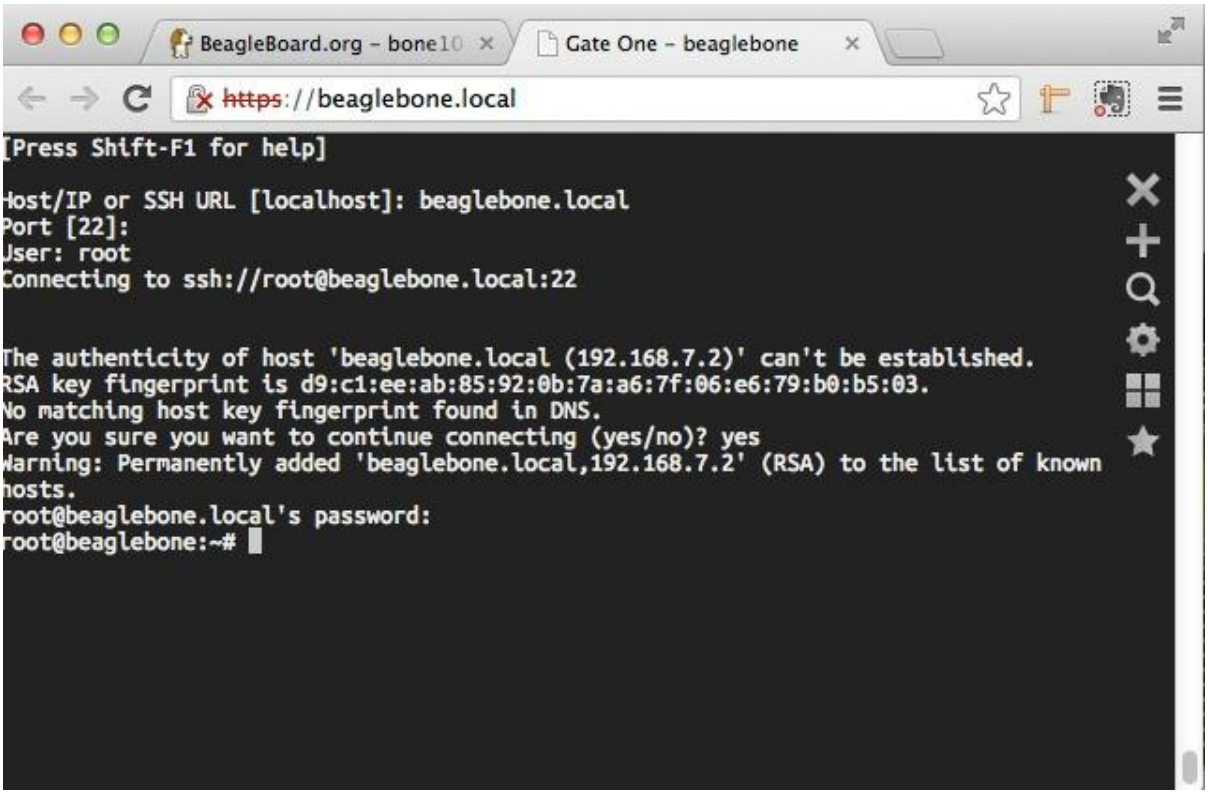
Setting up IO Python Library on BeagleBone Black

Created by Justin Cooper



https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black

Last updated on 2021-11-15 06:01:31 PM EST

Table of Contents

Overview

The BeagleBone Black is unique in that it has quite a few pins that are available on easy to use pin headers, as well as being a fairly powerful little system. There are 2 x 46 pins available (well, not all of them are, but we'll get to that later) to use.

Some of the functionality that is available:

- 7 Analog Pins
- 65 Digital Pins at 3.3V
- 2x I2C
- 2x SPI
- 2x CAN Bus
- 4 Timers
- 4x UART
- 8x PWM
- A/D Converter

Quite the feature list! The Adafruit BeagleBone IO Python library doesn't support all of them, but we hope to add more as we go.

The next pages will guide you through installing the library, as well as basic usage to get you started.

This tutorial is written for Angstrom, Ubuntu and Debian installations only at this time.

Installation on Angstrom

Installing the Adafruit-BeagleBone-IO-Python (phew!) library is fairly simple. Let's make sure we have a good foundation setup first.

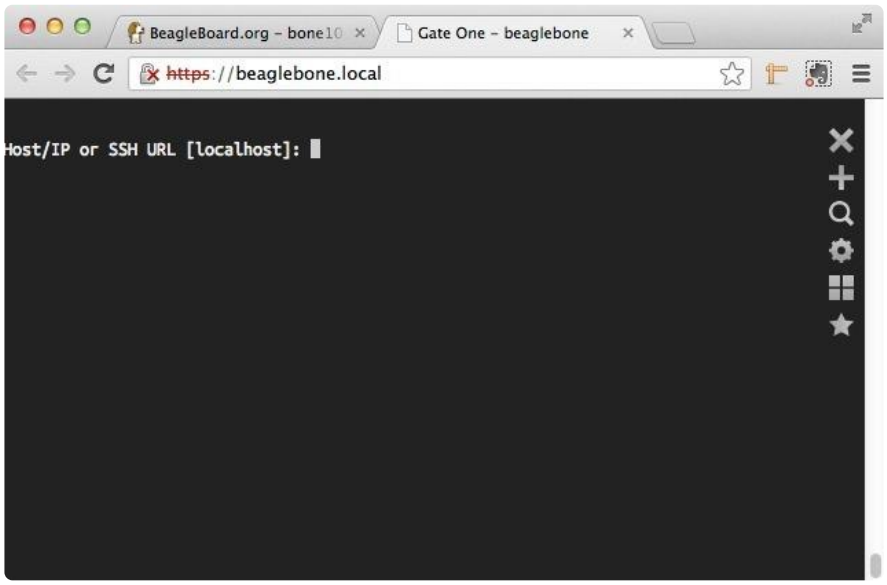
The most important part here is that you are using a Linux distribution with the 3.8 kernel. This kernel version made some fairly significant changes with how GPIO and PWM is accessed. The good news is that your BeagleBone Black came pre-installed with the proper kernel. It just may not be the latest and greatest. If you have some extra time, it may not be a bad idea to follow our installation guide for Angstrom (https://adafru.it/cg2), and flash your BeagleBone Black with the latest version.

Connecting to your BeagleBone Black (SSH)

Once you have the latest version of Angstrom on your BBB, let's ssh into the system so we can execute commands. The easiest way to gain access to the system is by using GateOne SSH. You can easily access GateOne by typing in the following into your browser window:

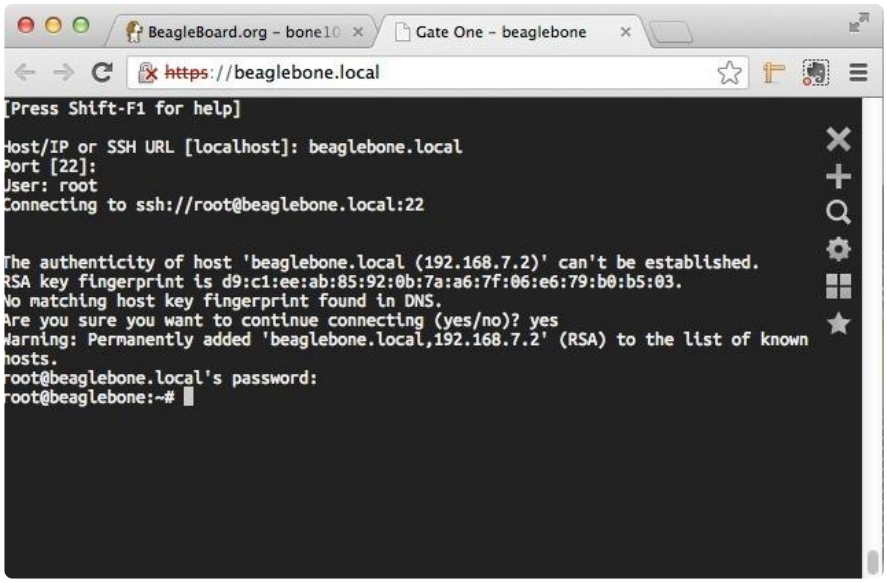
```
http://beaglebone.local
```

Once the page loads successully (you should see a green box that says "Your board is connected!"), you can click on the "GateOne SSH link to the upper left, in the sidebar. Then, click the "GateOne SSH client" link to get started. Some browsers may complain about invalid certificates, but you can proceed anyways.



To sign into the beaglebone, type the following at the prompts (assuming root user on a fresh Angstrom installation):

```
Host/IP or SSH URL [localhost]:  beaglebone.local
Port [22]:  (just hit enter)
User: root
Connecting to ssh://root@beaglebone.local:22
```



Commands to setup and install Adafruit_BBIO

Now that you're connected to the BBB, you'll want to start with setting the date and time so that it's accurate. Copy and paste the following into your terminal (you may want to make it execute this on startup in the future):

```
/usr/bin/ntpdate -b -s -u pool.ntp.org
```

These commands will require internet access. If you get errors, please view the FAQ page for resolutions.

Next, execute each of the following lines. Copy and paste the following one-by-one into the terminal, and hit enter:

```
opkg update &amp;amp;&amp;amp; opkg install python-pip python-setuptools python-smbus
pip install Adafruit_BBIO
```

Test your Installation (optional)

You can optionally test if your installation was successful by simply trying to load one of the modules. Execute the following command from the console (not from within the python interpretor), it shouldn't throw any errors, but return one line:

```
python -c "import Adafruit_BBIO.GPIO as GPIO; print GPIO"

#you should see this or similar:
```

```
&lt;module 'Adafruit_BBIO.GPIO' from '/usr/local/lib/python2.7/dist-packages/
Adafruit_BBIO/GPIO.so'&gt;
```

You can also validate by executing the 'python' command to enable the interpreter, and run the following code (you can tell you're in the right place when you see the ">>>" in your terminal):

```
import Adafruit_BBIO.GPIO as GPIO; print GPIO

#you should see this or similar:
&lt;module 'Adafruit_BBIO.GPIO' from '/usr/local/lib/python2.7/dist-packages/
Adafruit_BBIO/GPIO.so'&gt;
```

Manual Installation (optional)

You can also install Adafruit_BBIO by cloning the git repository. The following commands should get it installed as well:

```
git clone git://github.com/adafruit/adafruit-beaglebone-io-python.git
#set the date and time
/usr/bin/ntpdate -b -s -u pool.ntp.org
#install dependency
opkg update &amp;&amp; opkg install python-distutils python-smbus
cd adafruit-beaglebone-io-python
python setup.py install
```

Installation on Debian and Ubuntu

The majority of this library will need to be run as sudo in Debian and Ubuntu.

Installing the Adafruit-BeagleBone-IO-Python (phew!) library is fairly simple. Let's make sure we have a good foundation setup first.

The most important part here is that you are using a Linux distribution with the 3.8 kernel. This kernel version made some fairly significant changes with how GPIO, PWM and ADC are accessed.

Connecting to your BeagleBone Black (SSH)

Let's ssh into the system so we can execute commands. Open your favorite terminal, and SSH into your BeagleBone Black (BBB). Note, Ubuntu does not come with Avahi-Daemon pre-installed. This means you need to use the IP address to connect and not the hostname.

```
ssh ubuntu@your.bbb.ip.address
```

Enter the the password (default is 'temppwd' most likely). You should now have a prompt available to enter commands.

Commands to setup and install BBIO

Now that you're connected to the BBB, you'll want to start with setting the date and time so that it's accurate. Copy and paste the following into your terminal (you may want to make it execute this on startup in the future):

```
sudo ntpdate pool.ntp.org
```

Next install the dependencies:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-setuptools python-pip python-smbus -y
```

Depending on which version of Debian or Ubuntu you have installed, you may need a patched version of dtc.

The patched version of dtc (device-tree-compiler) includes the ability to compile overlays. The Adafruit_BBIO library compiles a set of overlays for SPI and UART. If you have no use for SPI or UART, you can skip this step of upgrading and patching dtc.

You can test dtc by following this guide (https://adafru.it/d8k). If you're not comfortable following the guide, it shouldn't cause any issues to just install the patched version of dtc.

You can view the overlays in our Github repository (https://adafru.it/clo).

You can find the patched version of dtc with instructions here (https://adafru.it/Ccw).

Once you've determined if you need the patched version of dtc, and installed it, execute the command to install BBIO:

```
sudo pip install Adafruit_BBIO
```

Test your Installation (optional)

You can optionally test if your installation was successful by simply trying to load one of the modules. Execute the following command from the console (not from within the python interpretor), it shouldn't throw any errors, but return one line:

```
sudo python -c "import Adafruit_BBIO.GPIO as GPIO; print GPIO"
```

```
#you should see this or similar:
&lt;module 'Adafruit_BBIO.GPIO' from '/usr/local/lib/python2.7/dist-packages/
Adafruit_BBIO/GPIO.so'&gt;
```

You can also validate by executing the 'python' command to enable the interpreter, and run the following code (you can tell you're in the right place when you see the ">>>" in your terminal):

```
import Adafruit_BBIO.GPIO as GPIO; print GPIO

#you should see this or similar:
&lt;module 'Adafruit_BBIO.GPIO' from '/usr/local/lib/python2.7/dist-packages/
Adafruit_BBIO/GPIO.so'&gt;
```

Manual Installation (optional)

You can also install BBIO by cloning the git repository. The following commands should get it installed as well:

```
sudo ntpdate pool.ntp.org
sudo apt-get update
sudo apt-get install build-essential python-dev python-pip python-smbus -y
git clone git://github.com/adafruit/adafruit-beaglebone-io-python.git
cd adafruit-beaglebone-io-python
sudo python setup.py install
cd ..
sudo rm -rf adafruit-beaglebone-io-python
```

Using the Adafruit_BBIO Library

This library has quite a few changes being made to it. Please read the CHANGELOG anytime you update the library to ensure it doesn't break your programs.

Using the Adafruit_BBIO library with the BeagleBone Black (BBB) is fairly simple, especially if you're familiar with the RPi.GPIO library for the Raspberry Pi.

To start, you'll want to import the library. There are two different options at this time to import. The first one is for GPIO:

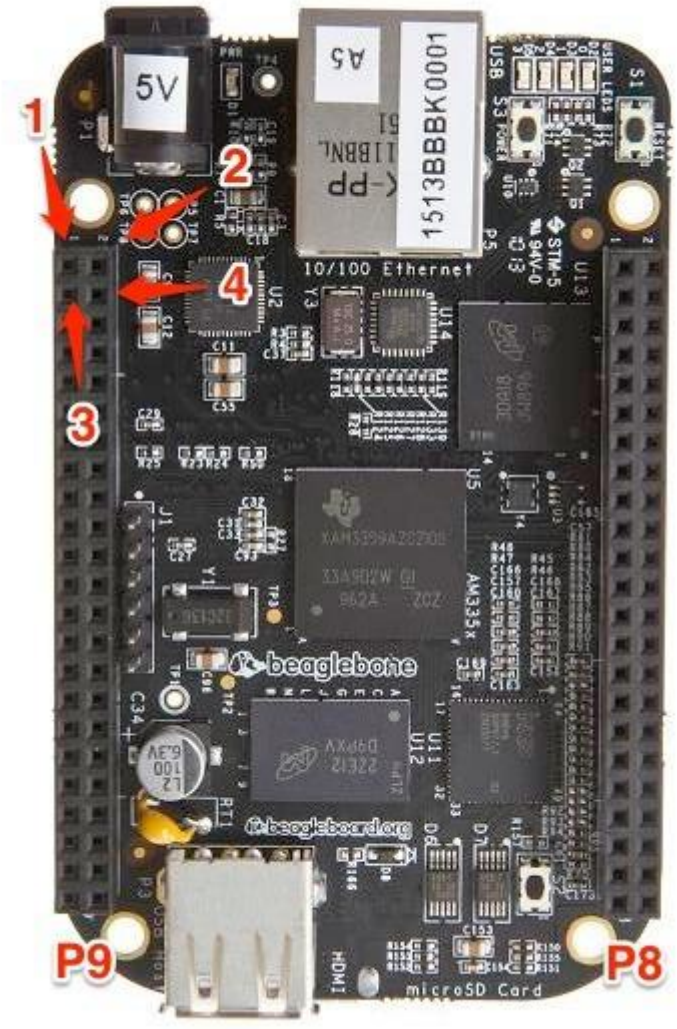
```
import Adafruit_BBIO.GPIO as GPIO
```

If you'd like to use PWM, then import as follows:

```
import Adafruit_BBIO.PWM as PWM
```

You can access the channels by either referencing the pin "key" or the name. If you look at your BeagleBone Black, you can see that each set of pin headers has a name, either P8 or P9. Then, you can see that there are pin numbers that start from 1, and go to 46.

When you count the pins, you don't go length-wise, but start at 1, then across to 2, and then back up to the next pin 3, and so on. The following image illustrates this a bit better:



So, to access the first pin on P9, you'd use "P9_1". You can also use the name of the pin to access it, which would be . You wouldn't want to do this though, as P9_1 is actually gnd! You'll want to view the last page of this guide to see which pins are available to use.

Not all pins are necessarily available. HDMI, and the eMMC flash module take up quite a few of them by default.

I2C is only compatible with Python2 due to the python-smbus dependency.

GPIO

Below are a few examples of using the Adafruit_BBIO.GPIO module. It's fairly simple to use.

You may need to run this library with sudo, particularly on Ubuntu.

Setup

To setup a digital pin as an output, set the output value HIGH, and then cleanup after you're done:

```
import Adafruit_BBIO.GPIO as GPIO

GPIO.setup("P8_10", GPIO.OUT)
GPIO.output("P8_10", GPIO.HIGH)
GPIO.cleanup()
```

You can also refer to the pin names:

```
GPIO.setup("GPIO0_26", GPIO.OUT)
```

In the first example, you can see we used the "P8_10" key to designate which pin we'd like to set as the output, and the same pin in the second example, but using it's name "GPIO0_26".

You can also set pins as inputs as follows:

```
import Adafruit_BBIO.GPIO as GPIO

GPIO.setup("P8_14", GPIO.IN)
```

Once you've done that, you can access the input value in a few different ways. The first, and easiest way is just polling the inputs, such as in a loop that keeps checking them:

```
if GPIO.input("P8_14"):
    print("HIGH")
else:
    print("LOW")
```

You can also wait for an edge. This means that if the value is falling (going from 3V down to 0V), rising (going from 0V up to 3V), or both (that is it changes from 3V to 0V or vice-versa), the GPIO library will trigger, and continue execution of your program.

The wait_for_edge method is blocking, and will wait until something happens:

```
GPIO.wait_for_edge("P8_14", GPIO.RISING)
```

Another option, that is non-blocking is to add an event to detect. First, you setup your event to watch for, then you can do whatever else your program will do, and later on, you can check if that event was detected.

A simple example of this is as follows:

```
GPIO.add_event_detect("P9_12", GPIO.FALLING)
#your amazing code here
#detect wherever:
if GPIO.event_detected("P9_12"):
    print "event detected!"
```

We'll continue to add more examples, and features as we go, so check back often!

PWM

Below are a few examples of using the Adafruit_BBIO.PWM module. It's fairly simple to use as well!

Setup

To setup a pin to use PWM:

```
import Adafruit_BBIO.PWM as PWM
#PWM.start(channel, duty, freq=2000, polarity=0)
PWM.start("P9_14", 50)

#optionally, you can set the frequency as well as the polarity from their defaults:
PWM.start("P9_14", 50, 1000, 1)
```

The valid values for duty are 0.0 to 100.0. The start method activate pwm on that channel. There is no need to setup the channels with Adafruit_BBIO.PWM.

Once you've started, you can then set the duty cycle, or the frequency:

```
PWM.set_duty_cycle("P9_14", 25.5)
PWM.set_frequency("P9_14", 10)
```

You'll also want to either disable that specific channel, or cleanup all of them when you're done:

```
PWM.stop("P9_14")
PWM.cleanup()
```

ADC

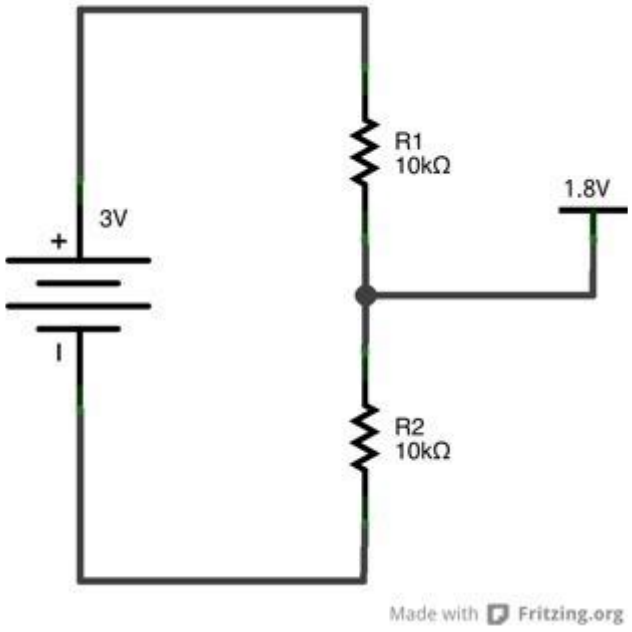
ADC currently has three methods available. setup, read and read_raw. You'll need to ensure you use setup prior to reading, otherwise an error will be thrown.

ADC is available on only a few pins, as listed below:

```
"AIN4", "P9_33"
"AIN6", "P9_35"
"AIN5", "P9_36"
"AIN2", "P9_37"
"AIN3", "P9_38"
"AIN0", "P9_39"
"AIN1", "P9_40"
```

1.8V is the maximum voltage. Do not exceed 1.8V on the AIN pins! VDD_ADC (P9_32) provides 1.8V. Use GNDA_ADC (P9_34) as the ground.

An easy way to drop the 3.3V of your device to the required 1.8V would be by using resistor divider. Two identical resistors (10K to 100K) in series from your 3v analog signal to ground, then connect the analog input pin between the two. The divider will divide the 0 to 3.3v into 0-1.65V which gives you a little bit of headroom as well.



Setup

To setup ADC, simply import the module, and call setup:

```
import Adafruit_BBIO.ADC as ADC

ADC.setup()
```

Then, to read the analog values on P9_40, simply read them:

```
value = ADC.read("P9_40")
```

In addition to the key (above), you can also read using the pin name:

```
value = ADC.read("AIN1")
```

There is currently a bug in the ADC driver. You'll need to read the values twice in order to get the latest value.

The values returned from read are in the range of 0 - 1.0. You can get the voltage by doing the following:

```
import Adafruit_BBIO.ADC as ADC

ADC.setup()
value = ADC.read("P9_40")
voltage = value * 1.8 #1.8V
```

You can also use read_raw to get the actual values:

```
import Adafruit_BBIO.ADC as ADC

ADC.setup()
value = ADC.read_raw("P9_40")
```

I2C

The Adafruit_I2C.py module is now included in the Adafruit_BBIO library as a top-level module. This means that many of the popular Python libraries built for the Raspberry Pi, will now just work on the BeagleBone Black if they are using I2C, such as the BMP085 sensor library (https://adafru.it/ciz).

To use the module, it's as simple as importing it, and setting the I2C address, and optionally the bus (the default is I2C-1):

```
from Adafruit_I2C import Adafruit_I2C

i2c = Adafruit_I2C(0x77)
```

I2C requires the python package 'python-smbus' installed from your distribution's package manager (opkg or apt-get) in order to function properly. It was included as part of the Adafruit_BBIO installation instructions. python-smbus is only compatible with Python2 thus far.

The I2C SCL and SDA pins enabled by default are as follows:

```
P9_19: I2C2, SCL
P9_20: I2C2, SDA
```

Probe the I2C busses for connected devices:

```
i2cdetect -y -r 0
i2cdetect -y -r 1
```

Latest pydoc of the I2C module:

```
class Adafruit_I2C
|  Methods defined here:
|
|  __init__(self, address, busnum=-1, debug=False)
|
|  errMsg(self)
|
|  readList(self, reg, length)
|      Read a list of bytes from the I2C device
|
|  readS16(self, reg)
|      Reads a signed 16-bit value from the I2C device
|
|  readS16Rev(self, reg)
|      Reads a signed 16-bit value from the I2C device with rev byte order
|
|  readS8(self, reg)
|      Reads a signed byte from the I2C device
|
|  readU16(self, reg)
|      Reads an unsigned 16-bit value from the I2C device
|
|  readU16Rev(self, reg)
|      Reads an unsigned 16-bit value from the I2C device with rev byte order
|
|  readU8(self, reg)
|      Read an unsigned byte from the I2C device
|
|  reverseByteOrder(self, data)
|      Reverses the byte order of an int (16-bit) or long (32-bit) value
|
|  write16(self, reg, value)
|      Writes a 16-bit value to the specified register/address pair
|
|  write8(self, reg, value)
|      Writes an 8-bit value to the specified register/address
```

```
|
|  writeList(self, reg, list)
|      Writes an array of bytes using I2C format
```

SPI

SPI is included with the Adafruit_BBIO library. The following are the basics on how to use it.

You can import the SPI module:

```
from Adafruit_BBIO.SPI import SPI
```

Once you've imported it, you'll want to initialize the bus and device:

```
spi = SPI(0,0)
```

The BeagleBone Black (BBB) includes SPI0, as well as SPI1. SPI1 is currently not available by default as the HDMI interface is utilizing one of the pins.

Note: It is not possible to use SPI1 on the BeagleBone Black without disabling the HDMI interface.

There are four /dev/spidev* bus and device combinations available. They are available by executing the following code:

```
#import the library
from Adafruit_BBIO.SPI import SPI

#Only need to execute one of the following lines:
#spi = SPI(bus, device) #/dev/spidev&lt;bus&gt;.&lt;device&gt;
spi = SPI(0,0)     #/dev/spidev1.0
spi = SPI(0,1)     #/dev/spidev1.1
spi = SPI(1,0)     #/dev/spidev2.0
spi = SPI(1,1)     #/dev/spidev2.1
```

If you'd like to disable HDMI to access SPI1, you can add the following to your uEnv.txt file in the small FAT partition on your BBB:

```
mkdir /mnt/boot
mount /dev/mmcblk0p1 /mnt/boot
nano /mnt/boot/uEnv.txt
#change contents of uEnv.txt to the following:
optargs=quiet capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

Pins used for SPI0 and SPI1

PORT	CS0	DO	DI	SCLK
SPI0	P9_17	P9_21	P9_18	P9_22
SPI1	P9_28	P9_29	P9_30	P9_31

UART

The Adafruit IO Python library will export the UART device tree overlays as a convenience. There are five serial ports brought to the expansion headers (UART3 only has a single direction, TX), and one (UART0) with dedicated headers that aren't available to use in your Python programs.

Setup

To setup and export the UART, you can do the following:

```
import Adafruit_BBIO.UART as UART

UART.setup("UART1")
```

That's it!

Also, there is a cleanup() method ready to go, but it's not currently working due to a bug in the kernel that causes kernel panics when unloading device tree overlays. We'll update this when it's working. A workaround is to either leave the UART enabled, or restart your BeagleBone Black.

Pin Table for UART

UART	RX	TX	CTS	RTS	Device
UART1	P9_26	P9_24	P9_20	P9_19	/dev/ttyO1
UART2	P9_22	P9_21			/dev/ttyO2
UART3		P9_42	P8_36	P8_34	/dev/ttyO3
UART4	P9_11	P9_13	P8_35	P8_33	/dev/ttyO4

UART5	P8_38	P8_37	P8_31	P8_32	/dev/ttyO5

Using UART with Python

You can use the pyserial module in Python, but you'll first need to install it using pip. If you don't have pip installed, you can follow the instructions on the installation pages for this tutorial.

SSH into the BeagleBone Black, and execute the following command:

```
pip install pyserial
```

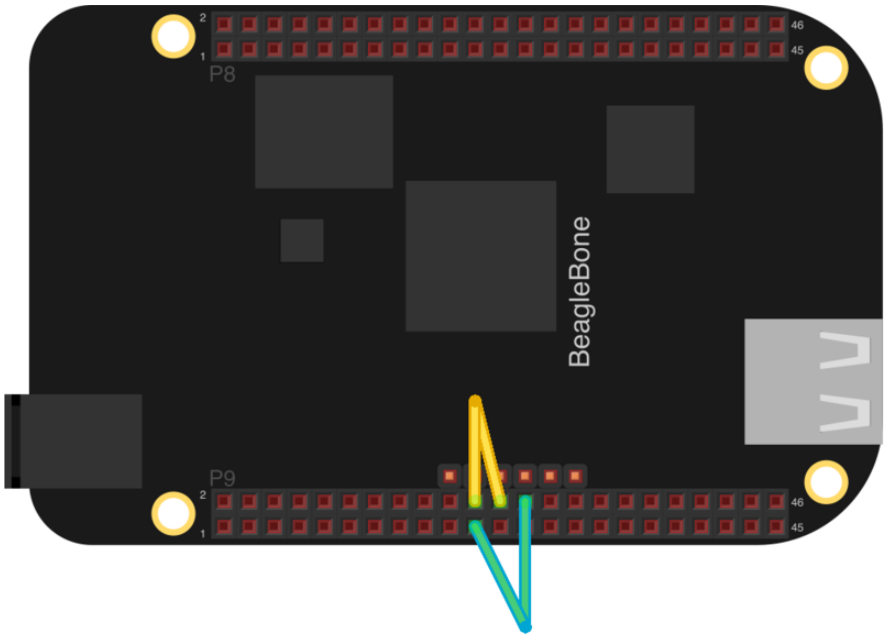
Below is a very simple python program that is a good starting point. Save it to a file, and execute it with 'python file_name.py'

```
import Adafruit_BBIO.UART as UART
import serial

UART.setup("UART1")

ser = serial.Serial(port = "/dev/ttyO1", baudrate=9600)
ser.close()
ser.open()
if ser.isOpen():
        print "Serial is open!"
    ser.write("Hello World!")
ser.close()

# Eventually, you'll want to clean up, but leave this commented for now,
# as it doesn't work yet
#UART.cleanup()
```

Testing and Using the UART

You can easily test that everything is working, without having to code anything, or installing any other dependencies to get started.

Next, you'll want to connect two wires to the UART pins. We're just going to cross the RX/TX of the UART1 and UART2. The first wire should connect from P9_24 to P9_22. The second wire should connect from P9_26 to P9_21.

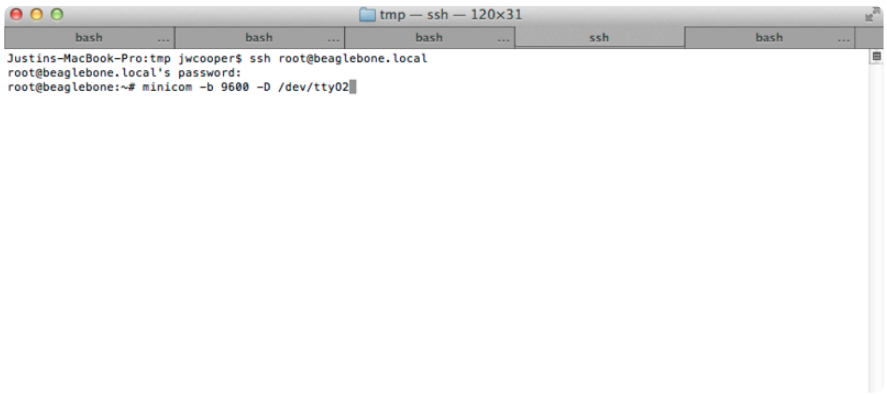


Next, export the UART1 and UART2 in the python interpreter with the Adafruit IO library:

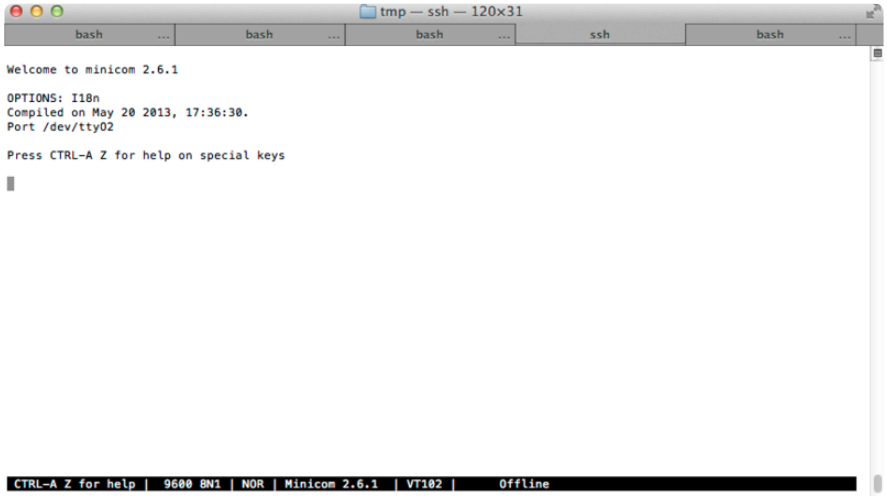
```
root@beaglebone:~# python
Python 2.7.3 (default, May 29 2013, 21:25:00)
[GCC 4.7.3 20130205 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
&gt;&gt;&gt; import Adafruit_BBIO.UART as UART
&gt;&gt;&gt; UART.setup("UART1")
&gt;&gt;&gt; UART.setup("UART2")
&gt;&gt;&gt; exit()
```

Once you've done that, execute the following commands to launch minicom (using two separate ssh sessions using your terminal of choice, mine are separated by tabs in OS X):

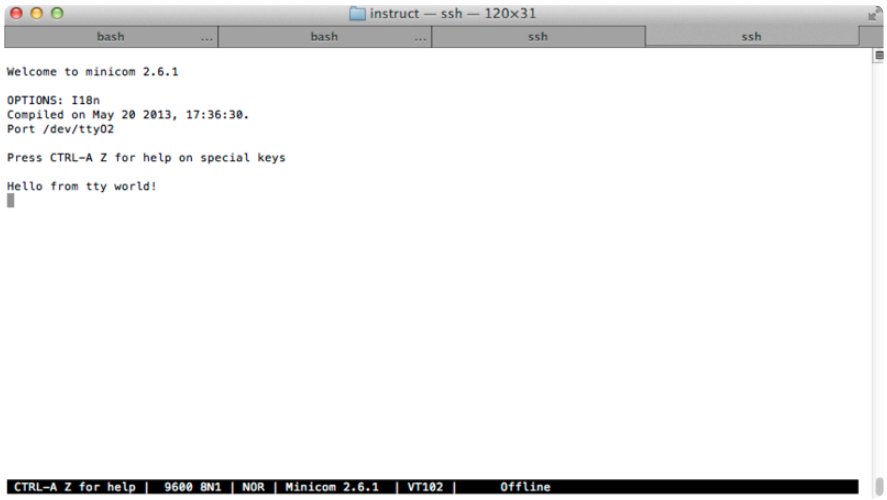
```
#first terminal window:
minicom -b 9600 -D /dev/ttyO1

#second terminal window:
minicom -b 9600 -D /dev/ttyO2
```



It should look like this once you've opened minicom:



Now, type in one of the terminal windows, and hit enter. Look at the other terminal, and you should see something like this:



You can exit from minicom by typing Ctrl-A, then Z, then X, and hit enter at the dialog.

Pin Details

More Details coming soon! The below images are from the BeagleBone Black System Reference Manual.

The below table is useful to find the GND and VDD pins:

Table 19. Expansion Voltages

Current	Name	P9			Name	Current
	GND	1		2	GND	
250mA	VDD_3V3B	3		4	VDD_3V3B	250mA
1000mA	VDD_5V	5		6	VDD_5V	1000mA
250mA	SYS_5V	7		8	SYS_5V	250mA
		:		:		
	GND	43		44	GND	
	GND	45		46	GND	

Avoid the eMMC and HDMI (LCD) pins altogether, unless you don't mind disabling those features.

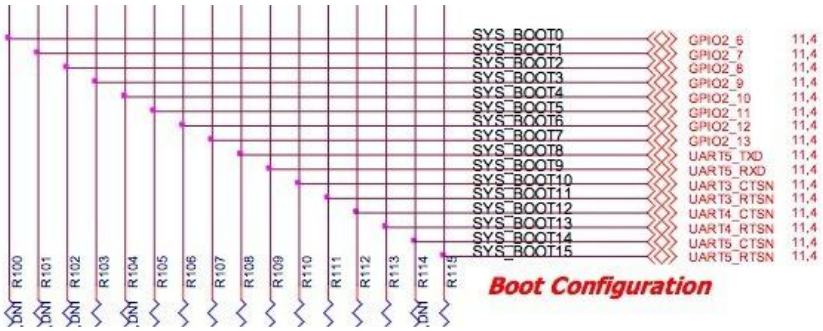
Table 14. P8 eMMC Conflict Pins

PIN	PROC	NAME	MODE2
22	V8	MMC1_DAT5	mmc1_dat5
23	U8	MMC1_DAT4	mmc1_dat4
24	V7	MMC1_DAT1	mmc1_dat1
5	R8	MMC1_DAT2	mmc1_dat2
4	T9	MMC1_DAT7	mmc1_dat7
3	R9	MMC1_DAT6	mmc1_dat6
6	T8	MMC1_DAT3	mmc1_dat3
25	U7	MMC1_DAT0	mmc1_dat0
20	V9	MMC1_CMD	mmc1_cmd
21	U9	MMC1_CLK	mmc1_clk

Table 13. P8 LCD Conflict Pins

PIN	PROC	NAME	MODE0
27	U5	GPIO2_22	lcd_vsync
28	V5	GPIO2_24	lcd_pclk
29	R5	GPIO2_23	lcd_hsync
30	R6	GPIO2_25	lcd_ac_bias_en
31	V4	UART5_CTSN	lcd_data14
32	T5	UART5_RTSN	lcd_data15
33	V3	UART4_RTSN	lcd_data13
34	U4	UART3_RTSN	lcd_data11
35	V2	UART4_CTSN	lcd_data12
36	U3	UART3_CTSN	lcd_data10
37	U1	UART5_TXD	lcd_data8
38	U2	UART5_RXD	lcd_data9
39	T3	GPIO2_12	lcd_data6
40	T4	GPIO2_13	lcd_data7
41	T1	GPIO2_10	lcd_data4
42	T2	GPIO2_11	lcd_data5
43	R3	GPIO2_8	lcd_data2
44	R4	GPIO2_9	lcd_data3
45	R1	GPIO2_6	lcd_data0
46	R2	GPIO2_7	lcd_data1

The Boot Configuration pins are to be avoided during BeagleBone bootup.



FAQ

I get errors installing with opkg or doing ntpupdate to set the time. How can that be resolved?

You'll need internet connectivity to your BeagleBone Black in order to install the IO Python Library. Ensure that you can access the internet. You can try: ping adafruit.com as an example, or if you're using the HDMI out, and have a desktop, open a browser and test out the internet.

First, ensure you have ethernet connected to a switch or router. Or, if you've setup wifi using one of our guides, ensure that it's working properly.

Next, you can try to fix this by adding a DNS nameserver. Simply do that with the following command:

```
echo nameserver 8.8.8.8 &gt; /etc/resolv.conf
```

Where is the source for the Adafruit BBIO python core?

Check it! -> http://github.com/adafruit/adafruit-beaglebone-io-python

Does the Adafruit BBIO library support Python 3?

It's close. Everything is in place, but we mostly need people to test it, and submit any bugs (with fixes would be nice!) to the GitHub repository.

I have some ideas to improve the library, how should I contribute?

The source code is located on GitHub here (https://adafru.it/elY). Please submit pull requests with your code, along with tests supporting it.

If you have any bugs, please submit them there as well.

If you have any questions on how to use the library, please ask them on the Adafruit Forums (https://adafru.it/elZ) as they will be answered quickly there.