td>
<td>If the \texttt{index} = -1, send the contents of \texttt{color} to the LEDs. This code is same as before.</td>
</tr>
</tbody>
</table>

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 4.102: neo-rainbow.py - A python program using
/dev/rpmsg_pru30

#!/usr/bin/python3
from time import sleep
import math
```
len = 24
amp = 12
f = 25
shift = 3
phase = 0

# Open a file
fo = open("/dev/rpmsg_pru30", "wb", 0)

while True:
    for i in range(0, len):
        r = (amp * (math.sin(2*math.pi*f*(i-phase-0*shift)/len) + 1)) + 1;
        g = (amp * (math.sin(2*math.pi*f*(i-phase-1*shift)/len) + 1)) + 1;
        b = (amp * (math.sin(2*math.pi*f*(i-phase-2*shift)/len) + 1)) + 1;
        fo.write(b"%d %d %d %d\n" % (i, r, g, b))
        fo.write(b"-1 0 0 0\n");
        phase = phase + 1
        sleep(0.05)

# Close opened file
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:

```c
#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\textsuperscript{st} 'line' and the 17\textsuperscript{th} 'line' (64 x 2 RGB LEDs = 128 RGB LEDs), the second is the 2\textsuperscript{nd} and
18th line, etc until the last section which is the 16th and 32nd line. You might be asking, why are the lines paired this way? wouldn’t it be nicer to have the first section be the 1st and 2nd line, then 3rd and 4th, until the 15th and 16th? 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


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 4.103: rgb_python.py - Python code for driving RGB LED matrix

```python
#!/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)
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 4.104: rgb_python_setup.sh
```bash
# 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`
cfgpin $pin out

Your display should look like Display running rgb_python.py.

Fig. 4.173: 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.
blocks_rgb1 is the PRU code to drive the RGB LED matrix. Be sure to run `bone$ source rgb_setup.sh` first.

### Listing 4.105: PRU code for driving the RGB LED matrix

```c
#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) {
        // 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...
        __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...
            // 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)
gpio[r11_gpio][GPIO_CLEARDATAOUT] = r11_pin | g11_pin | ~b11_pin;
___delay_cycles(DELAY);

// Bottom row green
gpio[r12_gpio][GPIO_CLEARDATAOUT] = r12_pin | b12_pin;
___delay_cycles(DELAY);
gpio[r12_gpio][GPIO_SETDATAOUT] = g12_pin;
___delay_cycles(DELAY);

__R30 | = pru_clock;  // Toggle clock
___delay_cycles(DELAY);
__R30 ^= ~pru_clock;
___delay_cycles(DELAY);
}
__R30 | = pru_oe;  // Disable display
___delay_cycles(DELAY);
__R30 | = pru_latch;  // Toggle latch
___delay_cycles(DELAY);
__R30 ^= ~pru_latch;
___delay_cycles(DELAY);
__R30 ^= ~pru_oe;  // Enable display
___delay_cycles(DELAY);
}

rgb1.pru0.c

The results are shown in Display running rgb1.c on PRU 0.

Fig. 4.174: 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 Pocket Beagle.

<table>
<thead>
<tr>
<th>J1 Pin</th>
<th>Connector</th>
<th>Pocket Heads</th>
<th>gpio port and bit number</th>
<th>Linux gpio number</th>
<th>PRU R30 bit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>P2_10</td>
<td>1-20</td>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>P2_06</td>
<td>1-25</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>P2_04</td>
<td>1-26</td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>P2_01</td>
<td>1-18</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>P2_32</td>
<td>3-16</td>
<td>112</td>
<td>PRU0.2</td>
<td></td>
</tr>
<tr>
<td>LC</td>
<td>P1_31</td>
<td>3-18</td>
<td>114</td>
<td>PRU0.4</td>
<td></td>
</tr>
<tr>
<td>CLK</td>
<td>P1_33</td>
<td>3-15</td>
<td>111</td>
<td>PRU0.1</td>
<td></td>
</tr>
<tr>
<td>OE</td>
<td>P1_29</td>
<td>3-21</td>
<td>117</td>
<td>PRU0.7</td>
<td></td>
</tr>
<tr>
<td>G1</td>
<td>P2_08</td>
<td>1-28</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td>P2_02</td>
<td>1-27</td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LB</td>
<td>P2_30</td>
<td>3-17</td>
<td>113</td>
<td>PRU0.3</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>P2_34</td>
<td>3-19</td>
<td>115</td>
<td>PRU0.5</td>
<td></td>
</tr>
<tr>
<td>LAT</td>
<td>P1_36</td>
<td>3-14</td>
<td>110</td>
<td>PRU0.0</td>
<td></td>
</tr>
</tbody>
</table>

The J1 mapping to gpio port and bit number comes from [https://github.com/FalconChristmas/fpp/blob/master/capes/pb/panels/PocketScroller.json](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/1FRGvYOyW1RiNSEvprvstf1AVeapnASgDXHtxeDOjgw/edit#gid=0](https://docs.google.com/spreadsheets/d/1FRGvYOyW1RiNSEvprvstf1AVeapnASgDXHtxeDOjgw/edit#gid=0).

**Oscilloscope display of CLK, OE, LAT and R1** shows four of the signal waveforms driving the RGB LED matrix.

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 comparision.

**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
Fig. 4.175: RGB Matrix J1 connector
Fig. 4.176: Oscilloscope display of CLK, OE, LAT and R1

4.2. PRU Cookbook
Fig. 4.177: FPP waveforms
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.


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 05blocks/code/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*
made [-C /lib/modules/4.9.88-ti-r111/build M=$PWD
make[1]: Entering directory `/usr/src/linux-headers-4.9.88-ti-r111'
   LD /home/debian/PRUCookbook/docs/05blocks/code/module/built-in.o
   CC [M] /home/debian/PRUCookbook/docs/05blocks/code/module/rpmsg_client_sample.o
   CC [M] /home/debian/PRUCookbook/docs/05blocks/code/module/rpmsg_pru.o
   Building modules, stage 2.
   MODPOST 2 modules
   CC /home/debian/PRUCookbook/docs/05blocks/code/module/rpmsg_client_sample.
   -> mod.o
   LD [M] /home/debian/PRUCookbook/docs/05blocks/code/module/rpmsg_client_sample.ko
   CC /home/debian/PRUCookbook/docs/05blocks/code/module/rpmsg_pru.mod.o
   LD [M] /home/debian/PRUCookbook/docs/05blocks/code/module/rpmsg_pru.ko
make[1]: Leaving directory `/usr/src/linux-headers-4.9.88-ti-r111'
bone$ *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.

---

**Listing 4.106: copyright.c**

```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.
 */
(continues on next page)```

---

4.2. PRU Cookbook 715
4.2.6 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 aways until you see:

```
Listing 4.107: /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 Open-Core Protocol (OCP) port.

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 4.108: gpio.pru0.c**

```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_
```

(continues on next page)
This code will toggle P9_11 on and off. Here’s the setup file.

Listing 4.109: setup.sh

```bash
#!/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
done
```

Notice in the code `config-pin set P9_11 to gpio, not pruout`. This is because are are using the OCP interface to the pin, not the usual PRU interface.

Set your exports and make.

```bash
bone$ *source setup.sh*
```
TARGET = gpio.pru0

...  

bone$ *make*

*/var/lib/cloud9/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_Black,TARGET=gpio.

− pru0

− Stopping PRU 0

− copying firmware file /tmp/cloud9-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 4.28: gpio.pru0.c line-by-line

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-5</td>
<td>Standard includes</td>
</tr>
<tr>
<td>5</td>
<td>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 <em>Mode7</em> 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.</td>
</tr>
<tr>
<td>10</td>
<td>Line 103 of prugpio.h defines the address offset from GPIO0 that will allow us to <em>clear</em> 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.</td>
</tr>
<tr>
<td>11</td>
<td>Line 104 of prugpio.h is like above, but for <em>setting</em> bits.</td>
</tr>
<tr>
<td>12</td>
<td>Using this offset of line 105 of prugpio.h lets us just read the bits without changing them.</td>
</tr>
<tr>
<td>13</td>
<td>This shifts 0x1 to the 30^th^ bit position, which is the one corresponding to P9_11.</td>
</tr>
<tr>
<td>15</td>
<td>Here we initialize gpio0 to point to the start of GPIO port 0’s control registers.</td>
</tr>
<tr>
<td>18</td>
<td>gpio0[GPIO_SETDATAOUT] refers to the SETDATAOUT register of port 0. Writing to this register turns on the bits where 1’s are written, but leaves alone the bits where 0’s are.</td>
</tr>
<tr>
<td>19</td>
<td>Wait 100,000,000 cycles, which is 0.5 seconds.</td>
</tr>
<tr>
<td>20</td>
<td>This is line 18, but the output bit is set to 0 where 1’s are written.</td>
</tr>
</tbody>
</table>

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.
Fig. 4.179: gpio.pru0.c with pass:[__]delay_cycles(0)
The period is 80ns which is 12.0MHz. That’s about one forth the speed of the pass: [__] R30 method, but still not bad.

If you are using an oscilloscope, look closely and you’ll see the following.

![Agilent Technologies](image)

Fig. 4.180: PWM with jitter

The PRU is still as solid as before in it’s 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 happensthe 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 uiio. I find

```
### pru_uiio (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.
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.

**4.2.7 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 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 4.110: delay-test.pru0.c

```c
// Shows how to call an assembly routine with one parameter
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
#include "prugpio.h"

// The function is defined in delay.asm in same dir
// We just need to add a declaration here, the definition can be
// seperately 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 4.111: delay.pru0.asm

```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
        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 4.112: Makefile

```makefile
include /var/lib/cloud9/common/Makefile

Makefile

This Makefile includes a common Makfile at /var/lib/cloud9/common/Makefile, this the Makefile you need to edit. Edit /var/lib/cloud9/common/Makefile and go to line 195.

```makefile
$(GEN_DIR)/%.out: $(GEN_DIR)/%.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.

```sh
bone$ *config-pin P9_31 pruout*
bone$ *make TARGET=delay-test.pru0 TARGETasm=delay.pru0* 
/var/lib/cloud9/common/Makefile:29: MODEL=TI_AM335x_BeagleBone_Black,TARGET=delay-
--test.pru0
- Stopping PRU 0
- copying firmware file /tmp/cloud9-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()`.

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>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Declare <code>my_delay_cycles</code> to be global so the linker can find it.</td>
</tr>
<tr>
<td>4</td>
<td>Label the starting point for <code>my_delay_cycles</code>.</td>
</tr>
<tr>
<td>5</td>
<td>Label for our delay loop.</td>
</tr>
<tr>
<td>6</td>
<td>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.</td>
</tr>
<tr>
<td>7</td>
<td>qbne is a quick branch if not equal.</td>
</tr>
<tr>
<td>9</td>
<td>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.</td>
</tr>
</tbody>
</table>

`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.
Fig. 4.181: Output of my_delay_cycles()
## 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 4.113: delay-test2.pru0.c

```c
#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 seperately linked
extern uint32_t my_delay_cycles(uint32_t);

uint32_t ret;

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();
        __R30 |= -gpio; // Clear the GPIO pin
        ret = my_delay_cycles();
    }
}
```

delay-test2.pru0.c

delay2.pru0.asm is the assembly code.

#### Listing 4.114: delay2.pru0.asm

```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
        ldi delay, r14, TEST ; TEST is defined in delay-
        sub r14, r14, 1
```

(continues on next page)
delay2.pru0.asm

An additional feature is shown inline 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 4.115: cycle.pru0.c - Code to count cycles.

```c
// 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;

    PRU0_CTRL.CTRL.bit.CTR_EN = 1;  // Enable cycle counter
    __R30 |= gpio;                  // Set the GPIO pin to 1

    // Reset cycle counter, cycle is on the right side to force the compiler
    // to put it in its own register
    PRU0_CTRL.CYCLE = cycle;
    __R30 ^= gpio;                  // Clear the GPIO pin

    cycle = PRU0_CTRL.CYCLE;        // Read cycle and store in a register
    stall = PRU0_CTRL.STALL;        // Ditto for stall

    __halt();
}
```

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 4.30: Line-by-line for cycle.pru0.c

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Include needed to reference <code>CYCLE</code> and <code>STALL</code>.</td>
</tr>
<tr>
<td>16</td>
<td>Declaring <code>cycle</code> and <code>stall</code>. The compiler will optimize these and just keep them in registers. We'll have to look at the <code>cycle.pru0.lst</code> file to see where they are stored.</td>
</tr>
<tr>
<td>21</td>
<td>Enables <code>CYCLE</code>.</td>
</tr>
<tr>
<td>26</td>
<td>Reset <code>CYCLE</code>. It ignores the value assigned to it and always sets it to 0. <code>cycle</code> is on the right hand side to make the compiler give it its own register.</td>
</tr>
<tr>
<td>28, 29</td>
<td>Reads the <code>CYCLE</code> and <code>STALL</code> values into registers.</td>
</tr>
</tbody>
</table>

You can see where `cycle` and `stall` are stored by looking into `/tmp/cloud9-examples/cycle.pru0.lst Lines 113..119.

```
Listing 4.116: /tmp/cloud9-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 00000000c 200080240002C0 LDI32 r0, 0x00022000 ; [ALU_, → PRU] |23| $0C1
118 107 00000014 000000F1002081 LBBO &r1, r0, 0, 4 ; [ALU_, → PRU] |23|
119 108 00000018 000000F03E1E1 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/cloud9-examples/cycle.pru0.lst Lines 146..152.

```
Listing 4.117: /tmp/cloud9-examples/cycle.pru0.lst Lines 146..152
```

```
146 129;---------------------------------------------------------------------
147 130; 30 | cycle = PRU0_CTRL.CYCLE; // Read cycle and store in a_
148 131;---------------------------------------------------------------------
149 132 00000002c 000000F10C2081 LBBO &r1, r0, 12, 4 ; [ALU_, → PRU] |30| $0C1
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.

```
bones *sudo prudebug*
PRU0> *r*
r
```

Register info for PRU0
```
Control register: 0x00000009
    Reset PC:0x0000 STOPPED, FREE_RUN, COUNTER_ENABLED, NOT_SLEEPING, PROC_, → DISABED
    Program counter: 0x0012
```

(continues on next page)
Current instruction: HALT

R00: *0x00000005*  R08: 0x00000000  R16: 0x0000003c6  R24: 0x00110210
R01: *0x00000003*  R09: 0x00000200  R17: 0x00000000  R25: 0x00000000
R02: 0x000000fc  R10: 0xfff4ea57  R18: 0x000003c6  R26: 0x6e616843
R03: 0x0004272c  R11: 0x5fac6373  R19: 0x30203020  R27: 0x206c656e
R04: 0xffffffff  R12: 0x59bfeafa  R20: 0x0000000a  R28: 0x00003033
R05: 0x00000007  R13: 0xa4c19eaf  R21: 0x00757270  R29: 0xa03f9990
R06: 0xefd30a00  R14: 0x00000005  R22: 0x00000000  R30: 0xe6616843
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 - Transfering 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 4.118: xout.pru0.c

```c
#include <stdint.h>
#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.SECRO = 0xFFFFFFFF;
    CT_INTC.SECR1 = 0xFFFFFFFF;

    /* Load the buffer with default values to transfer */
    dmemBuf.reg5 = 0xDEADBEEF;
    (continues on next page)
```
dmemBuf.reg6 = 0xAAAAAAAA;
dmemBuf.reg7 = 0x12345678;
dmemBuf.reg8 = 0xBBBBBBBB;
dmemBuf.reg9 = 0x87654321;
dmemBuf.reg10 = 0xC0000000;

/* 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 4.119: 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"

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_SLEEPSING, PROC_DISABLED
  Program counter: 0x0026
    Current instruction: HALT
            R00: 0x00000012    R08: 0xbbbbbbbb*    R16: 0x0000003c6    R24: 0x00110210
            R01: 0x0000200000*    R09: 0x87654321*    R17: 0x000000000    R25: 0x000000000
            R02: 0x0000000e4*    R10: 0xcccccccc*    R18: 0x00000036e    R26: 0x06e616843
            R03: 0x0004272c    R11: 0x5fac6373    R19: 0x30203020    R27: 0x206c656e
            R04: 0xffffffff    R12: 0x59bfeafc    R20: 0x00000000a    R28: 0x000000333
*R05: 0xdeadbeef*    R13: 0xa4c19eaf    R21: 0x00757270    R29: 0x02100000
*R06: 0xaaaaaaaaaa*    R14: 0x000000005    R22: 0x00000001e    R30: 0xa03f9990
*R07: 0x12345678*    R15: 0x000000003    R23: 0x000000000    R31: 0x000000000

PRU0> *pru 1*
pru 1
Active PRU is PRU1.

PRU1> *r*
r
Register info for PRU1
  Control register: 0x00000001
    Reset PC:0x0000 STOPPED, FREE_RUN, COUNTER_DISABLED, NOT_SLEEPSING, PROC_DISABLED
  Program counter: 0x000b
    Current instruction: HALT
            R00: 0x000000100*    R08: 0xbbbbbbbb*    R16: 0x0e9da228b    R24: 0x28113189
            R01: 0x0000200000*    R09: 0x87654321*    R17: 0x66621777    R25: 0x07dd29ab1
            R02: 0x0000000e4*    R10: 0xcccccccc*    R18: 0x661f83ea    R26: 0xcf1cd4a5
            R03: 0x00044db97    R11: 0xdec387d5    R19: 0xa85ad878    R27: 0x70a2f202
            R04: 0x9a9e496f    R12: 0xbeac3878    R20: 0x048ffff2    R28: 0x7465f510
*R05: 0xdeadbeef*    R13: 0x5777b488    R21: 0xa392977c7    R29: 0x0aab9b530
*R06: 0xaaaaaaaaaa*    R14: 0xffa60550    R22: 0x99fb123e    R30: 0x52c42a0d
*R07: 0x12345678*    R15: 0xdeb2142d    R23: 0xa53129d    R31: 0x00000000

Discussion    xout.pru0.c Line-by-line shows the line-by-line for xout.pru0.c
Table 4.31: xout.pru0.c Line-by-line

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A different resource so PRU 0 can receive a signal from PRU 1.</td>
</tr>
<tr>
<td>9-16</td>
<td>dmemBuf holds the data to be sent to PRU 1. Each will be transferred to its corresponding register by xout().</td>
</tr>
<tr>
<td>21-22</td>
<td>Define the interrupts we’re using.</td>
</tr>
<tr>
<td>27-28</td>
<td>Clear the interrupts.</td>
</tr>
<tr>
<td>31-36</td>
<td>Initialize dmemBuf with easy to recognize values.</td>
</tr>
<tr>
<td>40</td>
<td>Wait for PRU 1 to signal.</td>
</tr>
<tr>
<td>45</td>
<td>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.</td>
</tr>
<tr>
<td>48</td>
<td>Clear the interrupt so it can go again.</td>
</tr>
</tbody>
</table>

Table 4.32: xin.pru1.c Line-by-line

<table>
<thead>
<tr>
<th>Line</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-15</td>
<td>Place to put the received data.</td>
</tr>
<tr>
<td>26</td>
<td>Signal PRU 0</td>
</tr>
<tr>
<td>30</td>
<td>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.</td>
</tr>
</tbody>
</table>

If you really need speed, considering using pass: [___] xout() and pass: [___] xin() in assembly.

Copyright

Listing 4.120: 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
 (continues on next page)
4.2.8 Moving to the BeagleBone AI

So far all our examples have focussed mostly on the BeagleBone Black and Pocket Beagle. 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.

Discussion 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>
<thead>
<tr>
<th>PRU 0</th>
<th>Bit 0</th>
<th>Black pin P9_31</th>
<th>AI PRU1 pin</th>
<th>AI PRU2 pin P8_44</th>
<th>Pocket pin P1.36</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>P9_29</td>
<td>P8_41</td>
<td></td>
<td>P1.33</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>P9_30</td>
<td>P8_42/P8_21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>P9_28</td>
<td>P8_12</td>
<td>P8_39/P8_20</td>
<td>P2.30</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>P9_92</td>
<td>P8_11</td>
<td>P8_40/P8_25</td>
<td>P1.31</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>P9_27</td>
<td>P9_15</td>
<td>P8_37/P8_24</td>
<td>P2.34</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>P9_91</td>
<td></td>
<td>P8_38/P8_5</td>
<td>P2.28</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>P9_25</td>
<td></td>
<td>P8_36/P8_6</td>
<td>P1.29</td>
</tr>
</tbody>
</table>

continues on next page
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.

```bash
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.

```bash
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

(continues on next page)
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_prul_gpi7
  P8.15b       109  A3  d  fast  down  pr1_prul_gp016
  P8.16        111  B4  d  fast  down  pr1_prul_gp018
  P8.18        98   F5  c  fast  pr1_prul_gpi5
  P8.19        99   E6  c  fast  pr1_prul_gpi6
  P8.26       110  B3  d  fast  down  pr1_prul_gp017
  P9.16        108  C5  d  fast  down  pr1_prul_gp015
  P9.19b       95   F4  c  fast  up  pr1_prul_gpi2
  P9.20b       94   D2  c  fast  up  pr1_prul_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...
```

**Solution** The pins on the AI must be configured at boot time and therefore cannot be configured with config-pin. Instead you must edit the device tree.
**Discussion** 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
→ 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*  pycrl
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
...  
5 DTC   src/arm/am335x-s150.dtb
4 DTC   src/arm/am5729-beagleboneai.dtb
5 DTC   src/arm/am335x-nano.dtb
...  
7 bone$ sudo make install
...
9 'src/arm/am5729-beagleboneai.dtb' -> '/boot/dtbs/4.14.108-ti-r131/am5729-
→ beagleboneai.dtb'
...
11 bone$ reboot
...  
13 bone$ *show-pins.pl -v | sort | grep -i pru*
14 P8.13  100 D3 c fast rx  pr1_pru1_gpi7
15 P8.15b  109 A3 d fast  down pr1_pru1_gpo16
16 P8.16  111 B4 d fast  down pr1_pru1_gpo18
17 P8.18  98 F5 c fast rx  pr1_pru1_gpi5
18 P8.19  99 E6 c fast rx  pr1_pru1_gpi6
19 P8.26  110 B3 d fast  down pr1_pru1_gpo17
20 P9.16  108 C5 d fast  down pr1_pru1_gpo15
```

(continues on next page)
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.

**Discussion** 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.

```
#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 */
  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);
  }
}
```

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.

### 4.2.9 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:**
- http://beagleboard.org/CapeContest/entries/BeagleBone+DMX+Cape/
- 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:**
- http://beagleboard.org/CapeContest/entries/Interacto/
- http://www.hitchhikeree.org/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:**
- 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:**

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
- https://github.com/mranostay/beaglebone-telemetry-presentation

Servo Controller Foosball Table

**Description:** Used for ball tracking and motor control

**Type:** Project Tutorial Code Library

**References:**
- https://docs.google.com/spreadsheet/pub?key=0AmI_ryMKXUGJdDQ3LXB4X3VBW1pxQTFWbGh6RGJHUEE&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:**
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/

Robotic Control

Description: Chubby SpiderBot

Type: Project Code Library Project Reference

References:
  - http://www.youtube.com/watch?v=dEes9k7-DYY
  - https://github.com/cagdasc/Chubby1_v1
  - http://www.youtube.com/watch?v=JXyewd98e9Q

Software UART

Description: Soft-UART implementation on the PRU of AM335x

Type: Code Library Reference

References:

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:**

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 a movie 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:**
- 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 Acquisition

**Description:** Reading data at high speeds

**Type:** Reference

**References:**

Prufh (PRU Forth)

**Description:** Forth Programming Language and Compiler. It consists of a compiler, the forth system itself, and an optional program for loading and communicating with the forth code proper.

**Type:** Compiler

**References:**
- [https://github.com/biocode3D/prufh](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](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:**

BeagleLogic

**Description:** 100MHz 14channel logic analyzer using both PRUs (one to capture and one to transfer the data)

**Type:** Project

**References:**
- [http://beaglelogic.net](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](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](https://github.com/deepakkarki/pruspeak)