GETTING STARTED with FreeBSD on BeagleBone BLACK

The BeagleBone Black (BBB) is a $45 PC that fits in an Altoids tin. It is built on the same Texas Instruments "Sitara" chip as the earlier BeagleBone (which had a white circuit board), but is now much cheaper and significantly more capable.

BY TIM KIENTZLE

As you can see from the BeagleBoard.org website, the potential of the BeagleBone Black (BBB) is immense. The sidebar below shows a few examples of possible applications: Best of all, the BBB can run a standard FreeBSD system with all the tools and support you've come to expect.

FreeBSD on BBB

Popular new ARM systems such as the BeagleBone and Raspberry Pi have generated a lot of developer interest in FreeBSD/ARM. In the last year, most parts of FreeBSD—boot loaders, kernel, toolchain, drivers, userland, and ports—have seen significant improvement on ARM platforms.

Right now, the development version of FreeBSD supports the BBB reasonably well:

- Serial console (requires an adapter cable similar to Adafruit #954).
- Full FreeBSD boot loader, including boot-time module loading and Forth scripting.
- Device-tree based kernel.
- Micro-SD and eMMC support.
- USB host support.
- Experimental USB client support (the BBB can act like a USB device).
- 10/100 Ethernet.
- FreeBSD/ARM now uses clang as the default compiler.
- FreeBSD/ARM now uses the EABI calling convention, which offers slightly better performance and better compatibility with other compilers; if you have binaries or libraries that were compiled before this change, you will have to recompile them.
- FreeBSD can rebuild and upgrade natively on the BBB.
- A growing number of ports build and run on the BBB.

Author's Warning: I'm writing this in September 2013 based on the current status of the FreeBSD development branch. Much of the following will have changed by the time FreeBSD10 is finally released. Ask on the FreeBSD/ARM or FreeBSD current mailing lists for more up-to-date information.

Possible Applications

- **MICRO-SERVER:** The 1GHz ARM Cortex A8 processor, 512MB of RAM, 10/100 Ethernet, and 2GB of flash is plenty to run a personal or small-office web or mail server.
- **EMBEDDED:** Embedded: The BBB includes extensive GPIO and hardware expansion support and requires only 2.5 watts for basic operation. (USB requires additional power.)
- **EDUCATION:** The low price and expandability make BBB a good choice for learning about software and hardware development.

There are a few areas that still need improvement, however.

- The FreeBSD package team has plans for a public ARM package repository, but it is not yet available.
- Video and audio drivers have yet to be written.
- Expansion capes are not supported.

Your First FreeBSD Boot

To boot FreeBSD, you first build a micro-SD card with FreeBSD installed, and then boot the BBB from the micro-SD card.

What you'll need:

- BeagleBone Black.
- 5v power supply or Mini-USB cable.
- Micro-SDHC card 4GB or larger.
- Serial cable such as Adafruit #954 or FTDI TTL-232R-3V3 (optional but highly recommended).

1. Build or Download a FreeBSD Image

Below, I'll explain how you can build your own FreeBSD image. To get started, you can download an image from the FreeBSD.org website:

ftp://ftp.freebsd.org/pub/FreeBSD/snapshots/
http://ftp.freebsd.org/pub/FreeBSD/snapshots/

Caveat: "Snapshot" images are built from whatever source happened to be current in the FreeBSD development branch that day. Stable images will be released as soon as FreeBSD 10 is finalized, which is expected to occur before the end of 2013. Downloaded images are usually compresssed; you'll need to uncompress yours before you can copy it onto a micro-SD card.

2. Copy onto a Micro-SD Card

The 'dd' utility is used for raw data copies such as initializing a disk from a raw image. You first need to connect the SD card to your computer (likely using some sort of adapter) and identify the correct device name.

The easiest way to do this is to

```
$ ls -l /dev
```

before connecting the SD card.

Then do

```
$ ls -l /dev
```

again after you've connected it; the new entry should be obvious.

If the SD card is already formatted, you'll see several entries appear for each partition on the card. Since we want to overwrite the whole card, you need to identify the base device, which is generally a couple of letters followed by a single digit (e.g., "da7", "mmc4", or "sdhci0").

For this task, you want to provide three arguments to the 'dd' command: the input file (if), the output device (of), and the block size to use when copying (bs). You don't have to specify a block size, but the default setting results in very slow operation.

For example, if your micro-SD is connected as 'da7', then the full command will look like this:

```
$ dd if=FreeBSD-BeagleBone.img
of=/dev/da7 bs=8m
```

Depending on how your system security is set up, you will probably have to run this command as root using 'sudo' or similar.

Once the micro-SD card is imaged, you can insert it into the BBB.

3a. (Optional but Recomended) Attach Serial Cable

Once you have the SD card built, you're ready to hook up the BeagleBone Black and boot FreeBSD. Since FreeBSD doesn't yet support the HDMI output on the BBB, you should consider using a serial cable so you can see what's going on. Without a serial cable, you can wait until it boots and try to connect over SSH, but it's much harder to diagnose if anything goes wrong.

The BBB has a low-voltage serial interface that requires a special adapter cable. Make certain you are using a 3.3v adapter, since similar cables come in 5v and 1.8v versions that will not work with BBB (Figure 1).

3b. Open a Terminal Window

The serial adapter is powered by USB from the host system, so it starts working as soon as it is

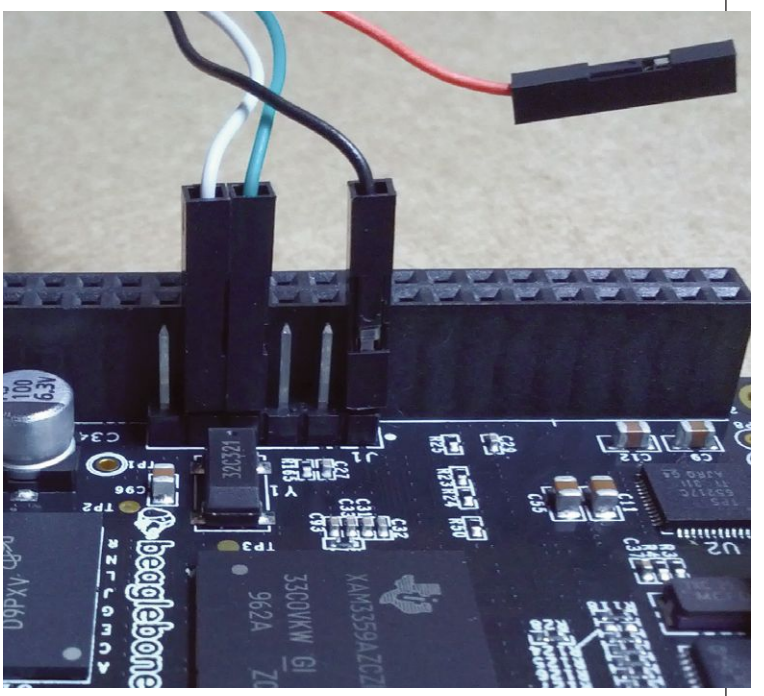

Figure 1. Adafruit #954 serial cable connected to BBB.

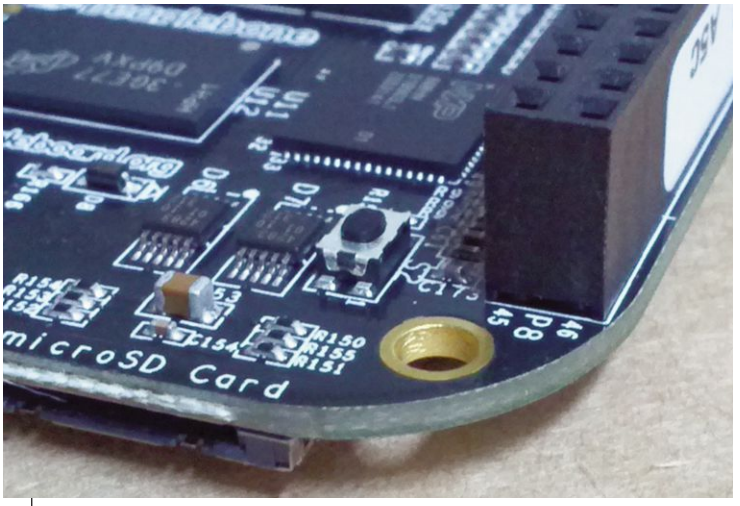

Figure 2. The boot switch is just above the micro-SD slot.

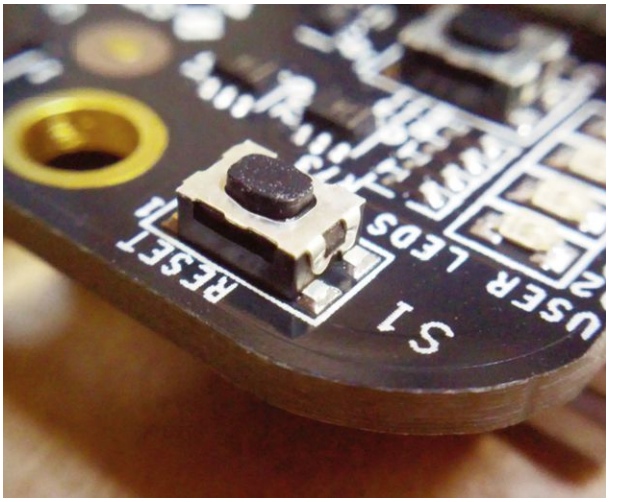

Figure 3. The reset switch is in the corner of the board at the Ethernet adapter end.

plugged into the USB, even before the BBB has power.

To use it from FreeBSD, use the 'cu' utility, specifying the line speed of 115200 baud and the appropriate "tty" device:

```
$ sudo cu -s 115200 -l /dev/ttyU0
```

Of course, you won't see anything until you actually apply power.

4. Hold the Boot Switch and Apply Power

At this point, you should NOT have any power connected to your BBB. If you've already connected a 5v power supply or a mini-USB cable, then unplug it and read the following carefully.

(The detailed logic for when the BBB boots from eMMC or micro-SD is a little complicated. I've been confused many times when the BBB booted from the wrong source.)

The "boot switch" determines whether the BBB boots from eMMC (the default) or from micro-SD (Figure 2).

To boot from micro-SD reliably, you must:

* Hold down the boot switch
* Apply power
* Count to 3
* Release the boot switch

The BBB power chip remembers the boot switch status, so it will continue to boot and reboot from micro-SD until you disconnect the power supply entirely.

Hint: If you need to reboot, leave the power connected and tap the reset switch (Figure 3), which will reboot from the same source.

Hint: If you get random shutdowns and are powering with a mini-USB cable, try getting a separate 5v power supply. The BBB power requirements are just at the edge of what standard USB ports will provide.

Hint: If you see the four LEDs start flashing rapidly, you've booted the Linux image from eMMC. Remove power, hold the boot switch, and try again.

What You Should See When You Boot

If you're familiar with how FreeBSD boots on i386 or amd64 PCs, then the BBB boot process will look very familiar, although there are a couple of differences. Most obviously, the initial boot stages are handled by "U-Boot", a GPL boot loader project that supports a wide variety of hardware.

1. MLO/SPL: U-Boot First Stage

When the TI Sitara chip first initializes, it does not have access to the main RAM. As a result, the very first boot stage must fit into 128k of on-chip memory.

```
U-Boot SPL 2013.04
(Aug 03 2013 - 21:27:30)
OMAP SD/MMC: 0
reading bb-uboot.img
reading bb-uboot.img
```

U-Boot provides a small program called SPL which the TI Sitara chip loads from a file called "MLO". This program is just enough to initialize the DRAM chip and load the main U-Boot program from the micro-SD card.

2. U-Boot Main Loader

U-Boot is a GPL-licensed boot loader that supports a wide variety of hardware. Although originally developed for Linux, U-Boot's robust hardware support, scriptability, and active community make it a good choice for booting FreeBSD as well.

U-Boot starts by initializing the USB, network, and MMC/SD interfaces.

```
U-Boot 2013.04 (Aug 03 2013 -
21:27:30)
  ... other messages ...
reading bb-uEnv.txt
reading bbubldr
240468 bytes read in 33 ms (6.9 MiB/s)
reading bboneblk.dtb
14210 bytes read in 7 ms (1.9 MiB/s)
Booting from mmc ...
## Starting application at 0x88000054
...
```

Once it has the MMC/SD initialized, it reads three files into memory.

* bb-uEnv.txt is empty by default, but you can edit this to redefine the U-Boot startup functions.

* bbubldr is the FreeBSD boot loader that will be run next.

* bboneblk.dtb is the DTB file described below.

3. About the DTB File

Operating systems for newer embedded processors are increasingly using a "device tree" file—sometimes called a "flattened device tree" (fdt)—to initialize the kernel. This file lists all the peripherals and helps the kernel decide which drivers to enable. Device trees are compiled: The source version is called DTS and the binary compiled version is called a DTB file.

The U-Boot initialization checks which hardware you are currently running and then loads the appropriate DTB file into memory. This data is not directly used by U-Boot or by ubldr, but is eventually passed to the FreeBSD kernel. The key advantage of this arrangement: The exact same kernel can run on both BeagleBone and BeagleBone Black since key configuration such as the amount of RAM and number of drives is provided by the DTB.

Eventually, the FreeBSD/ARM developers hope to have a single GENERIC kernel that boots on a number of boards. This requires more work on the kernel to ensure that the various board support routines can coexist. It also requires more work on the boot loader side to ensure that all of the various loaders correctly provide a DTB file to the kernel.

4. FreeBSD Ubldr

U-Boot knows a lot about the BBB hardware and how to initialize it, but does not know anything about the FreeBSD kernel and modules.

So the BBB uses U-Boot to load "ubldr". This is essentially the same as "BTX loader" used to boot FreeBSD on i386/amd64, but with a few changes so that it works with U-Boot instead of the PC BIOS (hence the name "ubldr" for "U-Boot compatible LoaDeR").

```
Consoles: U-Boot console
Compatible API signature found
@8f246240
Card did not respond to voltage
select!
Number of U-Boot devices: 2
FreeBSD/armv6 U-Boot loader, Revision
1.2
(root@fci386.localdomain, Fri Aug 16
12:59:51 PDT 2013)
DRAM:  256MB
Device: disk
Loading /boot/defaults/loader.conf
/boot/kernel/kernel
data=0x449864+0x17d3c8
syms=[0x4+0x82890+0x4+0x4ec85]
Hit [Enter] to boot immediately, or
any other key for command prompt.
Booting [/boot/kernel/kernel]...
Using DTB provided by U-Boot.
Kernel entry at 0x80200100...
Kernel args: (null)
```

5. Load loader.rc, loader.conf

Ubldr pulls in a lot of standard FreeBSD configuration. In particular, it reads loader.conf and possibly loader.rc. These can be used to load kernel modules into memory so they are available when the kernel first boots.

6. Load FreeBSD Kernel

Ubldr can now load the FreeBSD kernel proper.

7. Start FreeBSD Kernel

Once everything is ready, ubldr actually starts the FreeBSD kernel. The last lines printed by ubldr indicate how it is going to launch the kernel:

```
Booting [/boot/kernel/kernel]...
Using DTB provided by U-Boot.
Kernel entry at 0x80200100...
Kernel args: (null)
```

8. Initialize FreeBSD Kernel

Unlike ubldr, which relies heavily on U-Boot, the FreeBSD kernel runs completely on its own.

So it must first set up its own memory management and console handling. Once that is done, the kernel can show its first message:

```
KDB: debugger backends: ddb
KDB: current backend: ddb
Copyright (c) 1992-2013 The FreeBSD
Project.
Copyright (c) 1979, 1980, 1983, 1986,
```

```
1988, 1989, 1991, 1992, 1993, 1994
      The Regents of the University
of California. All rights reserved.
FreeBSD is a registered trademark of
The FreeBSD Foundation.
FreeBSD 10.0-CURRENT #0 r254265: Fri
Aug 16 12:58:43 PDT 2013
root@fci386.localdomain:/usr/..../src/
sys/BEAGLEBONE arm
```

The kernel then proceeds to use the device tree data to identify each system that needs to be initialized.

9. Start FreeBSD userland

After the FreeBSD kernel has finished initializing everything, it mounts the root filesystem so that it can load the first programs from the SD filesystem.

Here are the last messages printed by the kernel:

```
Trying to mount root from
ufs:/dev/mmcsd0s2a [rw,noatime]...
warning: no time-of-day clock regis-
tered, system time will not be set
accurately
```

(In particular, the warning here is expected, since the BBB does not have a battery-backed RTC.)

If you've used FreeBSD or Linux or any similar system before, the remaining boot steps should be quite familiar: The rc system runs a bunch of scripts to set up various standard systems, including network services such as SSHd and NTPd. The very first time you boot, this can take a little while, since some of these services need to set up their initial configurations. Most obviously, the SSHd service needs to create encryption keys for this particular machine.

Finally, the system is ready to accept logins.

```
Wed Sep  4 00:46:40 UTC 2013
FreeBSD/arm (beaglebone) (ttyu0)
login:
```

Most BBB images are set up to automatically configure the Ethernet port and start sshd.

So you should be able to connect remotely using SSH at this point as well.

Using FreeBSD on the BeagleBone Black

The BBB runs a completely standard FreeBSD system, so if you're comfortable with FreeBSD on i386 or amd64, then you should feel right at home.

Here are a few notes to help get you started:

Ethernet: The network interface is "cpsw0". You can configure it with the ifconfig command or edit /etc/rc.conf to set it up on every boot. Most FreeBSD images should have DHCP enabled by default.

Time: Since the BBB does not have a battery-backed clock, you'll need to either set the time manually on boot-up or use NTP to set the time from the network.

Disk: The external micro-SD interface is called "mmcsd0". The standard FreeBSD images for BBB are formatted with two partitions:

* mmcsd0s1 is the FAT slice with U-Boot and other boot files
* mmcsd0s2 is the slice used by FreeBSD

The root partition on mmcsd0s2a is generally formatted with Soft Updates + Journaling (SU+J). SU+J allows the system to reboot quickly when power is removed and reapplied.

eMMC: The 2GB built-in eMMC chip is available as "mmcsd1". By supporting 8-bit transfers, it is significantly faster than the micro-SD interface. The BBB ships with a Linux distribution installed on the eMMC, but you can easily reformat this and use it as an extra drive for FreeBSD. Soon, we expect to be able to install FreeBSD and boot it directly from eMMC.

SU+J: The BBB doesn't have an Off button; you usually just remove power. This does lead to data loss if you have software running when you disconnect power. Using "UFS Soft Updates with Journaling" (UFS SU+J) does not prevent data from being lost, but does seem to do a good job of avoiding fatal filesystem corruption.

Swap: Although 512MB RAM is sufficient for many purposes, you will probably want to enable some swap. For a number of reasons, people are generally using a swap file on the root partition rather than a separate swap partition.

You can use "swapctl -l" to find out if the image you are using already has swap configured. If not, it's easy to add a swap file:

1) Create the file: dd if=/dev/zero of=/usr/swap0 bs=1m count=768

2) Add the following line to /etc/fstab and reboot:

```
md none swap sw,file=/usr/swap0 0 0
```

Ports: If you have network access, then installing a ports tree is quite simple:

```
$ portsnap fetch
$ portsnap extract
```

You can then build and install ports as usual. For example, to install the Apache web server:

```
$ cd /usr/ports/www/apache24
$ make
$ make install
```

Packages: The FreeBSD package team does plan to provide ARM packages compatible with the new package-management tool 'pkg'. As of September 2013 this hasn't yet been implemented.

A number of individuals have had good success using Poudriere to automatically build their own package sets.

USB: USB generally works well on BBB. USB drives, USB network adapters, and printers have all been used successfully. There is one caveat, though: You should not plug any USB peripherals into the BBB unless the BBB is connected to a separate power supply. If you are powering the BBB from a mini-USB cable and try to connect any USB device, the BBB will most likely shut off.

Updating FreeBSD

Once you have FreeBSD up and running, you can download the FreeBSD source code and rebuild directly on the BBB.

Caveats:

* A full system rebuild on the BBB can take as much as two days, depending on a number of factors.

* A full source checkout is over 2G, so won't fit on the eMMC.

* FreeBSD-CURRENT (also called the 'head' branch) is the current development branch; it has the newest features and the newest bugs.

You can use the 'svnlite' command (which is a standard part of FreeBSD now) to check out the source code from the FreeBSD project's Subversion repository:

$ svnlite co http://svn.freebsd.org/base/head /usr/src

$ cd /usr/src

Read /usr/src/UPDATING, especially the summary information near the end that outlines common upgrade scenarios. Generally, a full upgrade from source looks like the following:

```
$ cd /usr/src
$ make buildworld
$ make kernel
<reboot>
$ cd /usr/src
$ mergemaster -p
$ make installworld
$ mergemaster
<reboot>
```

The UPDATING file also explains how to do partial updates, kernel-only updates, and some techniques for doing partial upgrades.

Building Your Own FreeBSD Image

If you are comfortable with the process for building and upgrading FreeBSD from source code, you can use the Crochet tool to build a custom BBB image on a fast i386 or amd64 machine.

In particular, this makes it easy to track the most recent changes to FreeBSD as the support for BBB continues to improve.

Detailed instructions are at:
https://github.com/kientzle/crochet-freebsd;
the following is a quick summary:

1) Get Crochet. You'll need the devel/git package installed, and then you can get a copy of the Crochet scripts:

$ git clone https://github.com/kientzle/crochet-freebsd

To update, use the "git pull" command from inside the source directory.

2) Create a configuration file beagleblack.sh with the following contents:

```
board_setup BeagleBone
option ImageSize 3900mb
option UsrSrc
option UsrPorts
FREEBSD_SRC=${TOPDIR}/src
```

The 'option' lines here preinstall a full FreeBSD source tree in /usr/src and a full ports tree in /usr/ports. Omitting those lines will result in a smaller image.

3) Build the image:

```
$ sudo ./crochet.sh -c beagleblack.sh
```

The script first checks whether you have all the necessary source code and tools. If any are missing, it will print instructions for obtaining them. Once it has all the pieces, a fast PC can compile a complete FreeBSD system and assemble the image in about an hour. ●

Tim Kientzle has been a FreeBSD committer for 10 years and a FreeBSD user for much longer than that. Most recently, he's been working on image-building tools and boot support for BeagleBone and Raspberry Pi.