# Device Setup (software and OS)

## Change the password for root (Angstrom and Debian):

Plug the board into your laptop.

SSH into the board. Default USB-to-Ethernet IP address is 192.168.7.2

Type passwd, give root a good new password (been using “harmony”)  
  
The following directory should be where logger files should be located:

**/srv/field-research/**

You can create a symbolic link to this from the root directory by typing:

~/#: ln –s /srv/field-research/field-code field-code

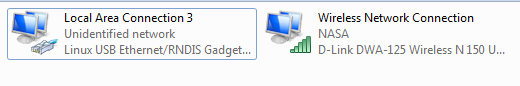
~/#: ln –s /srv/field-research/data data

## If just USB-linked to Laptop, allow internet connection sharing over laptop:

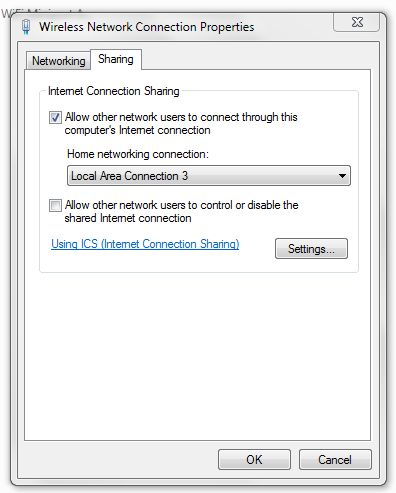
OpenControl Panel -> Network and Internet -> Network and Sharing Center

**2.**One the left side panel, click Change Adapter Settings

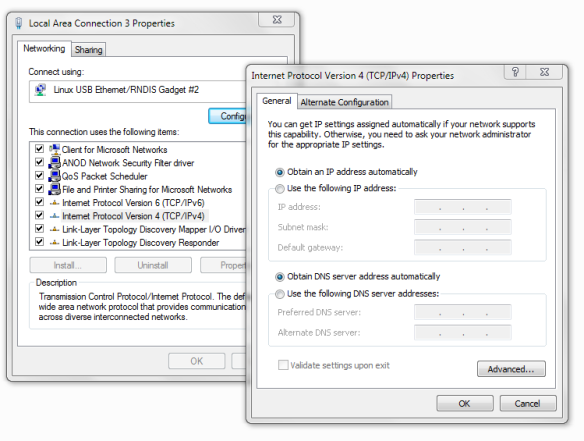
**3.**Now, this should show all the network adapter available on the computer. Here’s mine, the one on the left is the BeagleBone Black, you can tell by the description Linux USB Ethernet. And the one on the right is my Internet source, a wireless adapter.

[](https://billwaa.files.wordpress.com/2014/08/network.png)

**4.**Here, we are going to enable Internet Sharing on Windows’ side by right clicking on your Source Internet Adapter. In my case, Wireless Network Network Connection, thenProperties. Click Sharing, check Allow other network user to connect…and then select the BeagleBone network adapter from the drop down menu.

[](https://billwaa.files.wordpress.com/2014/08/internet-sharing.png)

**5.** So after doing this, Windows will reconfigure the IP address for the BeagleBone, we need to set this back to automatic. Right Click on BeagleBone Network -> Properties -> highlight Internet Protocol Version 4 (TCP/IPv4)->  click Properties ->mark bothObtain IP and DNS server adress automatically.

[](https://billwaa.files.wordpress.com/2014/08/autoip.png)

**6.**We are all set on the Windows side, now we need to configure the BBB the route the Internet through your main computer. We do this by connect to[BBB through SSH like before](http://billwaa.wordpress.com/2014/08/03/beaglebone-black-expand-file-system-capacity-on-the-microsd/). Since we need to do this step every time we start up again, we might as well write a script using the nano editor. We will call this file EnableUSBInternet.

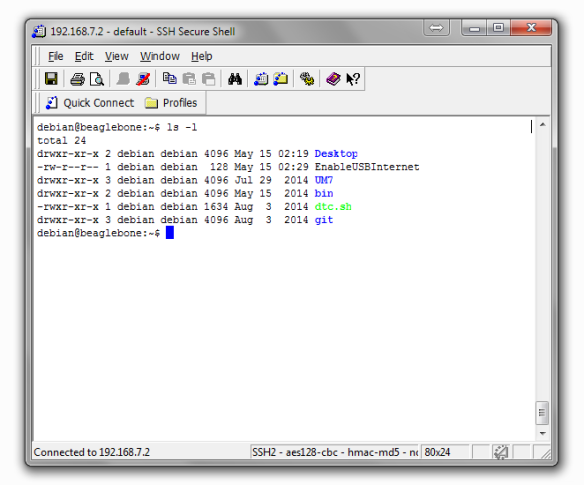
|  |  |
| --- | --- |
| 1 | nano EnableUSBInternet |

Write the following into the script… basically, it route the internet to the pc, who’s address is 192.168.7.1, while the BeagleBone’s IP is 192.168.7.2

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | echo "Routing Default Gateway to 192.168.7.1"  sudo /sbin/route add default gw 192.168.7.1  echo "Updating the nameserver entry"  echo "nameserver 8.8.8.8" >> /etc/resolv.conf  echo "USB Internet Sharing Enabled!" |

**Ctrl + X** to exit, press **Y**to save and press **Enter**to confirm to file name.

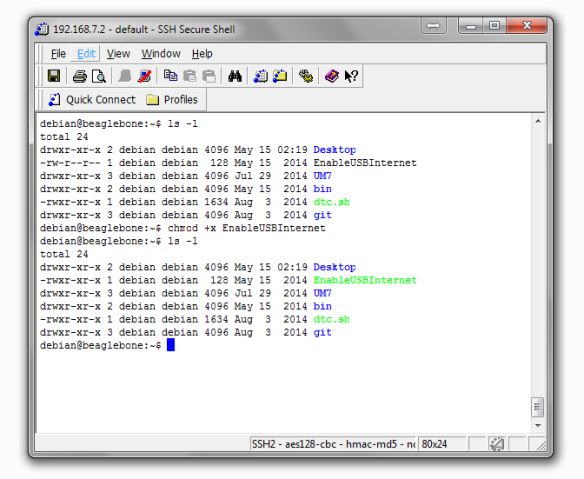
**7.**Use the **clear** command to clear the terminal for better view. Type in **ls -l** to list the files available in the current directory.

[](https://billwaa.files.wordpress.com/2014/08/ls-l.png)

This pretty much show everything in the current directory. What we concern ourselves here is the EnableUSBInternet file, the left side -rw-r–r– list the permission. Right now, we can write, read, but we can’t execute the file. Therefore, we need to use the following command to change the permission to allow execution.

|  |  |
| --- | --- |
| 1 | chmod +x EnableUSBInternet |

We can see now that the file is executable with the different color and the “x” permission indicator.

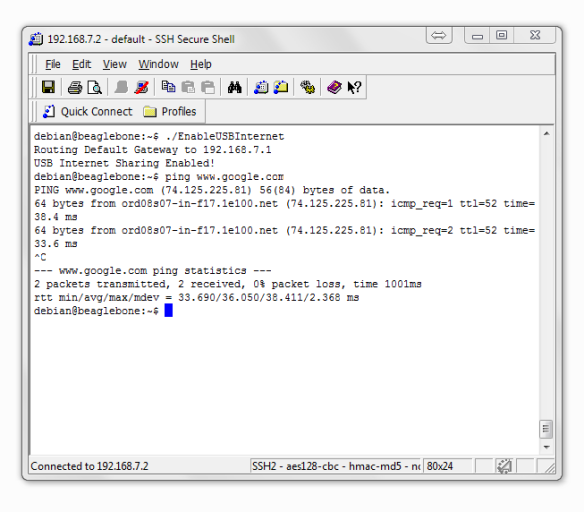
[](https://billwaa.files.wordpress.com/2014/08/permission.png)

**8.**Finally, we just need to run this script to set up Internet. Remember to run this script every time during start up to get Internet.

|  |  |
| --- | --- |
| 1 | ./EnableUSBInternet |

We can confirm that we actually got Internet access by pinging google, do **Ctrl+C** to exit pinging process.

|  |  |
| --- | --- |
| 1 | ping www.google.com |

[](https://billwaa.files.wordpress.com/2014/08/ping-google.png)

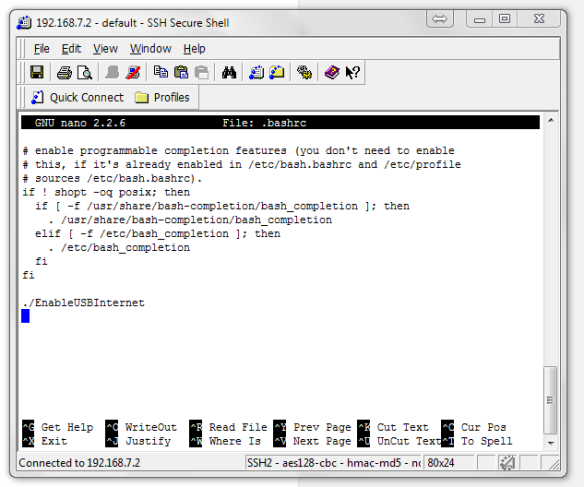
**9.**Okay, since it’s annoying having to hand launch this script every time we start up, why not make it automatic? Therefore, we going to make edit to the .bashrc file and have it launch the script whenever we log it.

|  |  |
| --- | --- |
| 1 | sudo nano .bashrc |

Scroll all the way down and add the following line:

|  |  |
| --- | --- |
| 1 | ./EnableUSBInternet |

**Ctrl+X**-> **Y**-> **Enter** to save the file. Now, just reboot and it will launch when we log in!

[](https://billwaa.files.wordpress.com/2014/08/autostart.png)

Resources:

<http://billwaa.wordpress.com/2014/08/03/beaglebone-black-enable-usb-internet-sharing-from-windows/>

<http://derekmolloy.ie/beaglebone/getting-started-usb-network-adapter-on-the-beaglebone/>

## Update kernel to latest( for Debian):

Ensure there’s an outbound internet connection, and SSH into the BBB, then do:

cd /opt/scripts/tools/

git pull

sudo ./update\_kernel.sh

sudo reboot

## For Auto-sync of the date and time (no timezone) (for Angstrom)

Modify the cron command by SSH’ing in and typing “crontab –e”

then edit the first line that normally reads:

30 \* \* \* \* /usr/bin/ntpdate-sync silent

To be:

30 \* \* \* \* /usr/bin/ntpdate –b –s –u pool.ntp.org

For **Debian,** it would be:

30 \* \* \* \* /usr/sbin/ntpdate –b –s –u pool.ntp.org

**Changing the Hostname**

Edit /etc/hostname to be (e.g.)

beagleboneCM02

and /etc/hosts to be(e.g.):

127.0.0.1 localhost

127.0.0.1 beagleboneCM02

## To make a Python process execute on every startup:

Check out Steps in the section Circuitco RTC Cape, Revision A1   
Another resource: <http://stackoverflow.com/questions/11152657/angstrom-start-up-processes-beaglebone> as reproduced here:  
Create a new file in /lib/systemd/system/ (rfidreader.service in my example) with a content like:

[Unit]

Description=Start Python RFID reader

[Service]

WorkingDirectory=/...Python script path.../

ExecStart=/usr/bin/python rfidreader.py

KillMode=process

[Install]

WantedBy=multi-user.target

Then execute the following command to install the service:

systemctl enable rfidreader.service

To start the service, you can reboot or execute

systemctl start rfidreader.service

To check if the service is running and get the latest outputs from the script:

systemctl status rfidreader.service

The systems that are deployed have the following startup script “LoggerMain.service”, which should be located in /lib/systemd/system/

## To mount an SDCard as extra storage (Angstrom):

If you want to store data out to the SDcard for extra storage space and portability, format an SDcard (must be fat32) and put the file uEnv.txt onto it:

Insert a USB-to-microSD converter with SDcard into the USB port of the Beaglebone black

Create a temporary directory to mount the drive to (such as ~/usbDrive/)

Type “mount /dev/sda1 ~/usbDrive/” to mount the drive.

Type fdisk /dev/sda1 to drop into format disk

Type p to list partitions

If there are any, delete them, then type ‘n’ to create a new partition. Select as primary partition, maximize the blocks, type ‘t’ and then ‘c’ to change to type fat32, then enter ‘w’ to write. Type ‘reboot’ to renew the partition. This instruction is adapted from [here](http://elinux.org/Beagleboard:Expanding_File_System_Partition_On_A_microSD).

Open uEnv.txt and fill it with the following 4 lines of code:

mmcdev=1

bootpart=1:2

mmcroot=/dev/mmcblk1p2 ro

optargs=quiet

Save the file and shutdown your BeagleBone Black  
Remove the microSD from the USB hub and insert it into the microSD slot on the BeagleBone Black  
Apply power to your BeagleBone Black and when its starts you should have the microSD mounted as available storage (at /mnt/card )

More detail at:

<http://elinux.org/Beagleboard:MicroSD_As_Extra_Storage>

## To mount an SDCard as extra storage(Debian):

Create a partition on the drive in FAT32 format just as with Angstrom

Type df to see the filesystems:

Filesystem 1K-blocks Used Available Use% Mounted on

rootfs 3553816 1462516 1907444 44% /

udev 10240 0 10240 0% /dev

tmpfs 101692 620 101072 1% /run

tmpfs 254224 0 254224 0% /dev/shm

tmpfs 254224 0 254224 0% /sys/fs/cgroup

tmpfs 5120 0 5120 0% /run/lock

tmpfs 102400 0 102400 0% /run/user

/dev/mmcblk1p1 98094 90384 7710 93% /boot/uboot

/dev/mmcblk0p1 7557120 160 7556960 1% /media/29DD-233B

Here you’ll see that the uSD card has been mounted to the directory /media/29DD-233B. The specific directory may be different for your specific uSD card.

Just create a symbolic link in the root home directory to link to this auto-mount location:

Cd ~

Ln –s /media/29DD-233B/ uSDcard

## To run everything off a micro SD Card (Debian):

To run everything off of a micro SD Card, first download the latest Debian release and send the image file to a micro SD card:

<http://beagleboard.org/latest-images>

<http://elinux.org/BeagleBoard_Community#Dual_partition_card>

The Debian image should, if the SD card is inserted during boot-up, first try to boot off of the micro SD card.

Insert the micro SDcard into a powered Beaglebone black (so that the drive mounts while powered on and running the OS off the eMMC)

## Expand the File System Partition On A micro SD

By default the microSD image is only about 2GB in size; to fully expand the partition to the full size of your microSD card, you have to remove the swap partition (if it exists), then delete the main linux partition, then add a new partition starting from where the main Linux partition was and specifying its end point as being the end of the microSD storage. The two tools to do this are “fdisk” and “resize2fs”. Use the following instructions that show how to do this:

One of the main reasons to boot from the SD card rather than the internal eMMC image is that we need more room to work and right now the prebuilt Debian image is sized to fit neatly into the internal eMMC space. However the image can be resized to take full advantage of additional space available on our SD card.

**fdisk /dev/mmcblk0**

Command (m for help): **p**

Disk /dev/mmcblk0: 3904 MB, 3904897024 bytes

4 heads, 16 sectors/track, 119168 cylinders, total 7626752 sectors

Units = sectors of 1 \* 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk identifier: 0x80000000

Device Boot Start End Blocks Id System

/dev/mmcblk0p1 \* 2048 4095 1024 1 FAT12

/dev/mmcblk0p2 4096 3751935 1873920 83 Linux

Command (m for help): **d**

Partition number (1-4): **2**

Command (m for help): **n**

Partition type:

p primary (1 primary, 0 extended, 3 free)

e extended

Select (default p): **p**

Partition number (1-4, default 2): **2**

First sector (4096-7626751, default 4096):

Using default value 4096

Last sector, +sectors or +size{K,M,G} (4096-7626751, default 7626751):

Using default value 7626751

Command (m for help): **p**

Disk /dev/mmcblk0: 3904 MB, 3904897024 bytes

4 heads, 16 sectors/track, 119168 cylinders, total 7626752 sectors

Units = sectors of 1 \* 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk identifier: 0x80000000

Device Boot Start End Blocks Id System

/dev/mmcblk0p1 \* 2048 4095 1024 1 FAT12

/dev/mmcblk0p2 4096 7626751 3811328 83 Linux

Command (m for help): **w**

The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16: Device or resource busy.

The kernel still uses the old table. The new table will be used at

the next reboot or after you run partprobe(8) or kpartx(8)

Syncing disks.

and now we've altered the partition table go ahead and reboot.

Once the board has rebooted, ssh back into and go ahead and resize the file system to fill the now larger partition,

**resize2fs /dev/mmcblk0p2**

resize2fs 1.42.5 (29-Jul-2012)

Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing required

old\_desc\_blocks = 1, new\_desc\_blocks = 1

The filesystem on /dev/mmcblk0p2 is now 952832 blocks long.

all of the space on the SD card is now available to use, on a 4GB card for instance we should have roughly 3.1GB free,

**df –h**

Filesystem Size Used Avail Use% Mounted on

rootfs 3.6G 356M 3.1G 11% /

/dev/root 3.6G 356M 3.1G 11% /

devtmpfs 248M 0 248M 0% /dev

tmpfs 50M 228K 50M 1% /run

tmpfs 5.0M 0 5.0M 0% /run/lock

tmpfs 100M 0 100M 0% /run/shm

/dev/mmcblk0p1 1004K 478K 526K 48% /boot/uboot

NOTE: The booted device is always device 0, i.e. mmcblk0. The internal eMMC always has the mmcblkXboot0 and mmcblkXboot1 entries. If the device is booted from eMMC the entries will be mmcblk0boot0 and mmcblk0boot1. If the device is NOT booted from eMMC the entries will be mmcblk1boot0 and mmcblk1boot1.

Resources:

<https://github.com/TheThingSystem/steward/wiki/Bootstrapping-the-BeagleBone-Black-with-Debian>

<http://raspberrypi.stackexchange.com/questions/499/how-can-i-resize-my-root-partition>

## Backing up an SD Card or capturing a preferred image

The following site provides a number of great ways to backup the SDcard on the BBB, resulting in an image file that can be duplicated in another BBB:

<http://www.gigamegablog.com/2012/09/26/beaglebone-101-linux-tricks-for-backing-up-and-resizing-your-microsd-card/>

My (Ben A’s) process, captured here:

Because I set up an 8Gb uSD card, the process for doing a backup on the BBB directly was erroring out (not enough space). I’m not sure if this was in reference to a temp file needed to do the gzip because there was plenty of space to store the gzip.

So instead, I installed Virtual Box by Oracle on the Windows 7 laptop and did a “Net install CD” install of Debian. I added a shared drive between Windows 7 and Debian VM. I tried Windows’ Virtual PC, but this was running into issues with the Debian install .iso. After having Debian up and running, I removed the BBB uSD card and inserted it as a USB drive for the laptop.

Linking the laptop’s USB resource to Debian’s Virtual Machine, I executed the following command:

root@debian:~# fdisk –l  
(This first to find the correct USB resource)

root@debian:~# dd if=/dev/sdb | gzip -1 > /media/sf\_Ctemp/sd\_backup.img.gz

This command took a full image (both partitions, FAT and Linux) of the USB drive “sdb” and zipped and placed it on the Window 7 shared drive spot. I could then go ahead and use Win32 disk imager to duplicate this on other 8GB SD cards.

## For adding the correct python libraries for interfacing GPIO (for angstrom):

SSH into the board.

Run the following (each individually, as if you do it in one line, it seems to error out):

opkg update

opkg install python-pip

opkg install python-setuptools

opkg install python-smbus

pip install Adafruit\_BBIO

Verify the installation:

python -c "import Adafruit\_BBIO.GPIO as GPIO; print GPIO"

A typical output would then be:

<module 'Adafruit\_BBIO.GPIO' from '/usr/local/lib/python2.7/dist-packages/Adafruit\_BBIO/GPIO.so'>

## For adding the correct python libraries for interfacing GPIO (for Debian):

SSH into the board.

Run the following:

sudo apt-get update

sudo apt-get install python-pip

sudo apt-get install python-setuptools

sudo apt-get install python-smbus

pip install Adafruit\_BBIO

## For setting up FTP Server (dropbear may already work with port 22):

Next, install a vsftpd for FTP server, following instructions here:

Type in "**sudo apt-get install vsftpd**" and hit Return / Enter  
Now type in "**sudo nano /etc/vsftpd.conf**" and hit Return / Enter  
  
Search through the file and change the following lines:  
**anonymous\_enable=YES** Change To**anonymous\_enable=NO**  
**#local\_enable=YES** Change To**local\_enable=YES  
#write\_enable=YES**Change To**write\_enable=YES**  
Also, add a line to the bottom of the file:  
**force\_dot\_files=YES**  
Thenhold the **Ctrl** key and press "**x**", then "**y**", then the Return / Enter key.  
  
Now restart the FTP server with "**sudo service vsftpd restart**"

## For setting up rsync (Debian Linux):

First, find out the UniqueID for the beaglebone black, see Getting a Unique ID for each Beaglebone Black within this document. In the example below, the unique ID is 3514BBBK0822.

Follow this instruction for setting up an rsync server (though this should be done at the IT level for permanent support): <http://everythinglinux.org/rsync/>

**App6.ecw.org** should be the server we use

**Secure.ecw.org** has this capability, but external SSH is disabled.

Follow this instruction for generating a trusted key for password-free rsync (over ssh):

<http://www.thegeekstuff.com/2011/07/rsync-over-ssh-without-password/> reproduced here:

### Use ssh-keygen to generate keys.

Use ssh-keygen on local server to generate public and private keys.

$ ssh-keygen  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:

**Note:** When it asks you to enter the passphrase just press enter key, and do not give any password here.  
**Use ssh-copy-id copies public key to remote host**

Use ssh-copy-id, to copy the public key to the remote host.

ssh-copy-id -i ~/.ssh/id\_rsa.pub frsa@app6.ecw.org

**Note:** The above will ask the password for your account on the remote host, and copy the public key automatically to the appropriate location. If ssh-copy-id doesn’t work for you, use the method we discussed earlier to [setup ssh password less login](http://www.thegeekstuff.com/2008/06/perform-ssh-and-scp-without-entering-password-on-openssh/).

### Perform rsync over ssh without password

rsync -avz -e ssh /root/uSDcard/data/ frsa@app6.ecw.org:/home/frsa/Data/2445\_CS/3514BBBK0822

Now, you should be able to ssh to app6.ecw.org without entering the password. Note: if you exclude the last slash in the first directory (/root/uSDcard/data/), the rsync command will create the subdirectory “data” beneath the destination folder.

If you want to schedule this rsync backup job automatically, use [cron](http://www.thegeekstuff.com/2009/06/15-practical-crontab-examples/) to set it up.

IF THIS IS an issue, this website showed some typical problems: <http://www.daveperrett.com/articles/2010/09/14/ssh-authentication-refused/> Also, use the ssh –v option for verbose debug.

### Cron Job Example

Enter “crontab –e” to open an editor of your username’s cron tab

An example crontab line is thus (two examples below):

0 \* \* \* \* rsync -avz -e ssh /root/uSDcard/data/ frsa@app6.ecw.org:/home/frsa/Data/2445\_CS/3514BBBK0822  
0 \* \* \* \* rsync -avz -e ssh /home/python\_queries/ frsa@secure.ecw.org:/home/frsa/learle/test/rPiTestMPLS

This runs every top of the hour.

This file is kept in /var/spool/cron/crontabs/

## Set up Reverse SSH tunneling to the server

Follow the steps outlined at <http://www.thegeekstuff.com/2013/11/reverse-ssh-tunnel/> as reproduced here:   
Reverse SSH is a technique that can be used to access systems (that are behind a firewall) from the outside world.

As you already know SSH is a network protocol that supports cryptographic communication between network nodes. Using this protocol, you can do a secure remote login, secure copy from/to a remote machine etc.  
  
You’ll typically do the following to connect to a remote server securely using [ssh command](http://www.thegeekstuff.com/2008/05/5-basic-linux-ssh-client-commands/).

$ ssh [your-account-login]@[server-ip]

What is Reverse SSH?

SSH is very good tool to access remote machine or server securely. But, the problem arises when you try to connect to a remote server which is behind a firewall and this firewall denies any incoming connection or data transfer request that has no prior outgoing request. This means that only those connections would be allowed which are initiated by the remote server machine. This is a real problem for those who want to access this server machine remotely.

Reverse SSH provides a technique through which you can simulate a normal SSH to this remote server machine.

The main problem is that firewall is rejecting ssh connection that your machine is trying to establish with remote server machine. But you know that the same firewall will not have any problem with the connections originating from server machine. So, why not ask some one who is sitting behind the firewall to do something with which you can achieve your goal of remotely accessing the server. To do this we have to use ssh -R option.

This is the description of the ssh -R option from the man page:

-R [bind\_address:]port:host:hostport

Specifies that the given port on the remote (server) host is to be forwarded to the given host and port on the local side. This works by allocating a socket to listen to port on the remote side, and whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to host port hostport from the local machine.

So you can use ssh command, with -R option, (from server in our case) to connect to your machine, allocate a port there and make sure that any connection request on that port is forwarded to ssh port of the remote server.

Instead of your machine doing an ssh, the server machine does an ssh and through port forwarding makes sure that you can ssh back to server machine.

How to Create a Reverse SSH Tunnel?

Here is the command your friend sitting on remote server side should run on the server :

ssh -fN -R 7000:localhost:22 username@yourMachine-ipaddress

So this ssh connection request originating from remote server to your machine will make sure that any ssh connection request for port 7000 on your machine is forwarded to port 22 of remote server.

Now do an ssh connection request from your machine to your own machine at port 7000:

ssh username@localhost -p 7000

Here, though it may seem like you are doing ssh on localhost but your request would be forwarded to remote host. So, you should use your account ‘username’ on remote server and when prompted for password, enter the corresponding password.

This should clear most of the aspects related to reverse ssh technique. But, there is one catch. The catch is that you have to ask a friend of yours — who is sitting behind the firewall — to create an ssh connection first. This is not feasible every time.

To overcome this problem, what you can do is that you can set up a machine which is out of firewall (just like yours) in a way that it is always on. Lets call this machine as machine\_z.

The benefit of machine\_z is that you can once do this reverse ssh set up on it and leave it like this. At any point of time, when you need to login to remote machine, you can ssh into machine\_z on a specified port (as shown earlier) and your connection request will be forwarded to remote server machine and Voila, you will be running commands on remote server.

On a related note, you can also setup [no password SSH](http://www.thegeekstuff.com/2008/11/3-steps-to-perform-ssh-login-without-password-using-ssh-keygen-ssh-copy-id/) to connect to another server without entering the password.

Some settings that you need to configure on machine\_z would include :

* Make sure that the parameters TCPKeepAlive, ClientAliveInterval, ClientAliveCountMax and GatewayPorts are set to appropriate values. These parameters are found in /etc/sshd\_config or [/etc/ssh/sshd\_config](http://www.thegeekstuff.com/2011/05/openssh-options/) file
* If you make some changes to above parameters, you should restart sshd daemon to reflect the changes.
* Also, make sure that you run the first ssh command (that is executed from the remote server to machine\_z) using [nohup command](http://linux.101hacks.com/unix/nohup-command/) so that this ssh session becomes immune to hangups which can happen when a user logs out.

## Upload python project files:

Use filezilla, login with root over port 22 to connect to the BBB

Upload all of the Files to the BBB – making sure field-code and data directories are separate.

Edit LogConfig.py to match the settings of the specific site.

Test the python script in the shell to verify correct operation.   
Press Ctrl+C to cancel the execution after it’s verified as able to run.

## Run python program for extended period:

First ensure the power supply has battery backup and the date has been set for the BeagleBoneBlack.

Use “no hangup” to execute the python program:  
**nohup python LoggerMain.py &**

Then delete the nohup.out (to avoid memory buildup):  
**rm nohup.out**

**Verify python program is running detached:**

**ps –a | grep python**

output should look like this:  
1721 pts/0 00:01:28 python

This shows a process is running, and it is indeed detached.

Exit the ssh session, and this program should continue to operate unattached to the ssh session. (can double verify by making sure valves are still switching).

## Using the Beaglebone Black Watchdog timer

The watchdog can be used by simply opening and writing to /dev/watchdog at least once every minute. If it doesn’t get written to within that time period, the board will reset itself.

To use it:

1. Open the file /dev/watchdog
2. Do not close the file
3. Write something (e.g. "\n") to the file at least every 59 seconds to keep the system running
4. In python, don’t forget to flush the file buffer by also executing <fileobject>.flush()
5. Highlevel testing shows it works if you violate the watchdog time within a program, but it ceases to count down if that process ends.   
   There’s also some work shown that the Debian package requires apt-get install watchdog and then adding /dev/watchdog as the watchdog-device in /etc/watchdog.conf
6. Testing this, it still seems that if python code is exited, the watchdog will not reset the system.
7. See the second resource – it may require a recompile or reconfigure of the kernel

Resource: <http://beaglebone.cameon.net/home/watchdog-timer>

<https://groups.google.com/forum/#!topic/beagleboard/wN_hxa2RtxA>

## Getting a Unique ID for each Beaglebone Black

The following Shell script will retrieve the EEPROM serial number, which is unique for each shipped Beaglebone black (place in a shell script and chmod +x it).

The script is captured in the field\_code directory and originally copied from here:

<https://github.com/gkaindl/beaglebone-ubuntu-scripts/blob/master/bb-show-serial.sh>

# Accessing the HarmonyGateway Remotely (some of this might be good for BBB)

There is a wiki page on how to set this up using a dynamic IP dns server like [no-ip.com](http://www.noip.com/):

<http://wiki.simplehomenet.com/index.php?title=Remote_Access> port forwarding, etc.

The ports that should be forwarded to the static IP for this gateway are:

50333 – TCP/IP based API

843 – secure web UI Client

22 – SSH server

21 – VSFTPD Server

## **For Raven XT Sierra Wireless Modems:**

If you are using a Cellular Modem for connectivity, it’s likely that they block ports below 1024, so in the rasperryPi server:

### Add some software to rPi

If the following packages are not yet installed, install the following via SSH command-line:

sudo apt-get install modemmanager

sudo apt-get install usb-modeswitch

sudo apt-get install network-manager

### MODIFY SSH

edit /etc/ssh/sshd\_config 🡪 add a “Port 2222” below “Port 22”. This allows listening on two ports (22 locally, 2222 locally or remotely)

restart the ssh server by typing “sudo service ssh restart”

### **MODIFY FTP**

edit /etc/vsftp.conf 🡪 add a “listen\_port=2121” below “listen=YES”

restart the vsftp server by typing “sudo service vsftpd restart”

Now, after completing setup, you should be able to connect to the Gateway remotely by using the above ports.

### Modify rPi Network Settings

You should be able to just connect the Raven XT to a spare USB port, and Linux will load the correct driver, but you’ll need to edit the /etc/network/interfaces file:

Make it look similar to the following (where eth1 is the Cellular Modem):

iface lo inet loopback

iface eth0 inet dhcp

metric 200

allow-hotplug eth1

iface eth1 inet dhcp

netmask 255.255.255.0

gateway 192.168.1.1

metric 1

iface default inet dhcp

Then remotely connect to the Raven XT Settings by Web UI, by entering its static IP address, followed by :9191, for example

<http://166.154.171.128:9191>

login with username “user”, default password “12345”.

Under LAN, change USB Device mode to USBNET, Device Ethernet IP and Device USB IP and Host IP to 192.168.1.1.

You can also connect using AceManager (and using a dynamic DNS over TCP if you’ve configured a Dynamic IP address for this device (xxxxxxx.eairlink.com)).

### A Note about Cellular Gateway RSSI (dBm)

The Cellular Gateway’s link quality is largely dependent upon antenna type, antenna placement, antenna wire length, interference, availability of nearby cell sites, etc (see B&B Electronics’ “10 Commandments of Wireless Communications”).

One very good metric to gauge link quality is the RSSI value. The less negative the number, the better. A value in the range of -50 dBm is great, while a -80 to -90 dBm value is very weak (and will result in intermittent communication).

The Received Signal Strength Indicator (RSSI), is available in the AceManager Status tab. Connect remotely to this interface using a web browser as documented in the previous section.

Resources:

<https://groups.google.com/forum/#!topic/beagleboard/Z9h2TI0iEW4>

## Add remote desktop (probably overkill):

Per here: <http://www.circuidipity.com/getting-started-with-beaglebone-black.html>

opkg update

# opkg install x11vnc

# x11vnc -bg -o %HOME/.x11vnc.log.%VNCDISPLAY -auth /var/run/gdm/auth-for-gdm\*/database -display :0 -forever

# Hardware Setup/Configuration

## Items investigated or purchased:

**Motherboard HW:** Beaglebone Black, PCB Rev B6

**Daughter Board:** Circuitco RTC Cape, Revision A1 (Requires CR2032 battery)

**Daughter Board:** Adafruit Beagle Bone Proto (long Thru-Hole female-male connectors hand-soldered on)

**Board Module:** Adafruit 16-bit I2C ADC+PGA, ADS1115

**USB to MicroSD converter:** Dane Elec

**MicroSD card:** Speed Grade 4, 8Gb

**Board Module:** Mace Tech Chrono Dot V2.1, <http://adafru.it/255> (Requires CR1632 battery)

**Accessory:** 5V to USB AC adapter, 1Amp max output

**Accessory:** USB-A to Mini-B cable

**Accessory:** Ethernet Cable

**Accessory: USB to Wi-Fi dongle**

**Sensor:** All Sensors Differential Pressure Sensor, DLVR-L02D-E1ND-C-NS3N, +/- 2 inH2O SPI DIP model

**Sensor:** All Sensors Differential Pressure Sensor, DLVR-L02D-E1ND-C-N83N, +/- 2 inH2O I2C DIP model

**Sensor:** COZIR CO2 Sensor, UART based output  
**Sensor:** Analog Output K-Type Thermocouple Amplifier - AD8495 Breakout, [1778](http://www.adafruit.com/product/1778)

## Basic Soldering and Assembly:

First, plug the MicroSD card into the USB to MicroSD converter.

Plug the USB-to-MicroSD converter into the Beaglebone Black USB port

If not performed already, insert the Through-Board, Female-to-Male Board-to-Board connectors through the Adafruit Beagle Bone Proto PCB (outer through-hole 2x23 grid) from the top and solder the outer male connectors protruding out the bottom (on the bottom side)

## Power Up Sequence:

Plug the USB cable into the USB AC Adapter

Plug the AC adapter into a 120vac outlet

Plug the Ethernet cable into a local router and the Beaglebone Black

Plug the USB cable into the Beaglebone black

Watch LEDs and wait for system to become stable

# Hardware Component Notes

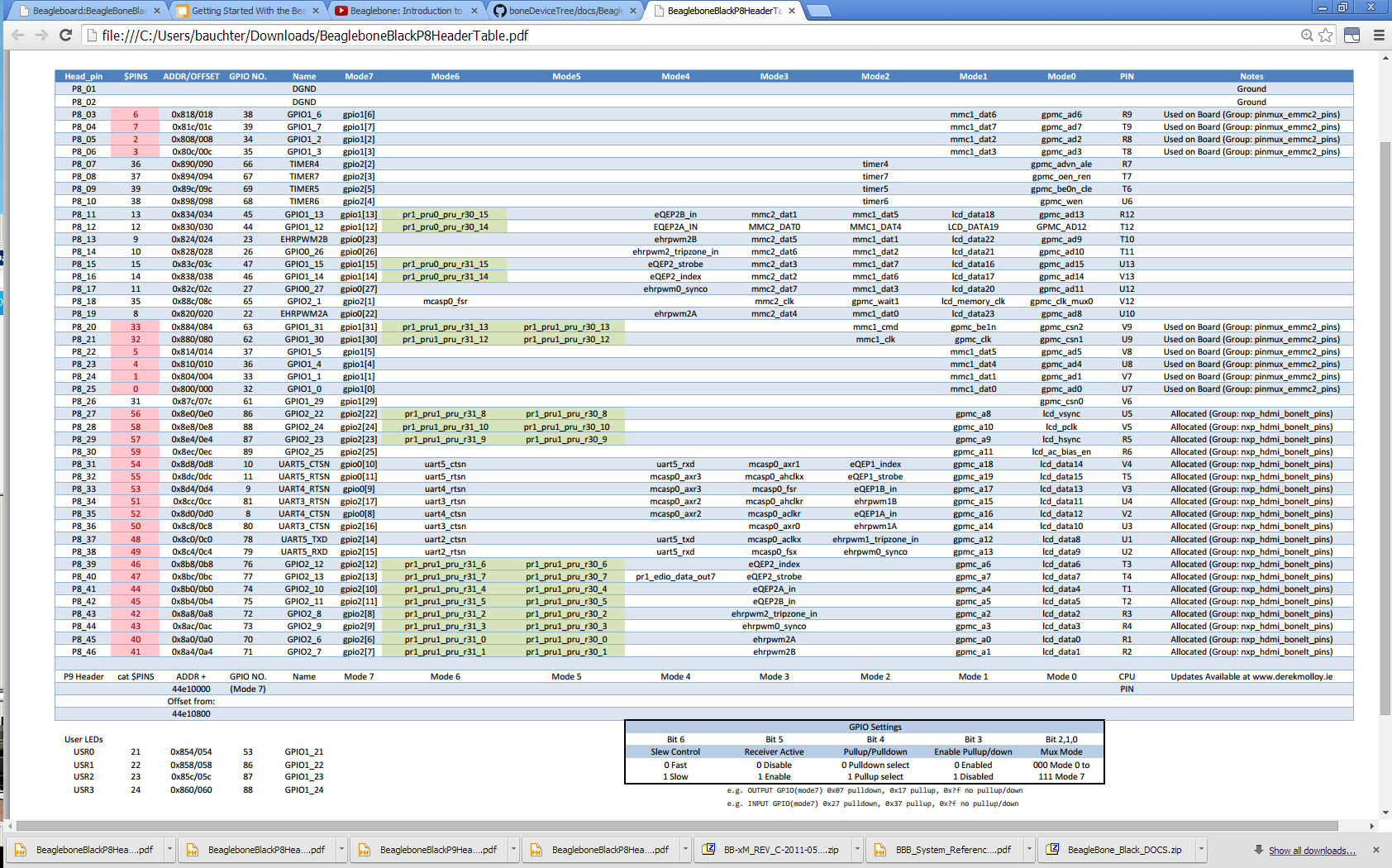
## BeagleBone Black, Rev B6

The BeagleBone Black has peripheral Pinouts for P9 and P8 as such:

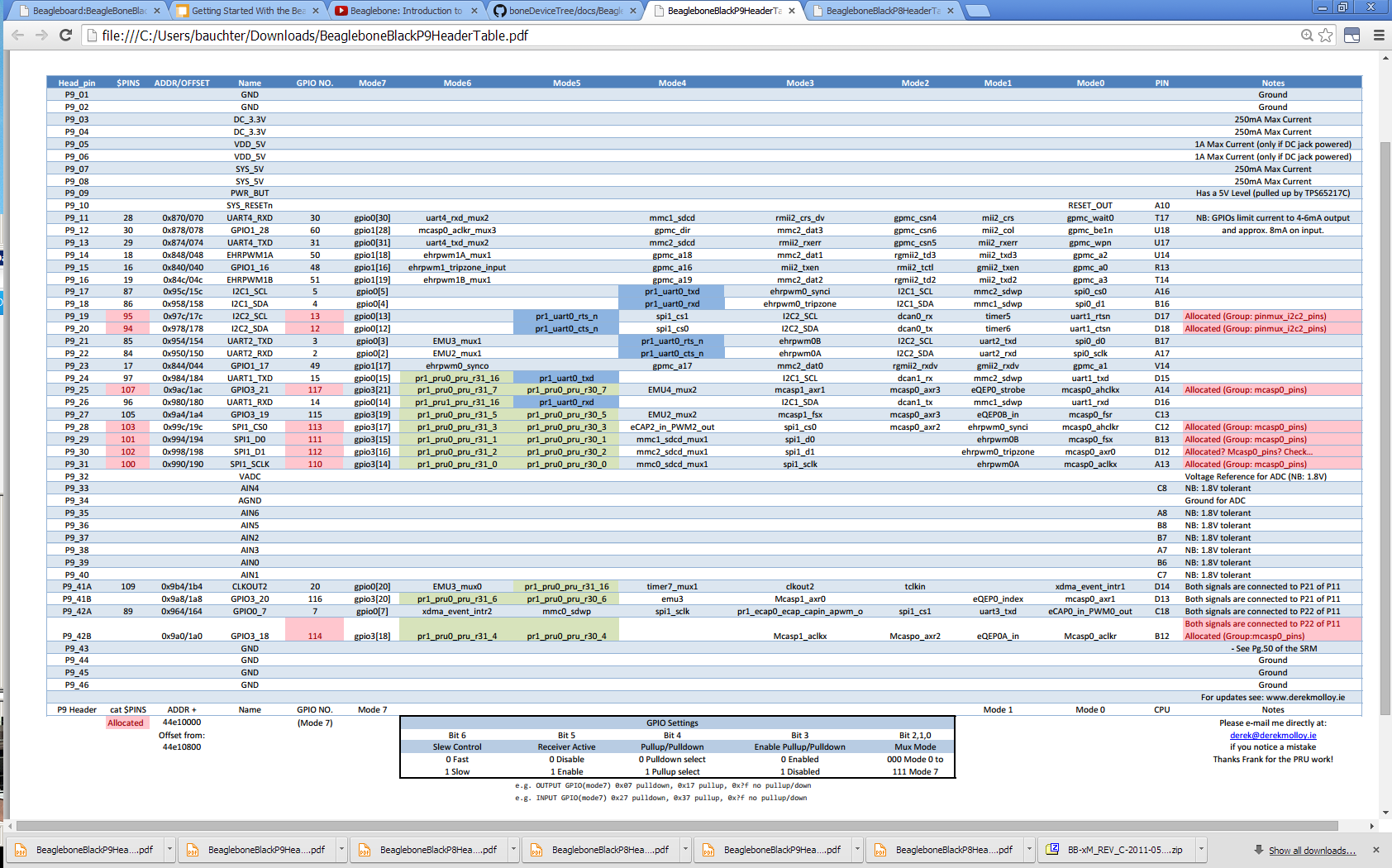


The Pinouts are described in greater detail within the datasheet, but GPIO are listed here for convenience.

P8:



P9:



## Reading GPIO

The way to read a GPIO input is to first assign it as a digital input pin to read (returning a value of ‘0’ or ‘1’). Here’s some example python code to perform a read:

import Adafruit\_BBIO.GPIO as GPIO

GPIO.setup(“P8\_14”,GPIO.IN)

print "Pin is: ",GPIO.input(“P8\_14”)

It’s also possible to wait for an edge to trigger on rising or falling in non-blocking or blocking statements. See additional resources for code to do that.

Here’s some example python code to perform an ADC Input Read:

import Adafruit\_BBIO.GPIO as GPIO

import Adafruit\_BBIO.ADC as ADC

ADC.setup()

Value = ADC.read("P9\_36") #Returns a value from 0 to 1

Voltage = Value\*1.8 #converts to a voltage value

print "Voltage is: ",Voltage," volts"

**KNOWN BUG:** Internal pull-down resistors are always enabled when using the adafruit library! See issue 5 on github. There’s a work around code snippet if internal pull-ups are needed. The nomel beaglebone may be another library to use. <https://github.com/nomel/beaglebone/tree/master/gpio-header>

Additional Resources:

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

<http://www.armhf.com/using-beaglebone-black-gpios/>

<https://github.com/adafruit/adafruit-beaglebone-io-python/issues/5>

## Writing GPIO

The writing of GPIO Digital Outputs (as values of ‘0’ or ‘1’) requires the pin mode to be set correctly and not in conflict with any other uses of that pin. Using Aadafruit’s library, that can be done using “Setup”

import Adafruit\_BBIO.GPIO as GPIO

GPIO.setup("P8\_10", GPIO.OUT)

GPIO.output("P8\_10", GPIO.HIGH)

GPIO.cleanup()

**KNOWN BUG:** Internal pull-down resistors are always enabled when using the adafruit library! See issue 5 on github. There’s a work around code snippet if internal pull-ups are needed. The nomel beaglebone may be another library to use. <https://github.com/nomel/beaglebone/tree/master/gpio-header>

Additional Resources:

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

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

<http://raspi.tv/2013/rpi-gpio-basics-3-how-to-exit-gpio-programs-cleanly-avoid-warnings-and-protect-your-pi>

<http://stackoverflow.com/questions/16872763/configuring-pins-mode-beaglebone>

<https://github.com/adafruit/adafruit-beaglebone-io-python/issues/5>

## ADS1115

The Adafruit ADS1115 16-bit, 4-channel ADC with Programmable Gain Amplification runs off of I2C. The Adafruit Python Library, Adafruit\_BBIO provides an I2C interface for communicating with the ADS1115. See the following website for usage of that library for I2C, SPI, etc:

<https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/using-the-bbio-library>

Connections:

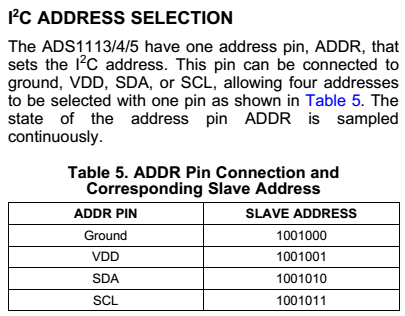
VDD – Wire to 3.3Volts – P9\_03

GND – Wire to DGND – P9\_01

SCL – Wire to I2C\_SCL – P9\_19. No need for an SCL pull-up – it’s on the board

SDA – Wire to I2C\_SDA – P9\_20, No need for an SDA pull-up – it’s on the board

ADDR – Wire as follows:



Pull the ADDR line to GND

ALRT – Leave floating

A0 – Analog Input 0, AISI1 or AIDI1+, Range: 0 to VDD

A1 – Analog input 1, AISI2 or AIDI1-, Range: 0 to VDD

A2 – Analog input 2, AISI3 or AIDI2+, Range: 0 to VDD

A3 – Analog input 3, AISI4 or AIDI2-, Range: 0 to VDD

Technical Details:

* IDE SUPPLY RANGE: 2.0V to 5.5V
* LOW CURRENT CONSUMPTION: Continuous Mode: Only 150µA Single-Shot Mode: Auto Shut-Down
* PROGRAMMABLE DATA RATE: 8SPS to 860SPS
* INTERNAL LOW-DRIFT VOLTAGE REFERENCE
* INTERNAL OSCILLATOR
* INTERNAL PGA
* I2C INTERFACE: Pin-Selectable Addresses
* FOUR SINGLE-ENDED OR TWO DIFFERENTIAL INPUTS
* PROGRAMMABLE COMPARATOR
* This board/chip uses I2C 7-bit addresses between 0x48-0x4B, selectable with jumpers.

After board bringup,

Run root@beaglebone:~/usbDrive/ADC1115# python ads1x15\_ex\_singleended.py

To verify communications and read Analog Input AISI1 (A0)

If instead you wire up more ADCs (addresses 0x48, 0x49, 0x4A, 0x4B), instead run ads1x15\_ex\_singleended\_multipleADCs.py to verify communications and read Analog Input, channel 0. This may take a little editing/ commenting out as you vary the quantity of daisy chained ADS1115’s.

To discover devices in the I2C bus, type the following in the SSH window (command line):

I2cdetect –y –r 1

## Analog Output K-Type Thermocouple Amplifier - AD8495 Breakout

The equation for converting this to a temp, in C, is as follows:

Temperature = (Vout - 1.25) / 0.005 V

As implemented on the sensor board, TC1-14 are all Vref = 0, so T = (Vout)/0.005 in C and 360\*Vout + 32 in F

However, as implemented on the Sensor board for TC15 & TC16, the design has changed to a reference point of 0.5V, and the equations are thus:

* Temperature conversion for TypeK with ADR130 voltage reference at 500mV
  + T  =  (Vout – 0.5) / 0.005  in units of deg C
  + T  =  (360\*(Vout-0.5)) + 32 in units of deg F

## Circuitco RTC Cape, Revision A1

The RTC Cape is an easy form-factor I2C interface Realtime Clock which provides accurate time keeping in the event of a power outage or lack of connectivity to the internet (NTP). This cape features the DS1338 I2C-based real-time clock/calendar and a holder for a coin battery type CR2032. Setup, reading, and setting up time is easy with the provided device tree overlay.

Uses the following two P9 pins (in addition to GND and VDD):

I2C1\_SCL\_P9\_17

I2C1\_SDA\_P9\_18

The product is available via Digikey, product interface website here:  
<http://elinux.org/CircuitCo:RTC_Cape>

1. Plug the RTC Cape into the Beaglebone black
2. Power up the BBB and SSH into it as root
3. Download the device tree files and extract them to /lib/firmware:

cd /lib/firmware

wget https://github.com/CircuitCo/RTC-Cape/raw/docs/BBB-RTC-01-00A1.tar.gz --no-check-certificate

tar xvzf BBB-RTC-01-00A1.tar.gz

1. Load the device tree overlay for the RTC Cape:

echo BBB-RTC-01:00A1 > /sys/devices/bone\_capemgr\*/slots

1. Check to ensure the overlay is loaded. In the messages below, the DS1338 is registered to rtc1:

dmesg | tail

[ 5721.844355] bone-capemgr bone\_capemgr.9: slot #7: 'Override Board Name,00A1,Override Manuf,BBB-RTC-01'

[ 5721.844607] bone-capemgr bone\_capemgr.9: slot #7: Requesting part number/version based 'BBB-RTC-01-00A1.dtbo

[ 5721.844656] bone-capemgr bone\_capemgr.9: slot #7: Requesting firmware 'BBB-RTC-01-00A1.dtbo' for board-name 'Override Board Name', version '00A1'

[ 5721.847380] bone-capemgr bone\_capemgr.9: slot #7: dtbo 'BBB-RTC-01-00A1.dtbo' loaded; converting to live tree

[ 5721.848460] bone-capemgr bone\_capemgr.9: slot #7: #3 overlays

[ 5721.860065] omap\_i2c 4802a000.i2c: bus 2 rev0.11 at 100 kHz

[ 5721.902076] rtc-ds1307 2-0068: rtc core: registered ds1338 as rtc1

[ 5721.902147] rtc-ds1307 2-0068: 56 bytes nvram

[ 5721.902274] omap\_i2c 4802a000.i2c: unable to select pin group

[ 5721.902468] bone-capemgr bone\_capemgr.9: slot #7: Applied #3 overlays.

1. Read the time from the RTC chip. It should look like something below since the time has not been set:

hwclock -r -f /dev/rtc1

Sat 01 Jan 2000 12:06:17 AM UTC -1.004531 seconds

1. Update the system time of the Beaglebone black (internet connection required) and write that to the RTC chip using hwclock command:

ntpdate -b -s -u pool.ntp.org

hwclock -w -f /dev/rtc1

1. Read the time again. This time it should have been updated.

hwclock -r -f /dev/rtc1

Fri 16 May 2014 10:57:12 PM UTC -0.755959 seconds

This device should now be setup to keep system time even if the system resets, loses power, etc.

However, it needs to be setup each time the BBB loads.

Therefore, the following command needs to be executed every boot up:

echo BBB-RTC-01:00A1 > /sys/devices/bone\_capemgr\*/slots

Instead, create a service that will run each time the BBB boots.

1. Create a directory and script that will be executed:

mkdir /usr/share/rtc\_cape

nano /usr/share/rtc\_cape/clock\_init.sh

1. Add the following to clock\_init.sh:

echo BBB-RTC-01:00A1 > /sys/devices/bone\_capemgr\*/slots

hwclock -s -f /dev/rtc1

hwclock -w

1. Next, create a service that will start on boot, and execute this script that was created:

nano /lib/systemd/system/rtc-cape.service

1. Copy the following contents into that file, and save it:

[Unit]

Description=DS1338 RTC Cape Service

[Service]

Type=simple

WorkingDirectory=/usr/share/rtc\_cape

ExecStart=/bin/bash clock\_init.sh

SyslogIdentifier=rtc\_cape\_ds1338

[Install]

WantedBy=multi-user.target

1. After saving the file, we'll need to actually enable the service so it starts each time as the system boots:

systemctl enable rtc-cape.service

1. You can always manually start and stop the service as well:

systemctl start rtc-cape.service

systemctl stop rtc-cape.service

1. That's it! Reboot your system and check that it all works:

shutdown –r now

Additional Resources:

<https://learn.adafruit.com/adding-a-real-time-clock-to-beaglebone-black/overview>

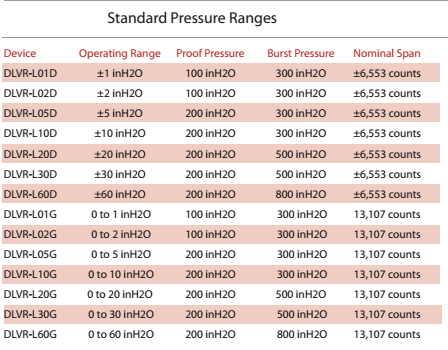
## All Sensors Differential Pressure Sensor, SPI interface, DLVR-L02D-E1ND-C-NS3N and I2C, DLVR-L02D-E1ND-C-NI3N

The DLVR Series Mini Digital Output Sensor is based on All Sensors’ CoBeam Technology. This reduces package stress susceptibility, resulting in improved overall long term stability. The technology also vastly improves position sensitivity compared to single die devices.

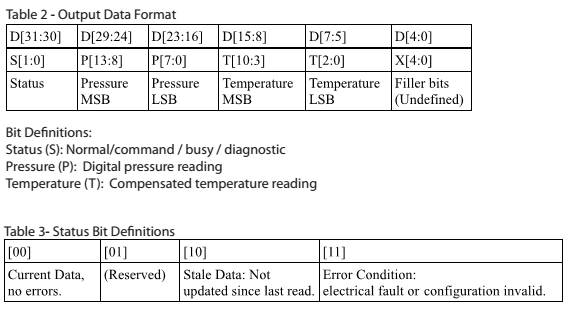
Features:

* 1 to 60 inH2O Pressure Ranges
* 3.3V Supply Voltage Standard / 5V Option
* I2C Standard Interface / SPI Interface Option
* Better than 1.0% Accuracy Over Temperature Typical

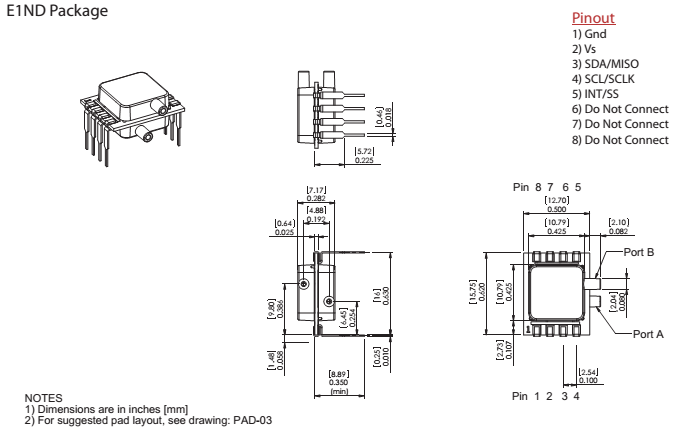
Here are the Pressure Ranges that can be ordered:



The digital output format is as follows:



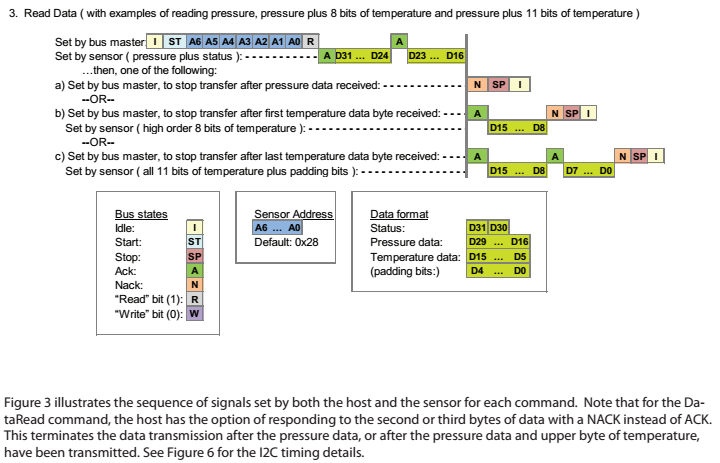
The E1ND Package and Pinout is as follows:



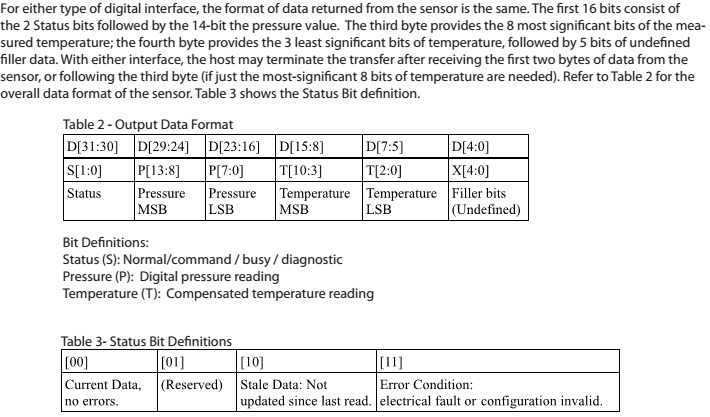
**The I2C model information (**DLVR-L02D-E1ND-C-NI3N)**:**

The default (manufacturer) I2C address is: 0x28

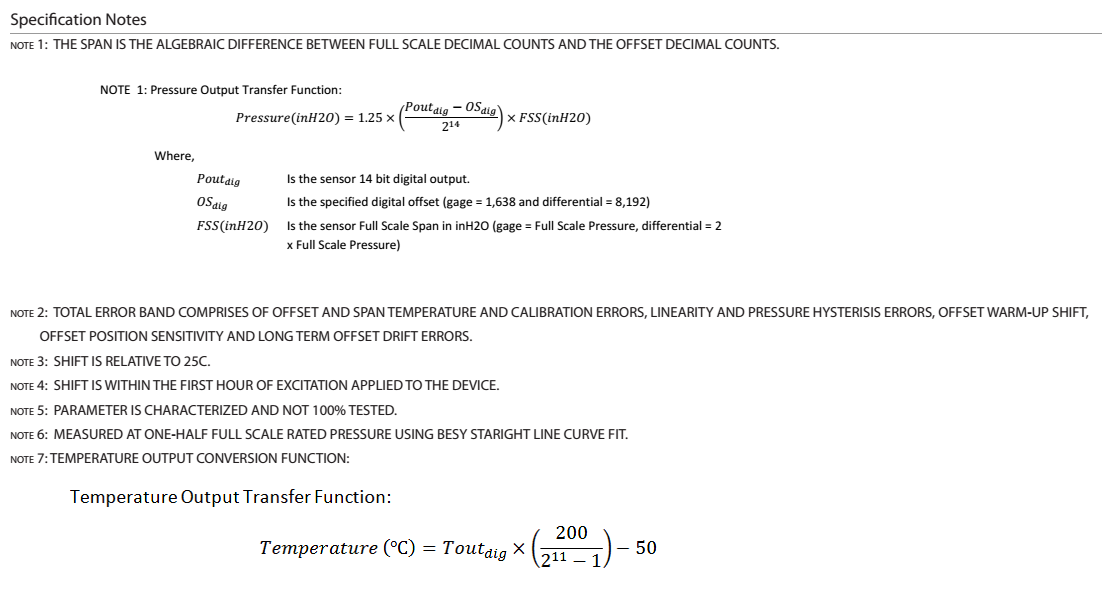
Here’s a typical communication diagram for Reading Data (default it is continually collecting at a fast rate):



And the format is as follows:



And conversion of output to real values is as follows:



To test, wire up the following connections to the BBB.

Connections:

With the BBB system off, Wire the Following:

BBB P9\_19 (SPI1, SCL) to Sensor Pin 4 (SCL)

BBB P9\_20 (SPI1,SDA) to Sensor Pin 3 (SDA)

BBB P9\_01 (DGND) to Sensor Pin 1 (Gnd)

BBB P9\_03 (3.3V VDD) to Sensor Pin 2 (Vs)

To discover devices in the I2C bus, type the following in the SSH window (command line):

I2cdetect –y –r 1

If 0x28 shows up, then you’re seeing the device available on the I2C line.

Type:

I2cget –y 1 0x28 0

To see if a high level byte read works.

If so, move on to running the python code: /usbDrive/all\_sensors\_DS-0300/I2C\_DS0300\_test.py

The response will provide a list output of, for example (in open air):

[32,15,102,228]

Converting these into hexadecimal yields:

20, 0F, 65, E4 and converting that into binary yields:

0010 0000 0000 1111 0110 0101 1110 0100

\/\---------------/ \-----------/\-----> filler bits

| | ^-Temp Reading

| Pressure Reading

Status Bit

Therefore yielding

**A Status Reading of**: 0b00 means “Current Data, No Errors”

**a Pressure Reading of:** 0b10000000001111 has decimal value 8207

…. Placing this into the output conversion yields:

or = 0.00458 inH20

**A Temperature reading of:** 0b01100101111 has decimal value of 815

…. Placing this into the output conversion yields:

or Temp(in deg. C) = **29.6 deg. C** or **85.3 deg.F**

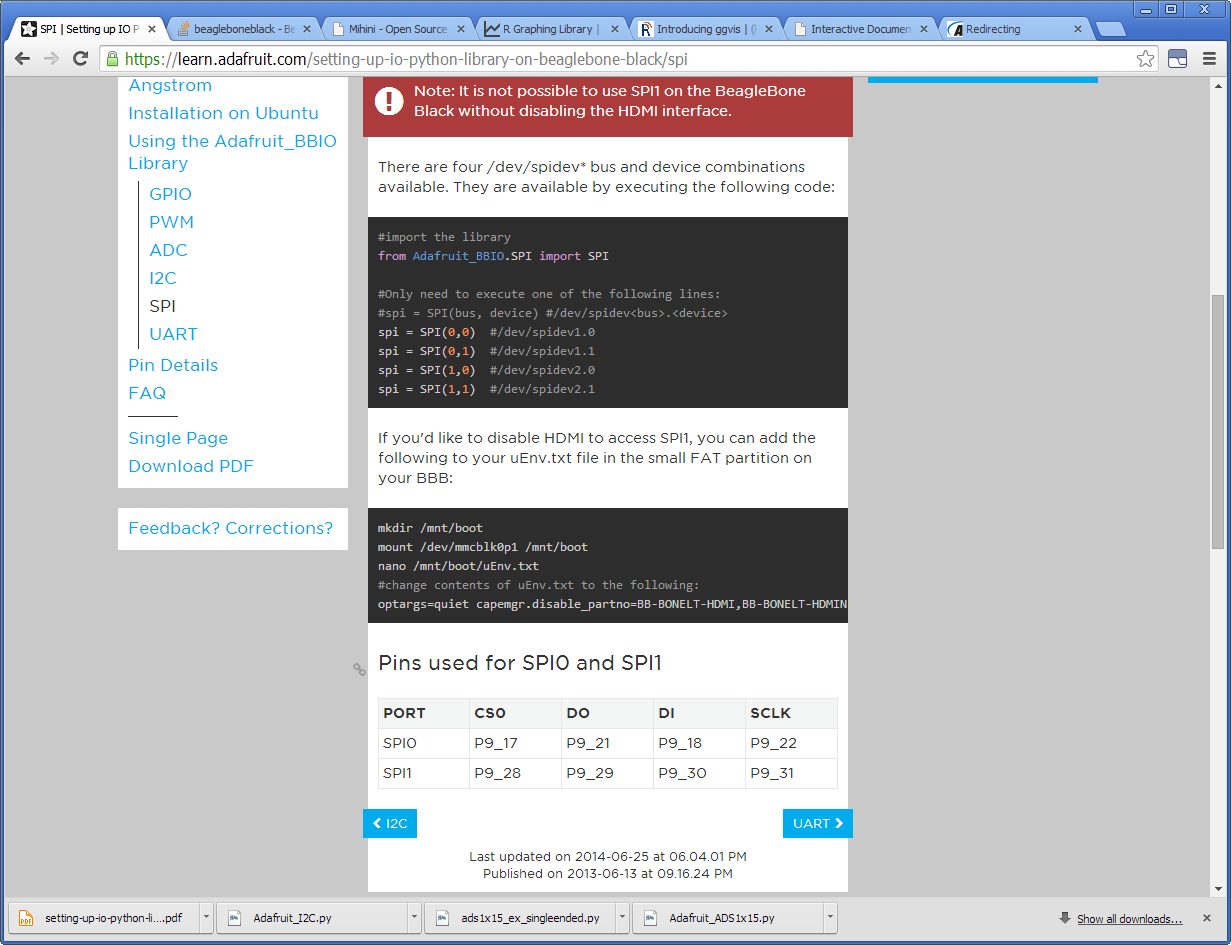


Be sure to only poll the Pressure sensor at the rate it is updated (see above figure)…

For Low Power, update time is 6mS, with a 9.5mS delay when it does an internal check every 31 cycles.

**Some SPI-model information:**

The breakout SPI port 0 IO on the Beaglebone black shares IO with the I2C port. Because the ADS and RTC Cape use up these I2C pins, we need to use the SPI Port 1 I/O. Here are the Pins used for both SPI0 and SPI1:



Connections:

So, With the BBB system off, Wire the Following:

BBB P9\_28 (SPI1, CSO) to Sensor P5 (INT/SS)

BBB P9\_30 (SPI1,DI) to Sensor P3 (SDA/MISO)

BBB P9\_31 (SPI1,SCLK) to Sensor P4 (SCLK)

BBB P9\_01 (DGND) to Sensor P1 (Gnd)

BBB P9\_03 (3.3V VDD) to Sensor P2 (Vs)

The BeagleBone Black by default reserves the SPI1 pins for an HDMI port, so in order to use SPI1, the HDMI Port Designations need to be disabled.   
To do this, 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

#reboot to take effect

shutdown –r now

2014-07-14 - To date, SPI communication has not been successful with this chip (using drivers from adafruit). Other drivers may improve this, but in the meantime I2C is the option.

Additional Resources:

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

<http://hipstercircuits.com/enable-spi-with-device-tree-on-beaglebone-black-copy-paste/>

<http://stackoverflow.com/questions/21276090/beaglebone-black-enable-spi-interface>

<http://elinux.org/BeagleBone_Black_Enable_SPIDEV>

<https://groups.google.com/forum/#!topic/beagleboard/ZoJFqmXraCw>

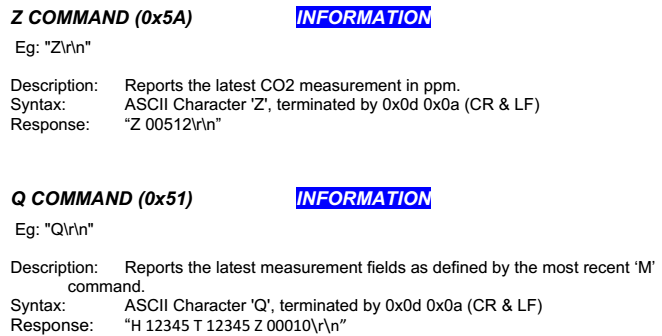
<http://tekuconcept.blogspot.com/2014/05/i2c-with-beaglebone-black.html>

## COZIR CO2 Sensor, UART based output

The CO2 Sensor is a UART based output.

Sending an ASCII output to the COZIR Sensor of “Z\r\n” will yield a response of “Z <#PPM>\r\n” with the output being PPM of CO2 against a reference background PPM (Default of 450ppm)

Sending an ASCII output to the COZIR Sensor of “Q\r\n” will yield a response of “H 12345 T 12345 Z 00010\r\n”, where H is the relative humidity, T is the Temperature, and Z is the PPM of CO2, measured once every ½ second.



## Wireless Linked Logger, Digi module

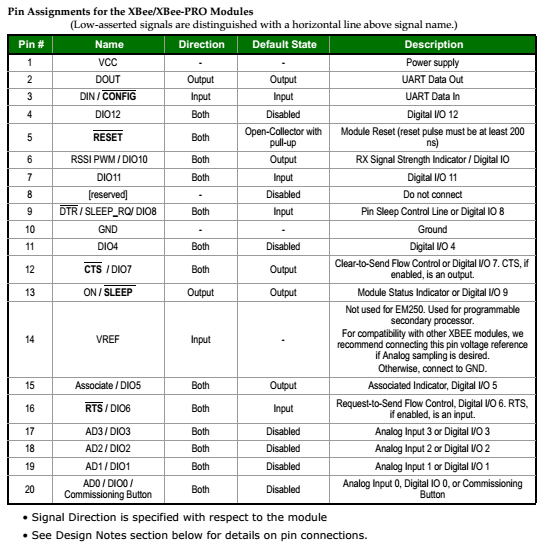
This logger would link to the main logger or to an external server.

Every so often it would record the value of just one or two inputs (like a current level-triggered switch) and write it to storage locally.

Every so often it would send this data back to either the main logger or to an external server.

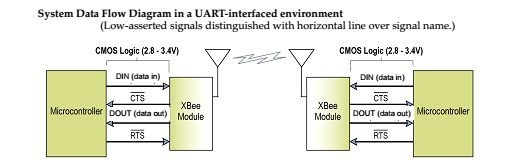
It would run on battery only as it would be in a wall or bedroom.

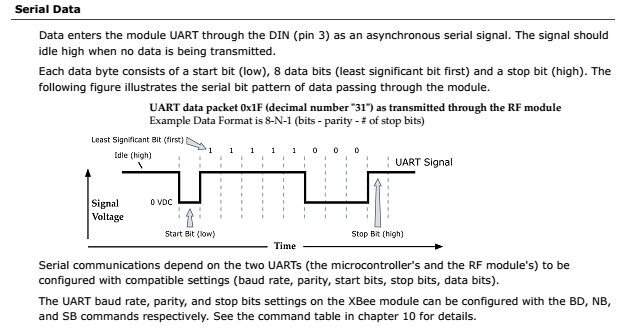
It would have to draw very little power.

2014-08-15 – it was decided that we go with a Digi XBee Pro type device (assuming the S2B). The pinouts for these devices are as follows:  


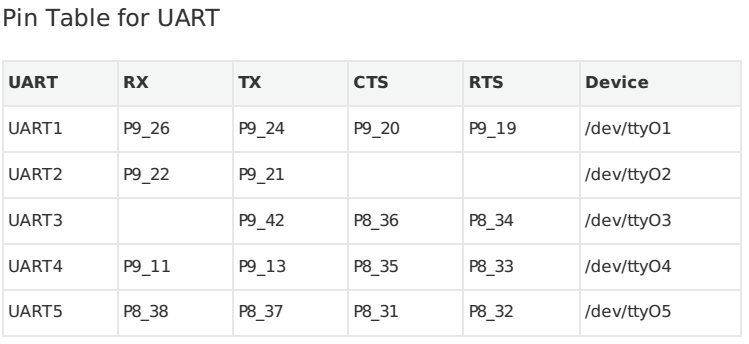
As default, these modules operate at 9600 baud, 8-N-1

Here is a dataflow (example diagram):



Example UART Data flow:  


To test interface to the Digi module, first a UART needs to be wired to the coordinator Digi device. UARTS have the following Pinouts for the Beaglebone Black:



Install the pyserial module if it isn’t already installed:

pip install pyserial

Wire the following up:

UART4 – P9\_11 to UART2 – P9\_21

UART4 – P9\_13 to UART2 – P9\_22

Next, export the UART4 and UART2 in the python interpreter with the Adafruit IO Library:

Python

>>> import Adafruit\_BBIO.UART as UART

>>> UART.setup(“UART4”)

>>> UART.setup(“UART2”)

>>> exit()

And make a small python file that includes the following:

import Adafruit\_BBIO.UART as UART

import serial

UART.setup("UART4")

ser = serial.Serial(port = "/dev/tty4" , baudrate=9600)

ser.close()

ser.open()

if ser.isOpen():

print "Serial is open!"

ser.write("Hello World!")

ser.close()

#UART.cleanup()

Open up a new, separate Putty Window, and create a minicom for observing ttyO2:

minicom -b 9600 -D /dev/ttyO2

Now, when running the small python file in your original ssh window, you should see a “Hello World!” getting output to the ttyO2 port (UART2).

**If all of this works, it’s time to wire up the Digi Xbee.**

Place the Xbee on a breadboard and wire the following:

P9\_11 on BBB (UART4 RX) to DOUT on Xbee

P9\_13 on BBB (UART4 TX) to DIN on Xbee

P8\_35 on BBB (UART4 CTS) to RTS on Xbee (optional?)

P8\_33 on BBB (UART4 RTS) TO CTS on Xbee (optional?)

GND to VSS  
3.3VDC to VCC on Xbee

5VDC to VDD on Xbee

Once powered and wired to UART4 on the Beaglebone black, this device should be interactive with the BBB. The other side should be hooked to a laptop via a USB port.

Code can now be written to interact with the two Digis.

Resources:

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

<http://learn.parallax.com/propeller-c-simple-protocols/full-duplex-serial>

Ideas previously investigated:

* Point Six wireless may be a solution.
* Use an Arduino and wirelessly linked shield.
* Use an iMonnit gateway
* Use a Seeeduino with Digi (for RTC and local storage)

<https://learn.adafruit.com/low-power-wifi-datalogging/overview>

<https://lowpowerlab.com/shop/index.php?_route_=moteino-r4>

Pin Allocation

## USB to Wi-Fi Dongle

The USB to Wi-Fi Dongle is a necessary component for when a hardwired Ethernet internet connection is not an option. The UWN200 has been verified to work with the AT&T Unite Pro cellular mi-fi connection.

The Wi-Fi Dongle evaluated is the Logic Supply component:

|  |  |  |  |
| --- | --- | --- | --- |
| **Compact USB Wi-Fi Adapter with 4" Antenna** | UWN200 | 3 | $38.85 |

Setting this interface up is a matter of matching the WiFi settings with a known nearby wi-fi access point.

A warning is that a 5V, 2A power supply is recommended to ensure no droop or brownout while the USB dongle is connected.

Another warning is that the HDMI port can cause interference on the Wi-Fi, giving you some issue. Space the Wi-Fi away or disable the HDMI interface.

1. Plug in the USB to Wi-Fi adaptor
2. If a longer term installation, setup the enable/disable of the Wi-Fi device using a script as in the logic supply or adafruit resources below.
3. Reboot to detect the Wi-Fi adaptor
4. Follow the instructions on the logicsupply website for installation (see Resources)

**Resources:**

<http://inspire.logicsupply.com/2014/07/beaglebone-wifi-installation.html>

<https://learn.adafruit.com/setting-up-wifi-with-beaglebone-black/overview>