PANDORA PRODUCTS

Beagle Bone, Yocto & Other Tricks
BBB Tricks

Jim Schimpf

Document Number: PAN-2015090001
Revision Number: 2.0
13 Januray 2016

Pandora Products.
215 Uschak Road
Derry, PA 15627

Pandora Products.

215 Uschak Road

Derry, PA 15627

Phone: 724-539.1276

Email: jim.schimpf@gmail.com

Document Revision History

Version	Author	Description	Date
0.1	js	Initial Version	2-Sep-2015
0.2	js	Create SD card	4-Sep-2015
0.3	js	Disable/Enable MMC	4-Sep-2015
0.4	js	Build and deploy	5-Sep-2015
0.5	js	Tutorials and Hob	5-Sep-2015
0.6	js	Hob information	19-Sep-2015
0.7	js	Add info about adding recipe to local.conf	8-Oct-2015
0.8	js	Creating a new recipe	8-Oct-2015
0.9	js	Getting elliot recipe working	10-Oct-2015
1.0	js	Adding a recipe layers	18-Oct-2015
1.1	js	Autotools notes	23-Oct-2015
1.2	js	Revise recipe section	28-Oct-2015
1.3	js	Building new image with libmill	1-Nov-2015
1.3	js	Add Beagle Bone use section	2-Nov-2015
1.4	js	Fix setting Boot partition flag	11-Nov-2015
1.5	js	Ubuntu SD card	11-Nov-2015
1.6	js	Spell check & formatting fixes	11-Nov-2015
1.7	js	Add opencv	14-Nov-2015
1.8	js	Bonjour advertise ssh & sftp	14-Nov-2015
1.9	js	New tiltle and emphasis	15-Dec-2015
2.0	js	Add Sodium library	13-Jan-2016

Contents

1 Materials **2**
1.1 Hardware 2
1.1.1 Beagle Bone Black 2
1.1.2 USB <-> Serial Converter 2
1.1.3 SD Card(s) 3
1.1.4 USB <-> SD Card Interface 3
1.1.5 USB Mini Cables 4
1.1.6 Backup boot SD 4
1.1.6.1 Testing 4
1.2 Software 5
1.2.1 Ubuntu 5
1.2.2 Tools 5
1.2.3 Yocto 5
1.3 Building the TEST SD card 6
1.3.1 Finding your card. 6
1.3.2 Starting gparted 7
1.3.3 Boot Partition 7
1.3.4 Mark as boot 8
1.3.5 Root Partition 9
1.3.6 Do it 9

2 Build and deploy **10**
2.1 Initial Build 10
2.2 Deploy 11
2.2.1 Introduction 11
2.2.2 Copying the files 11

3 Running BBB **12**
3.1 Running with an SD card 12
3.2 Setting the board to ALWAYS boot from the SD card 12
3.3 Restoring MMC Boot 14

4 Customizing Yocto **14**
4.1 Introduction 14
4.2 Tutorials 14
4.3 Creating a new layer 15
4.3.1 Create the folder for your new layer 15
4.3.2 Add the configuration 15
4.3.3 Adding the layer 15
4.4 Creating a recipe for LIBMILL 16
4.4.1 Introduction 16
4.4.2 Header information 16
4.4.3 Getting the SRCREV 16
4.4.4 Getting the license hash 17
4.4.5 The recipe 17
4.4.6 Testing 17
4.5 Creating a recipe for SODIUM 18
4.5.1 Introduction 18
4.5.2 Header information 18
4.5.3 Getting the SRCREV 18
4.5.4 Getting the license hash 18
4.5.5 The recipe 19
4.5.6 Testing 19
4.5.7 Use 19
4.6 Building a board image 20
4.6.1 Introduction 20
4.6.2 Adding C tools 20
4.6.3 QEMU Removal 20
4.6.4 Lua and Libmill addition 20
4.6.5 Building the image 21
4.7 Hob 21
4.7.1 Adding new items 21
4.7.2 Advanced configuration 22
4.8 Layers and Recipes 22

5 Beagle Bone Use **25**

5.1 Introduction 25

5.2 GPIO 25

5.2.1 Introduction 25

5.2.2 Creating pin devices 25

5.3 Internet Monitor 26

5.3.1 Introduction 26

5.3.2 Development 26

5.3.3 Problems 26

5.4 Ubuntu SD Card 27

5.4.1 Get the SW 27

5.4.2 Put the SW on the SD card 28

5.4.3 Next Steps 28

5.5 Installing opencv on Ubuntu 28

5.5.1 Introduction 28

5.5.2 Precursors 28

5.5.3 Build directories 29

5.5.4 Clone opencv 29

5.5.5 Build opencv 29

5.5.6 Install opencv 29

5.5.7 Make python linkages 29

5.6 Advertising services via avahi 29

5.6.1 Introduction 29

5.6.2 Install avahi 30

5.6.3 Create Config file for SSH/SFTP 30

5.6.4 Restart the daemon 30

List of Figures

1 Beagle Bone Black 2
2 USB <-> Serial Device 3
3 SD Card 3
4 USB <-> SD 4
5 Angstrom boot 5
6 New Partition warning 7
7 Boot partition 8
8 Boot Flag 8
9 root Partition 9
10 Finish 10
11 MMC Boot partition 13
12 MMC Modified 13
13 Hob 21
14 Setting for just tar.bz2 22
15 hob layers 24
16 Adding lua 24
17 Advertised services in Bonjour Browser 31

List of Tables

Introduction

It's said that we see so far because we stand on the shoulders of giants. In this case the giants are Yocto and Beagle Bone. Yocto (http://yoctoproject.org) is a tool that builds bootable Linux images and the tool chain needed for the build from a description.[2] Beagle Bone (http://beagleboard.org/bone) is an Arduino sized board with a 750 Mhz ARM chip and 256 Meg of RAM.

The board is an excellent platform for various systems, we will show how to buiild fully customized images using Yocto and how to customize existing systems like Ubuntu.

Ever since I read Building Embedded Linux Systems[2] I wanted to try that. A combination of lack of time and lack of Linux skills caused me to never get it done. When I saw the article http:

//android.serverbox.ch/?p=1273 I knew this would be possible. The use of Yocto meant that most of the building of the tool chain and gathering of sources would be done for me. So the rest of this paper is copy of the work done by Andreas Rudolf ,Manuel Di Cerbo and Matthias Meier of the University of Applied Sciences Northwestern Switzerland with more details added where I misunderstood their directions.

1 Materials

1.1 Hardware

1.1.1 Beagle Bone Black

The board shown here has ethernet, USB both in and out, HDMI and a mini SD card slot. It comes pre-programmed with an Angstrom distribution of Linux. The distribution comes with a built in web server and built in manuals for the board. It also includes a Javascript IDE that can exercise the hardware.

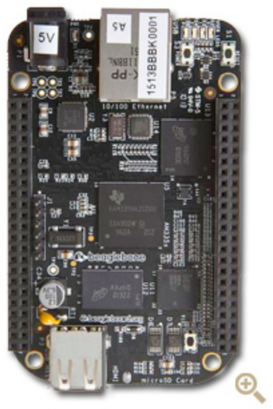


Figure 1: Beagle Bone Black

1.1.2 USB <-> Serial Converter

This is a device from Adafruit that fits on the 6 pin header on the left side of the board and gives the board a serial console. It can be found here (https://www.adafruit.com/products/284). Normally you don't need one of these to use the BBB. In our case since we are building new (and untested) bootable images, you need it so you can see if the build worked or how it failed. Shown here is it inserted on the board (Note pin on the header and pin 1 on the FTDI Basic is GND).

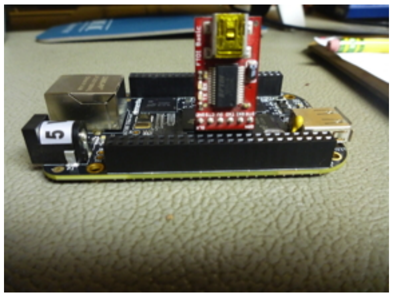


Figure 2: USB <-> Serial Device

In use your terminal program should be set for 115,200 baud.

1.1.3 SD Card(s)

The BBB will boot from these. You will need the micro SD card + an adapter so it can be put in a standard slot. When purchasing these buy the smallest available > 4GB. You won't need any bigger so spend less. You will need at least 2 of these.

Figure 3: SD Card

1.1.4 USB <-> SD Card Interface

This device is used to allow your PC to interface with the SD card. It makes the SD card look like a USB drive. The device shown is a rather fancy multiple adapter for many types of devices but any SD <-> USB device will work.

Figure 4: USB <-> SD

1.1.5 USB Mini Cables

You will need two USB mini cables, one for the USB <-> Serial converter and another to connect to and power the BBB.

1.1.6 Backup boot SD

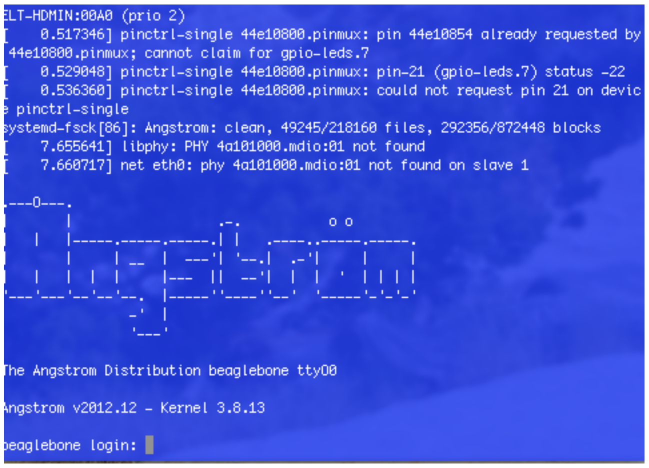
This is making a boot SD from a known good image which will allow you to recover if you have messed up the image in the BBB's flash. Go here: `http://downloads.angstrom-distribution.org/demo/beaglebone/` and download Angstrom-Cloud9-IDE-GNOME-eglibc-ipk-v2012.12-beaglebone-2013.09.05.img.xz. On OSX you can expand this with the The Unarchiver App (Apple Mac Store). You then can use the Pi Filler App `http://ivanx.com/raspberrypi/` which builds a bootable image on an SD card.

The advantage of this particular image is that when run it lets you see the MMC boot disk and modify files there. (Trust me this will be important later).

1.1.6.1 Testing

- Unplug the power USB line to the BBB.
- Put on the USB <-> Serial board and connect to it with a terminal program set to 115200 baud.
- Insert the Backup SD Micro card you just made.
- Now hold down the single switch at the Ethernet connector end of the BBB.
- Plug in the power USB line to the BBB while holding the switch.
- You should then see the boot process in the console (USB<->serial line).

If all this works label the card and put it aside for later use. It should get to here: (login is root)



```
ELT-HDMIN:00A0 (prio 2)
[    0.517346] pinctrl-single 44e10800.pinmux: pin 44e10854 already requested by
 44e10800.pinmux; cannot claim for gpio-leds.7
[    0.529048] pinctrl-single 44e10800.pinmux: pin-21 (gpio-leds.7) status -22
[    0.536360] pinctrl-single 44e10800.pinmux: could not request pin 21 on devic
e pinctrl-single
systemd-fsck[86]: Angstrom: clean, 49245/218160 files, 292356/872448 blocks
[    7.655641] libphy: PHY 4a101000.mdio:01 not found
[    7.660717] net eth0: phy 4a101000.mdio:01 not found on slave 1

.---O---.
|       |                  .-.           o o
|   |   |-----.-----.-----.| |   .----..-----.-----.
|       |     | __  |  ---'| '--.|  .-'|     |     |
|   |   |  |  |     |---  ||  --'|  |  |  '  | | | |
'---'---'--'--'--.  |-----''----''--'  '-----'-'-'-'
                -'  |
                '---'

The Angstrom Distribution beaglebone tty00

Angstrom v2012.12 - Kernel 3.8.13

beaglebone login:
```

Figure 5: Angstrom boot

1.2 Software

1.2.1 Ubuntu

You need a version of this running on real or virtual hardware. In my case I used 14.02 but I don't think the version if reasonably current (12 or 14) would make much difference.

1.2.2 Tools

You need a collection of tools and this apt-get will get them all at one go.

```
sudo apt-get -y install chrpath gawk diffstat texinfo g++ git gparted
```

1.2.3 Yocto

You need to make a working area (yocto) and then use git to pull in the daisy version.

```
mkdir yocto
cd yocto
git clone -b daisy git://git.yoctoproject.org/poky.git
```

1.3 Building the TEST SD card

This is the card you will use for booting software made with Yocto. You are going to format the card into two partitions, a small boot partition that will be used to bring the BBB up and a second which will be the rest of the card that will hold the Linux root file system. To make this you will a number of tools on the Ubuntu system.

This can be done before we use Yocto and once prepped you won't have to do this again unless you get a new SD card.

1.3.1 Finding your card.

Before you plug the card into your linux system, run the program lsblk to list all the drives in your system. You will get an output like this:

```
jim@ubuntu:~/yocto/poky/BBB$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
fd0      2:0    1     4K  0 disk
sda      8:0    0   100G  0 disk
├─sda1   8:1    0  95.9G  0 part /
├─sda2   8:2    0     1K  0 part
└─sda5   8:5    0   4.1G  0 part [SWAP]
sr0     11:0    1  1024M  0 rom
sr1     11:1    1  1024M  0 rom
```

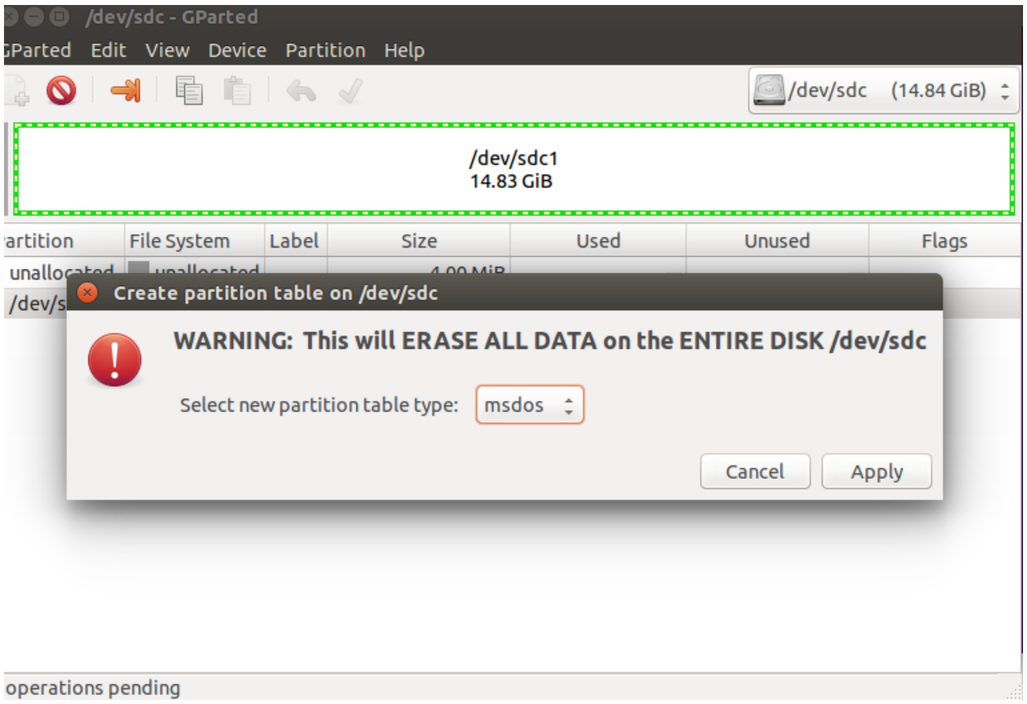
Yours might be different but we will only be interested what NEW drive appears when the SD card is plugged in. Running it then we get:

```
jim@ubuntu:~/yocto/poky/BBB$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
fd0      2:0    1     4K  0 disk
sda      8:0    0   100G  0 disk
├─sda1   8:1    0  95.9G  0 part /
├─sda2   8:2    0     1K  0 part
└─sda5   8:5    0   4.1G  0 part [SWAP]
sdc      8:32   1  14.9G  0 disk
└─sdc1   8:33   1  14.9G  0 part /media/jim/NUT
sr0     11:0    1  1024M  0 rom
sr1     11:1    1  1024M  0 rom
jim@ubuntu:~/yocto/poky/BBB$
```

It's easy to see that /dev/sdc is our SD card. You really want to be sure of this because the next steps are going to erase everything on it.

1.3.2 Starting gparted

Start gparted, don't forget to say **sudo gparted** as it must be run as superuser. After you start it pick the disk to use (in our case /dev/sdc, YMMV). Then under device pick New Partition table and you will get this warning.


/dev/sdc - GParted
GParted Edit View Device Partition Help
/dev/sdc (14.84 GiB)
/dev/sdc1
14.83 GiB
Partition File System Label Size Used Unused Flags
unallocated unallocated 4.00 MiB
/dev/s
Cancel
Apply
operations pending

Create partition table on /dev/sdc

WARNING: This will ERASE ALL DATA on the ENTIRE DISK /dev/sdc

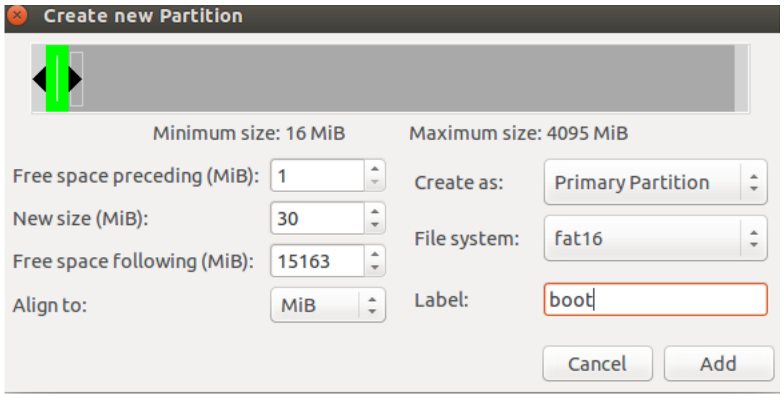
Select new partition table type: msdos

Figure 6: New Partition warning

Make sure again this the right disk and if so pick Apply.

1.3.3 Boot Partition

Then under partition pick new and you will get this dialog. Fill it out as shown.



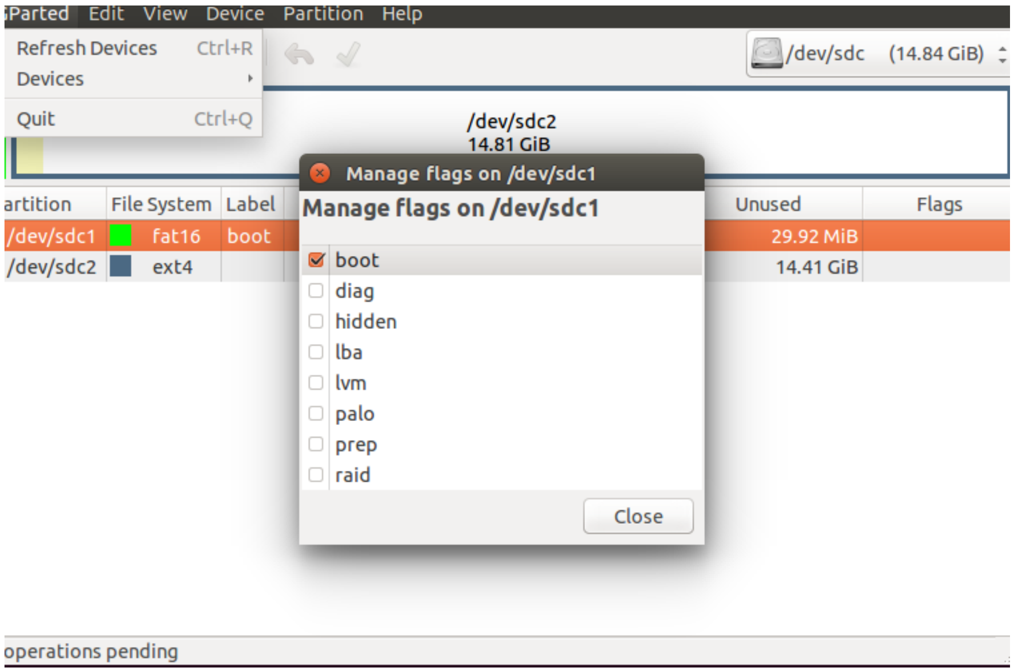
Free space preceding (MiB): 1
New size (MiB): 30
Free space following (MiB): 15163
Align to: MiB
Create as: Primary Partition
File system: fat16
Label: boot

Figure 7: Boot partition

This is a VERY small partition. Then click Add.

1.3.4 Mark as boot

You must mark this partition as a boot partition or it won't start the BBB. Right click the partition and pick Manage Flags, you will get the following dialog. Pick boot for this partition. If you don't do this the SD card won't boot.



artition	File System	Label	Unused	Flags
/dev/sdc1	🟩 fat16	boot	29.92 MiB	
/dev/sdc2	■ ext4		14.41 GiB	

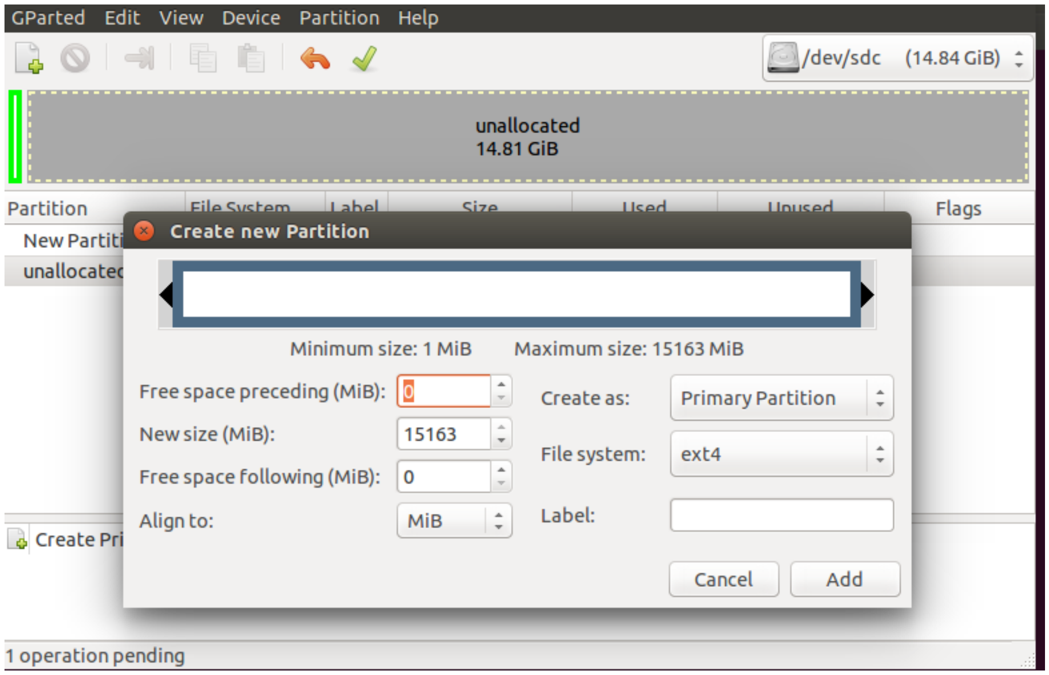
Manage flags on /dev/sdc1

- [✓] boot
- [ ] diag
- [ ] hidden
- [ ] lba
- [ ] lvm
- [ ] palo
- [ ] prep
- [ ] raid

Figure 8: Boot Flag

1.3.5 Root Partition

This partition will be an ext4 and will be the rest of the disk. Pick new partition again and fill it out as shown:



Minimum size: 1 MiB Maximum size: 15163 MiB

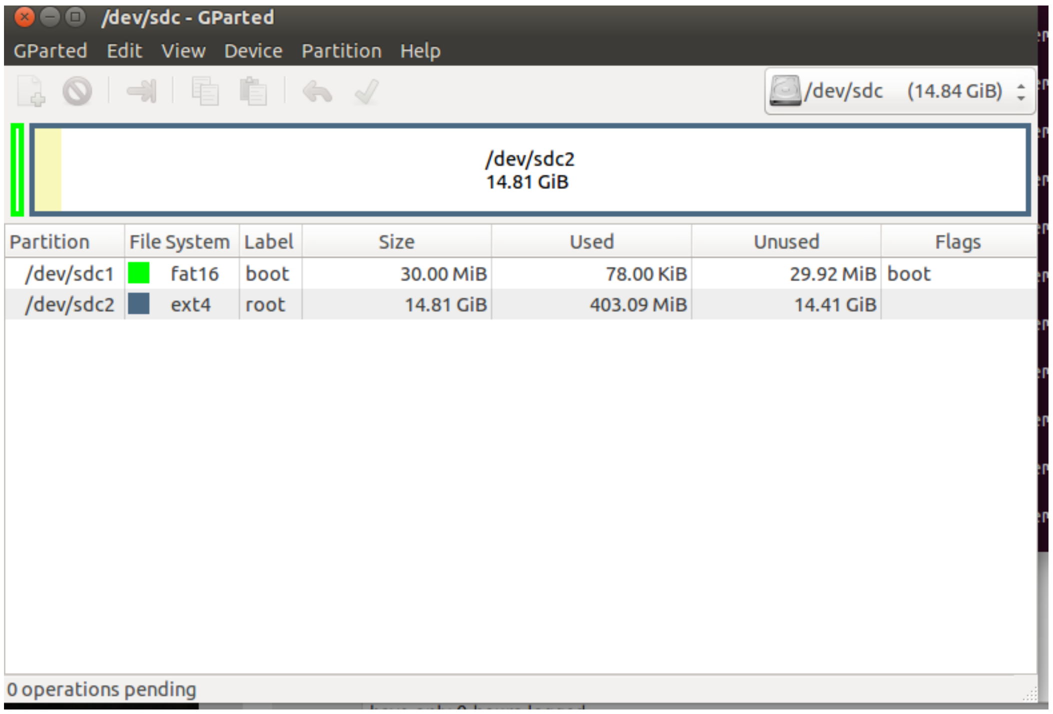
Free space preceding (MiB): 0
New size (MiB): 15163
Free space following (MiB): 0
Align to: MiB
Create as: Primary Partition
File system: ext4
Label:

Figure 9: root Partition

Click Add to create this one. No flags need be set. Note in the above image I forgot to label the partition, put in root for the label.

1.3.6 Do it

At this point you have a number of operations pending. Under device pick run operations. When done you have:



Partition	File System	Label	Size	Used	Unused	Flags
/dev/sdc1	🟩 fat16	boot	30.00 MiB	78.00 KiB	29.92 MiB	boot
/dev/sdc2	■ ext4	root	14.81 GiB	403.09 MiB	14.41 GiB	

Figure 10: Finish

You can then dismount the disk, it will be used later and filled out with results of the Yocto builds.

2 Build and deploy

2.1 Initial Build

The initial build which gets all the tools and source code can be run now since all the tools are in place. First cd to ~/yocto/pokey.

```
cd yocto/pokey
. oe-init-build-env <directory where you want to build>   <-- Sets up build directory (use a
```

Edit the config file at ~/yocto/pokey/<build name>/conf/local.conf. If build name was BB1 the file to edit would be ~/yocto/pokey/BB1/conf/local.conf.

Editing the beaglebone line:

```
# There are also the following hardware board target machines included for
# demonstration purposes:
#
```

```
MACHINE ?= "beaglebone"
#MACHINE ?= "genericx86"
#MACHINE ?= "genericx86-64"
#MACHINE ?= "mpc8315e-rdb"
#MACHINE ?= "edgerouter"
```

Now with the env set up and BeagleBone selected you can start a build

```
bitbake core-image-full-cmdline
```

This line starts the build which takes a LOOONG time. In my case with a slow internet link the initial build took about 7 hours.

2.2 Deploy

2.2.1 Introduction

Once the build is done files produced have to copied to the correct partitions of the TEST SD card. These files are:

- MLO This is the second stage boot code, called by the BBB hardware.
- U-Boot This is the third stage boot called by MLO.
- uImage This is the kernel code called by U-Boot.
- Root File system Used by the kernel and holds the Linux file system.

Also a number of other files are produced by Yocto but will not be used at this time.

2.2.2 Copying the files

When running the build we were in ~/yocto/pokey. And we ran **oe-init-build-env** with the name of our build directory (see 2.1 on the preceding page) . For example lets assume that the name was BB1. Then the results are in:

```
~/yocto/pokey/BB1/tmp/deploy/images/beaglebone
```

The files we need there are:

- MLO-beaglebone
- u-boot-beaglebone.img

- <name of build recipe>.tar.bz2. Thus if we built core-image-full-cmdline it would core-image-full-cmdline..tar.bz2.

Notice we did not use a uImage file. One is built but the one needed is actually inside the .tar.bz2 file.

Insert the TEST SD disk and ubuntu will automount it as root and boot. The actual path will be /media/<user>/root and /media/<user>boot. In my case <user> == jim. The copy commands now become:

```
copy MLO-beaglebone /media/jim/boot/MLO
copy u-boot-beaglebone.img /media/jim/boot/u-boot.img
```

The root file system is expanded into the root directory with tar

```
sudo tar -xvf core-image-full-cmdline.tar.bz2 -C /media/jim/root
```

3 Running BBB

Once you have built the image and copied to the card you need to test it. For this you need the serial connector and will hook a USB to it and run a terminal program to watch the output. You will also need a second USB to plug in to the BBB to power it.

3.1 Running with an SD card

- Attach the USB<->Serial converter to the board as show in 1.1.2 on page 2. Run a terminal program for this port at 115200 baud.
- Insert the SD card into the BBB
- Press down on the button nearest the SD card socket. Hold it down.
- Insert the other USB cable into the BBB

If all went well you should see a boot process like the Angstrom card for your new system.

3.2 Setting the board to ALWAYS boot from the SD card

It takes a bit of coordination to do the above, particularly holding the switch and inserting the USB cable. By modifying the boot partition in the MMC (on board flash) we can eliminate the button press. This will set the board to always boot from the SD. Not to worry with the Angstrom card you can easily restore the MMC boot.

- Start up the BBB with no SD card, booting from the internal MMC.
- This will bring up the boot partition on your desktop.

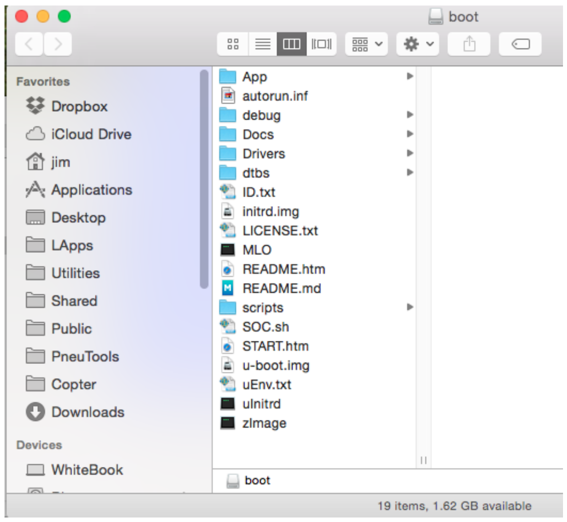


Figure 11: MMC Boot partition

- Change the name of the MLO file as shown.

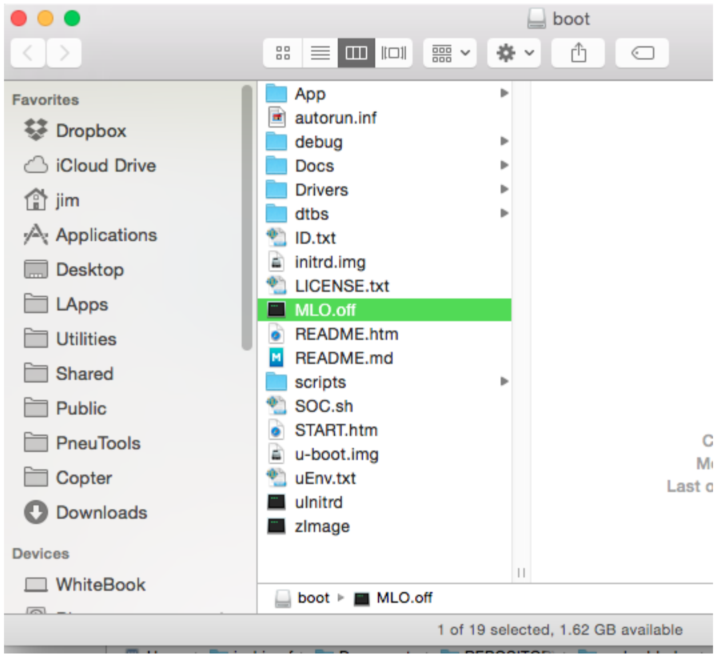


Figure 12: MMC Modified

With the name change MLO won't be found and the MMC won't boot.

3.3 Restoring MMC Boot

If you insert the Angstrom card and boot, it mounts the MMC boot partition and you can rename MLO back to normal and your card will now boot from MMC as stock.

```
root@beaglebone:~# ls /media/boot
App   ID.txt       README.htm   START.htm    dtbs
u-boot.img  zImage
Docs
LICENSE.txt  README.md   autorun.inf  initrd.img  uEnv.txt
Drivers  MLO.off      SOC.sh   debug        scripts
uInitrd
root@beaglebone:~#
```

As you can see we now can list the MMC boot partition at /media/boot. Doing the mv command will rename MLO.off back to MLO and MMC will now boot.

```
mv /media/boot/MLO.off /media/boot/MLO
```

4 Customizing Yocto

4.1 Introduction

Up to this point we have concentrated on the mechanics of the use of Yocto and the Beagle Bone board. That is using just building and using a simple example and getting it running on the BBB. The rest of this paper will concentrate on how to modify builds to customize the system you want. The main reference for this is the Yocto Mega manual http://www.yoctoproject.org/docs/1.6.3/mega-manual/mega-manual.html.

4.2 Tutorials

- Getting started with Yocto: http://video.linux.com/videos/getting-started-with-the-yo
- Yocto Project Introduction http://elinux.org/Yocto_Project_Introduction
- From Embedded Linux conference: https://wiki.yoctoproject.org/wiki/Training

There are many many more just search for them.

4.3 Creating a new layer

If we wish to add new recipes of our own it is best then to create a new recipe layer to hold them. This next section details how to add a layer called **meta-local**. At this point you should have run the very long initial build 2.1 on page 10 and thus have a working build directory. (In this case we will assume it's called BB1). Also following the example of http://stackoverflow.com/questions/18974858/how-to-write-own-package-for-recipe-in-arago-project-build which is an excellent guide to what we are about to do.

4.3.1 Create the folder for your new layer

Create the directory meta-recipe in yocto/pokey

```
cd ~
cd yocto/poky
mkdir meta-local
```

Create the subdirectories of recipes and the recipe folder

```
mkdir meta-local/recipes-lib  (following the URL above)
mkdir meta-local/recipes-IOT
mkdir meta-local/conf
```

4.3.2 Add the configuration

In the folder conf add the file layer.conf with the following text:

```
# We have a conf directory, append to BBPATH
BBPATH .= ":${LAYERDIR}"
# We have a recipes directory, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb ${LAYERDIR}/recipes-*/*/*.bbappend"
BBFILE_COLLECTIONS += "meta-local"
BBFILE_PATTERN_meta-local := "^${LAYERDIR}/"
BBFILE_PRIORITY_meta-local = "7"
```

4.3.3 Adding the layer

Edit the bblayers.conf file in your local configuration (in this case BB1/conf/bblayers.conf). To look like this:

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
```

```
LCONF_VERSION = "6"
BBPATH = "${TOPDIR}"
BBFILES ?= ""
BBLAYERS_NON_REMOVABLE ?= " \
  /home/jim/yocto/poky/meta \
  /home/jim/yocto/poky/meta-yocto \
  "
# Add my new meta-local layer
BBLAYERS ?= " \
/home/jim/yocto/poky/meta \
/home/jim/yocto/poky/meta-yocto \
/home/jim/yocto/poky/meta-yocto-bsp \
/home/jim/yocto/poky/meta-local \      <-- Added line
"
```

4.4 Creating a recipe for LIBMILL

4.4.1 Introduction

LIBMILL is a co-routine library that supports very fast context switching and Go language syntax for co-routines. It will be very useful in the BeagleBone so it was decided to support it. More information is available at https://libmill.org. It also has a gitHub site at https://github.com/sustrik/libmill.git. Before producing the recipe we have to gather the needed information.

4.4.2 Header information

This is the who,what and where of the library.

```
Site URL: https://libmill.org
Git URL: https://github.com/sustrik/libmill
Author: Martin Sustrik
License File: COPYING in repository
License Type MIT
```

4.4.3 Getting the SRCREV

This is the hash value that points to the correct branch of the git repo.

```
500 ~> git ls-remote https://github.com/sustrik/libmill
419f4eb0b9aa9298ccbf434f24464ec292469951 HEAD
bdf6badf0fe2f6fb9d0733089d1ecad14481fd75 refs/heads/gh-pages
419f4eb0b9aa9298ccbf434f24464ec292469951 refs/heads/master
       :
       :
```

We now have SRCREV as 419f4eb0b9aa9298ccbf434f24464ec292469951 for the HEAD of our repository.

4.4.4 Getting the license hash

We have a local copy of libmill and in there we run md5 on the license file COPYING.

```
500 libmill> md5 COPYING
MD5 (COPYING) = 6d82245bd22227ca739abaf3dcd3b80d
501 libmill>
```

4.4.5 The recipe

Using the information we have gathered we can build the following recipe in a folder **libmill** in **recipe-lib**.

```
# ---------------------------------------------------------
# Header Meta data
SUMMARY = "Co-routine & Network Library"
DESCRIPTION = "Easy network use and co-routine operation"
AUTHOR = "Martin Sustrik"
HOMEPAGE = "https://libmill.org"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://COPYING;md5=6d82245bd22227ca739abaf3dcd3b80d"
PR = "1.0"
# ---------------------------------------------------------
# Source data
SRCREV = "419f4eb0b9aa9298ccbf434f24464ec292469951"
SRC_URI = "git://github.com/sustrik/libmill.git;protocol=http"
inherit autotools gettext
S = "${WORKDIR}/git"
```

Notice the line **inherit autotools gettext** this is required as the project uses autotools and we must bring in the autotools class for Yocto.

4.4.6 Testing

See 2.1 on page 10 for startup information on Yocto. Running **bitbake libmill** in our directory where we did our initial build. The libmill build runs with no errors so we have successfully added libmil to our build system. If you have to try again use **bitbake -c cleanall libmill** to wipe out previous work and try again. You can also say **bitbake -c compile libmill** for example if the fetch went correctly but the compile didn't and just repeat the compile or other step for testing.

4.5 Creating a recipe for SODIUM

4.5.1 Introduction

The SODIUM library is a complete crypto library with multiple language bindings that is simple to use and quite well documented. More information can be found at https://github.com/jedisct1/libsodium.

4.5.2 Header information

This is the who,what and where of the library.

```
Site:           https://github.com/jedisct1/libsodium
GIT URL:        https://github.com/jedisct1/libsodium
Author:         Frank Denis
License:        ISC License
License File:   LICENSE
```

4.5.3 Getting the SRCREV

This is the hash value that points to the correct branch of the git repo.

```
500 ~> git ls-remote https://github.com/jedisct1/libsodium
fcf9441c7b48d62895dba869a2c9ade0628e7e9f HEAD
54e962ef949dedfae83a73dd0b8b2ce749cfa15d refs/heads/arm
1635f986384e79fbdebadb7ad710711461ae2294 refs/heads/coverity_scan
          :
          :
```

We now have SRCREV as fcf9441c7b48d62895dba869a2c9ade0628e7e9f for the HEAD of our repository.

4.5.4 Getting the license hash

We have a local copy of libmill and in there we run md5 on the license file LICENSE.

```
500  md5 LICENSE
MD5 (LICENSE) = 2278eb2755b451372dde7ffeae8cde98
```

4.5.5 The recipe

Using the information we have gathered we can build the following recipe in a folder **libsodium** in **recipe-lib**.

```
# ----------------------------------------------------
# Header Meta data
SUMMARY = "Crypto Library"
DESCRIPTION = "Peer reviewed Crypto"
AUTHOR = "Frank Denis"
HOMEPAGE = "https://github.com/jedisct1/libsodium"
LICENSE = "ISC"
LIC_FILES_CHKSUM = "file://LICENSE;md5=2278eb2755b451372dde7ffeae8cde98"
PR = "1.0"
# ----------------------------------------------------
# Source data Updated 13-Jan-2016
# My fork
SRCREV = "fcf9441c7b48d62895dba869a2c9ade0628e7e9f"
SRC_URI = "git://github.com/jedisct1/libsodium.git;protocol=https"
inherit autotools gettext
S = "${WORKDIR}/git"
```

Notice the line **inherit autotools gettext** this is required as the project uses autotools and we must bring in the autotools class for Yocto.

4.5.6 Testing

See 2.1 on page 10 for startup information on Yocto. Running **bitbake libsodium** in our directory where we did our initial build. The libmill build runs with no errors so we have successfully added libmil to our build system. If you have to try again use **bitbake -c cleanall libmill** to wipe out previous work and try again. You can also say **bitbake -c compile libsodium** for example if the fetch went correctly but the compile didn't and just repeat the compile or other step for testing.

4.5.7 Use

Compile code using Sodium with

```
cc <pgm> -l sodium
```

When using the Sodium library do set the library load path if not set already.

```
export LD_LIBRARY_PATH=/usr/local/lib
```

Also if you get errors of the form:

```
*** stack smashing detected ***: ./a.out terminated
```

add this to the compile line

```
cc test.c -lsodium -fno-stack-protector
```

4.6 Building a board image

4.6.1 Introduction

Now that we have our new recipes working it is time to put them into an image for the board. This image is going to have C development tools, lua and the libmil and sodium librarys. For this we have to modify the local.conf (BB1/conf/local.conf) file for these additions.

4.6.2 Adding C tools

We will need these if we wish to compile and link with libmill.

```
#
# Extra image configuration defaults
#
# The EXTRA_IMAGE_FEATURES variable allows extra packages to be added to the generated
# images. Some of these options are added to certain image types automatically. The
# variable can contain the following options:
```

adding these lines

```
# We default to enabling the debugging tweaks.
EXTRA_IMAGE_FEATURES = "debug-tweaks "
EXTRA_IMAGE_FEATURES = +"tools-sdk "
EXTRA_IMAGE_FEATURES = +"tools-debug "
```

so the C build tools will be added to the image.

4.6.3 QEMU Removal

For some reason it won't build with the Qemu enabled so it is all commented out

```
#
# Qemu configuration
#
# By default qemu will build with a builtin VNC server where graphical output can be
# seen. The two lines below enable the SDL backend too. This assumes there is a
# libsdl library available on your build system.
#PACKAGECONFIG_pn-qemu-native = "sdl"
#PACKAGECONFIG_pn-nativesdk-qemu = "sdl"
#ASSUME_PROVIDED += "libsdl-native"
```

4.6.4 Lua and Libmill addition

Adding these lines at the end of the conf file will cause lua the libmill library and sodium to be added.

```
CORE_IMAGE_EXTRA_INSTALL += "lua5.1"
CORE_IMAGE_EXTRA_INSTALL += "libmill"
CORE_IMAGE_EXTRA_INSTALL += "libsodium"
```

4.6.5 Building the image

At this point you can run bitbake and build the image

```
bitbake core-image-full-cmdline
```

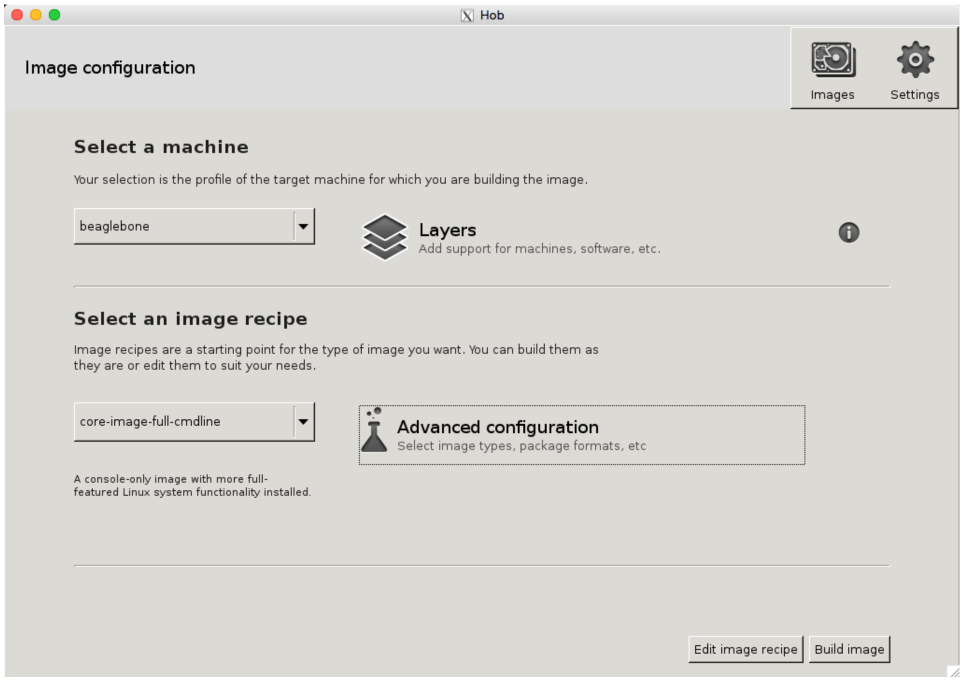
When done the image will be in BB1/tmp/deploy/images/beaglebone. The file name will be:

```
core-image-full-cmdline-beaglebone-YYYYMMDDHHMMSS.tar.bz2
```

You can then install this to an SD card as described in 2.2.2 on page 11.

4.7 Hob

Is a GUI version of bitbake that assists you in building custom systems. After you have set up the environment (2.1 on page 10) you can run to to configure and build a system. It is also useful in that picking an image give you commentary on what the image does.



Select a machine

Your selection is the profile of the target machine for which you are building the image.

beaglebone

Select an image recipe

Image recipes are a starting point for the type of image you want. You can build them as they are or edit them to suit your needs.

core-image-full-cmdline

A console-only image with more full-featured Linux system functionality installed.

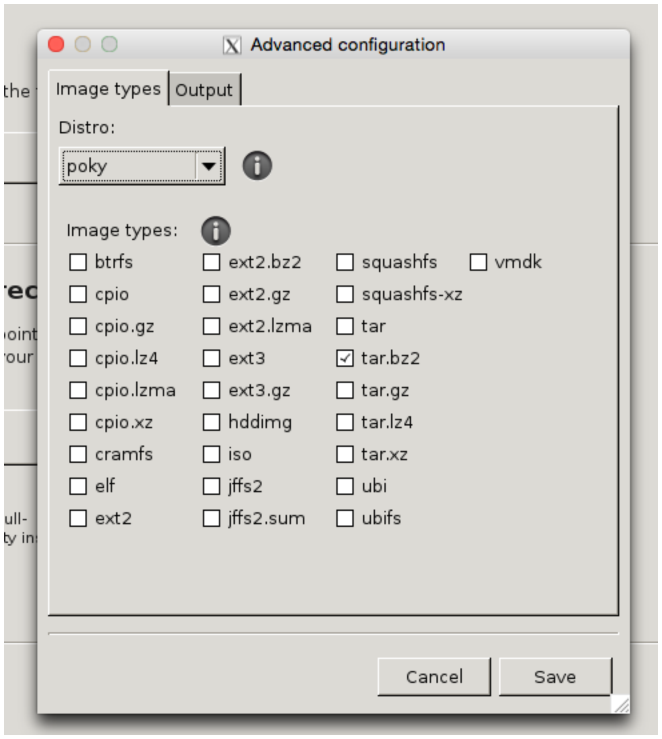
Figure 13: Hob

4.7.1 Adding new items

Once hob is running you can edit your build recipe to add or remove items with the edit script button.

4.7.2 Advanced configuration

This allows you to define the format of the output. It is useful for the BBB since we only use the tar.bz2 file to just make that one.



Distro: poky

Image types:

☐ btrfs	☐ ext2.bz2	☐ squashfs	☐ vmdk
☐ cpio	☐ ext2.gz	☐ squashfs-xz	
☐ cpio.gz	☐ ext2.lzma	☐ tar	
☐ cpio.lz4	☐ ext3	☑ tar.bz2	
☐ cpio.lzma	☐ ext3.gz	☐ tar.gz	
☐ cpio.xz	☐ hddimg	☐ tar.lz4	
☐ cramfs	☐ iso	☐ tar.xz	
☐ elf	☐ jffs2	☐ ubi	
☐ ext2	☐ jffs2.sum	☐ ubifs	

Figure 14: Setting for just tar.bz2

4.8 Layers and Recipes

If you look at the directory of yocto/poky you will find this:

```
jim@ubuntu:~/yocto/poky$ ls
BB1            LICENSE        meta-yocto                  README
BBB            meta           meta-yocto-bsp              README.hardware
bitbake        meta-selftest  oe-init-build-env           scripts
documentation  meta-skeleton  oe-init-build-env-memres
```

In this case you see BBB and BBB1 which are build directories and a number of meta-xxx files. These are layers that can be added to a build and inside each of them are recipes you can pick and choose to add. For example inside meta we have:

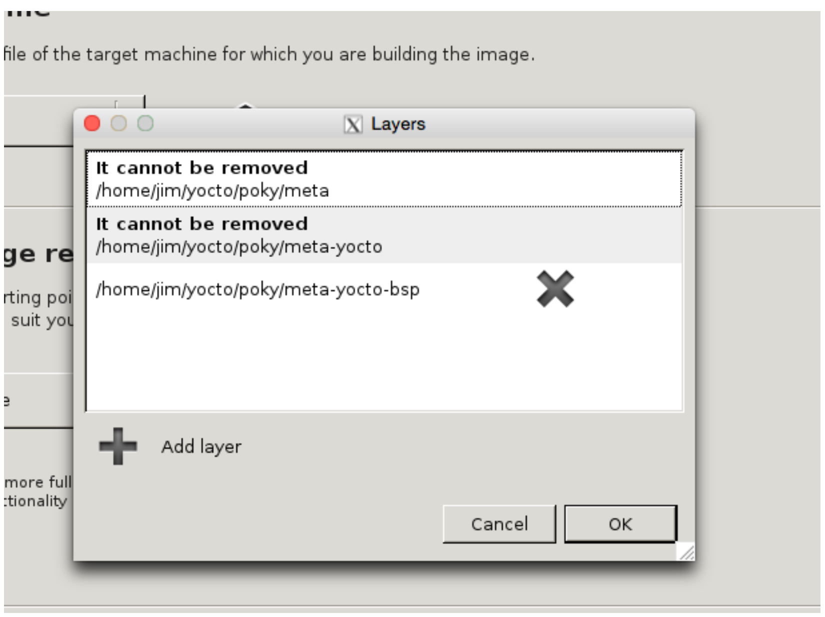
```
jim@ubuntu:~/yocto/poky$ ls meta
```

```
classes        recipes-bsp           recipes-graphics    recipes-sato
conf           recipes-connectivity  recipes-kernel      recipes-support
COPYING.GPLv2  recipes-core          recipes-lsb4        recipes.txt
COPYING.MIT    recipes-devtools      recipes-multimedia  site
files          recipes-extended      recipes-qt
lib            recipes-gnome         recipes-rt
```

Which are another group of layers that we can add to the project. And inside each of these are recipes we can choose to add.

```
jim@ubuntu:~/yocto/poky$ ls meta/recipes-extended/
at            ethtool      libaio      mdadm          quota         tzcode
augeas        findutils    libarchive  mingetty       rpcbind       tzdata
bash          foomatic     libidn      minicom        screen        unzip
bc            gamin        libtirpc    mktemp         sed           watchdog
blktool       gawk         libuser     msmtp          shadow        wget
byacc         ghostscript  lighttpd    net-tools      slang         which
bzip2         gperf        logrotate   newt           stat          xdg-util
chkconfig     grep         lsb         packagegroups  sudo          xinetd
cpio          groff        lsof        pam            sysklogd      xz
cracklib      gzip         ltp         parted         sysstat       zip
cronie        hdparm       lua         pax            tar
cups          images       mailx       perl           tcp-wrappers
cwautomacros  iptables     man         pigz           texi2html
diffutils     iputils      man-pages   procps         texinfo
ed            less         mc          psmisc         time
```

At present hob is configured to use three layers:



file of the target machine for which you are building the image.

Layers

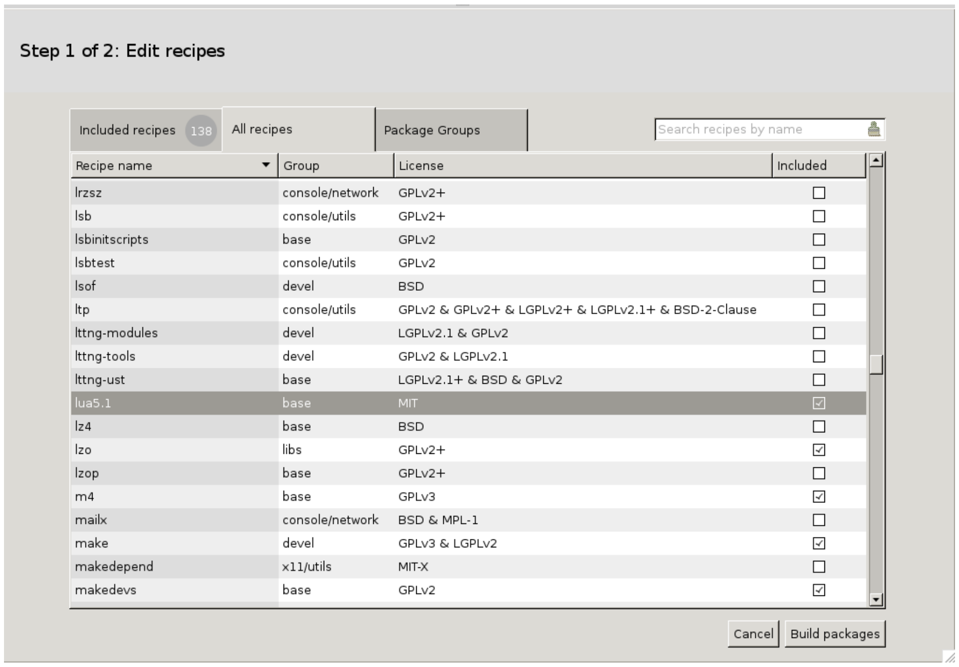
It cannot be removed
/home/jim/yocto/poky/meta

It cannot be removed
/home/jim/yocto/poky/meta-yocto

/home/jim/yocto/poky/meta-yocto-bsp

Figure 15: hob layers

Thus we can add recipes from these three layers. Pushing the Edit Image Recipe button we get a list of all that is included, all that you could add and packages that could be added. The packages are groups of recipes that support each other. In this case we want to add lua to our build.



Step 1 of 2: Edit recipes

Recipe name	Group	License	Included
lrzsz	console/network	GPLv2+	☐
lsb	console/utils	GPLv2+	☐
lsbinitscripts	base	GPLv2	☐
lsbtest	console/utils	GPLv2	☐
lsof	devel	BSD	☐
ltp	console/utils	GPLv2 & GPLv2+ & LGPLv2+ & LGPLv2.1+ & BSD-2-Clause	☐
lttng-modules	devel	LGPLv2.1 & GPLv2	☐
lttng-tools	devel	GPLv2 & LGPLv2.1	☐
lttng-ust	base	LGPLv2.1+ & BSD & GPLv2	☐
lua5.1	base	MIT	✓
lz4	base	BSD	☐
lzo	libs	GPLv2+	✓
lzop	base	GPLv2+	☐
m4	base	GPLv3	✓
mailx	console/network	BSD & MPL-1	☐
make	devel	GPLv3 & LGPLv2	✓
makedepend	x11/utils	MIT-X	☐
makedevs	base	GPLv2	✓

Figure 16: Adding lua

then you can pick Build Packages and the new image will be built.

5 Beagle Bone Use

5.1 Introduction

Now that the core-image-full-cmdline image has been built (see 4.6.5 on page 21) this section will cover various topics using this image with the board.

5.2 GPIO

5.2.1 Introduction

The GPIO pins are all the pins on the headers see [1]pages 85and 86 for the tables of the uses of the pins and their positions on the connector. The rest of section details how to access these pins as devices.

5.2.2 Creating pin devices

The set up of all these devices requires root or sudo privileges. All of the setup is done in the directory /sys/class/gpio.

- Determine which pin on the connector you wish to use say GPIO1_28.
- Find in the tables on pages 85,86 in [1].
- Look at the last column in the table and see that it says gpio1[28] also in the first column for the connector # 12 on P9.
- Calculate the pin value for setup. This is 32 * gpiomux + pin #. The gpiomux is the value after gpio[#] in the name. Thus gpio_7 would have a gpiomux = 0 and our gpio1[28] has a value of 1. Thus the # is 32 * 1 + 28 = 60.
- Now set up the I/O device by

  ```
  sudo sh -c “echo 60 > /sys/class/gpio/export”
  ```

- This will create gpio60 in the /sys/class/gpio directory.
- You now can control the device by:

  ```
  sudo sh -c "echo <in/out> > /sys/class/gpio/gpio60/direction"
  sudo sh -c "echo <0/1> > /sys/class/gpio/gpio60/value"
  ```

- The device can be removed by:

```
sudo sh -c "echo 60 > /sys/class/gpio/unexport"
```

5.3 Internet Monitor

5.3.1 Introduction

I have a very slow DSL connection (~1.5 mb/s), with DSL the speed doesn't vary by much over time but the channel capacity is used up. A more sensitive monitor of measure of how the connection is performing is a ping. Pinging a website and graphing the times is a very good realtime measure of DSL performance. I wrote a program (Pinger - See Files) that pings either Google (default) or anywhere else 1/second then outputs a stript recorder like output.

```
144.785 |                                  *
152.038 |                                  *
234.803 |                                   *
136.562 |                                 *
216.993 |                                   *
 81.579 |                               *
 84.011 |                               *
 72.387 |                               *
154.773 |                                  *
 73.424 |                               *
 79.566 |                               *
223.920 |                                   *
```

The left side is time in ms and the dot is a log scale representation of the value. After building the Yocto full-cmdline version with code development tools I decided this would be a good first program to test cc & make.

5.3.2 Development

The code was developed on OSX using XCode then converting the XCode project to a makefile. It was much easier to test and edit in XCode. Once running on OSX then copying and attempting to build there.

5.3.3 Problems

Once moved to the BBB two problems arose:

1. The text of the ping messge was slightly different between OSX (BSD) and Linux. This made a conditional compile necessry with two different sscanf() lines.

2. There was no log function in the the math library. Neither Log10 or Log seemed to be present so a simple Taylor series log function was written.

```
// ---------------------------------------------------------------------------
/*
    double mylog(double x)  - Log 10 with taylor series

    INPUT:  x   Number to for log value
    OUTPUT: log10(x)

    From: http://www.efunda.com/math/taylor_series/logarithmic.cfm # 3
*/
// ---------------------------------------------------------------------------
#define LN_TO_LOG 2.303
#define TERMS   10000
static double mylog(double x)
{
    int i;
    double sum = 0.0;
    double fraction;
    double part;
    double div = 1.0;
    fraction = (x-1)/x;
    part = fraction;
    for( i=0; i<TERMS; i++ )
    {
        sum += part/div;
        part *= fraction;
        div++;
    }
    // Convert to base 10
    sum =  sum /LN_TO_LOG;
    return sum;
}
```

5.4 Ubuntu SD Card

This section will describe how to make an SD card with Ubuntu on it. First prepare an SD card following the steps shown here 1.2 on page 5 and here 1.3 on page 6. With the newly made card mounted proceed as follows in Ubuntu running on a larger machine.

5.4.1 Get the SW

```
Boot software: wget http://s3.armhf.com/dist/bone/bone-uboot.tar.xz
Root software: wget\
     http://s3.armhf.com/dist/bone/ubuntu-trusty-14.04-rootfs-3.14.4.1-bone-armhf.com.tar.xz
```

This should download these two archives.

5.4.2 Put the SW on the SD card

The SD card should be mounted as /media/<user>/boot and /media/<user>/root. Where <user> is your login name.

```
Boot Load: tar xJvf bone-uboot.tar.xz -C /media/<user>/boot
Root Load: tar xJvf\
      ubuntu-trusty-14.04-rootfs-3.14.4.1-bone-armhf.com.tar.xz -C /media/<user>/rootfs
```

5.4.3 Next Steps

- User Name: ubuntu
- Password : ubuntu

This system has open-ssh enabled & X11 so ssh -Y -C <ip address> will allow X11 use from the client.

The package cache has been emptied before proceeding do this to get apt-get ready for use.

```
sudo apt-get update
sudo apt-get upgrade
```

The system does not have git. You can add it via **sudo apt-get install git**.

5.5 Installing opencv on Ubuntu

5.5.1 Introduction

opencv is a complete soup to nuts image processing libray with C,C++ and Python bindings. It gives you high level calls for image processing functions and allow your program to control the data flow while the processing of image data is done in the most efficient manner possible. This section is derived from http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html

and shows how to install the support libraries and build opencv on the BeagleBone. One import criteria is you must have a large enough SD card to support this. opencv takes about ~3 G of free space to build and install. After the build process and removal of the build support it will used about 0.5G.

5.5.2 Precursors

If you don't have compile tools apt-get the following:

```
sudo apt-get install build-essential
sudo apt-get install git
```

5.5.3 Build directories

Make a directory in your area for the cloned opencv. Make a second directory where you want to put the results of the build.

```
mkdir opencv-clone
mkdir opencv-build
```

5.5.4 Clone opencv

Move to the clone directory and clone the repository

```
cd opencv-clone
git clone https://github.com/Itseez/opencv.git
```

5.5.5 Build opencv

Move to the build directory and start the build. It took about 4 hours.

```
cd ~/opencv-build
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ../opencv-clone
make
```

5.5.6 Install opencv

```
cd ~/opencv-build
sudo make install
```

5.5.7 Make python linkages

```
sudo apt-get install python-opencv
```

5.6 Advertising services via avahi

5.6.1 Introduction

Boujour is an open source service advertising protocol developed by Apple. It allows a device to broadcast the services (TCP/IP) that it supports. This is useful for the BBB since it comes up via DHCP and using Bonjour we can find the TCP/IP address for ssh or other services. This is from an article http://www.fiz-ix.com/2012/12/using-avahi-in-ubuntu-to-broadcast-services-t

In our case we wish to advertise the SFTP and SSH services.

5.6.2 Install avahi

If not already installed you need to install the avahi daemon.

```
sudo apt-get install avahi-daemon
sudo update-rc.d avahi-daemon defaults
```

The second line updates the config file for it.

5.6.3 Create Config file for SSH/SFTP

You need to create the file /etc/avahi/services/ssh.service. The put this text into it.

```
<?xml version="1.0" standalone='no'?><!--*-nxml-*-->
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
   <name replace-wildcards="yes">%h</name>
   <service>
      <type>_sftp-ssh._tcp</type>
      <port>22</port>
   </service>
   <service>
      <type>_ssh._tcp</type>
      <port>22</port>
   </service>
</service-group>
```

5.6.4 Restart the daemon

The avahi deamon will re-start on boot but it you want to test it now:

```
sudo /etc/init.d/avahi-daemon restart
```

Then the services will appear in the Bonjour Browser .

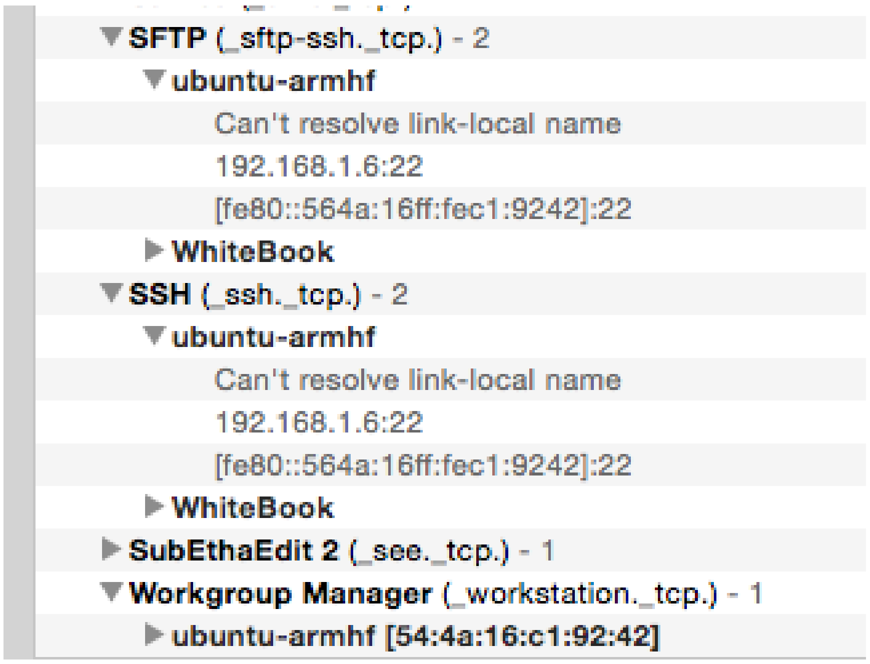

▼ SFTP (_sftp-ssh._tcp.) - 2
▼ ubuntu-armhf
Can't resolve link-local name
192.168.1.6:22
[fe80::564a:16ff:fec1:9242]:22
▶ WhiteBook
▼ SSH (_ssh._tcp.) - 2
▼ ubuntu-armhf
Can't resolve link-local name
192.168.1.6:22
[fe80::564a:16ff:fec1:9242]:22
▶ WhiteBook
▶ SubEthaEdit 2 (_see._tcp.) - 1
▼ Workgroup Manager (_workstation._tcp.) - 1
▶ ubuntu-armhf [54:4a:16:c1:92:42]

Figure 17: Advertised services in Bonjour Browser

Or via dns-sd

```
524 TEST Image1> dns-sd -B _ssh._tcp
Browsing for _ssh._tcp
DATE: ---Sat 14 Nov 2015---
19:28:34.715  ...STARTING...
Timestamp     A/R    Flags  if Domain               Service Type         Ins
19:28:34.716  Add        3   5 local.               _ssh._tcp.           Whi
19:28:34.716  Add        2   5 local.               _ssh._tcp.           ubu
525 TEST Image1> dns-sd -B _sftp-ssh._tcp
Browsing for _sftp-ssh._tcp
DATE: ---Sat 14 Nov 2015---
19:29:54.020  ...STARTING...
Timestamp     A/R    Flags  if Domain               Service Type         Ins
19:29:54.021  Add        3   5 local.               _sftp-ssh._tcp.      Whi
19:29:54.021  Add        2   5 local.               _sftp-ssh._tcp.      ubu
```

References

[1] Gregory Coley. *BeagleBone Black System Reference Manual*. BeagleBone.org, www.BeagleBone.org, revision c.1 edition, My 2014.

[2] Jon Ben-Youssef Gilad Gerum Phillipe Yaghhmour, Karim Masters. *Building Embedded Linux Systems, 2nd Edirtion*. Number ISBN: 978-0-596-52968-0. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2nd edition, August 2008.