A HANDY U-BOOT TRICK

Deploy and test new builds quickly with no more than rebooting the board.

BHARATH BHUSHAN LOHRAY

Embedded developers working on kernels or bare-metal programs often go through several development cycles. Each time the developer modifies the code, the code has to be compiled, the ELF (Executable and Linkable Format)/kernel image has to be copied onto the SD card, and the card then has to be transferred from the PC to the development board and rebooted. In my experience as a developer, I found the last two steps to be a major bottleneck. Even copying files to the fastest SD cards is slower than copying files between hard drives and sometimes between computers across the network.

Moreover, by frequently inserting and removing the SD card from the slot, one incurs the risk of damaging the fragile connectors on the development boards. Believe me! I lost a BeagleBoard by accidentally applying too much force while holding the board and pulling out the SD card. The pressure caused the I^2C bus to fail. Because the power management chip was controlled by I^2C, nothing other than the serial terminal worked after that. Setting aside the cost of the board, a board failure at a critical time during a project is catastrophic if you do not have a backup board.

After losing the BeagleBoard, I hit upon the idea to load my bare-metal code over the LAN via bootp and TFTP and leave the board untouched. This not only reduced the risk of mechanically damaging my board, but it also improved on my turn-around times. I no longer needed to copy files to the SD card and move it around.

In this article, I present a brief introduction to U-Boot and then describe the necessary configurations to set up a development environment using DHCP and TFTP. The setup I present here will let you deploy and test new builds quickly with no more than rebooting the board. I use the BeagleBone Black (**http://beagleboard.org/Products/BeagleBone%20Black**) as the target platform and Ubuntu as the development platform for my examples in this article. You may, however, use the methods presented here to work with any board that uses U-Boot or Barebox as its stage-2 bootloader.

U-Boot

U-Boot is a popular bootloader used by many development platforms. It supports multiple architectures including ARM, MIPS, AVR32, Nios, Microblaze, 68K and x86. U-Boot

U-Boot is a pretty advanced bootloader that is capable of loading the kernel and ramdisk image from the NAND, SD card, USB drive and even the Ethernet via bootp, DHCP and TFTP.

Listing 1. The Serial Console Output from the Stage-1 Bootloader

```
U-Boot SPL 2013.04-rc1-14237-g90639fe-dirty (Apr 13 2013 - 13:57:11)
musb-hdrc: ConfigData=0xde (UTMI-8, dyn FIFOs, HB-ISO Rx,
 ➥HB-ISO Tx, SoftConn)
musb-hdrc: MHDRC RTL version 2.0
musb-hdrc: setup fifo_mode 4
musb-hdrc: 28/31 max ep, 16384/16384 memory
USB Peripheral mode controller at 47401000 using PIO, IRQ 0
musb-hdrc: ConfigData=0xde (UTMI-8, dyn FIFOs, HB-ISO Rx,
 ➥HB-ISO Tx, SoftConn)
musb-hdrc: MHDRC RTL version 2.0
musb-hdrc: setup fifo_mode 4
musb-hdrc: 28/31 max ep, 16384/16384 memory
USB Host mode controller at 47401800 using PIO, IRQ 0
OMAP SD/MMC: 0
mmc_send_cmd : timeout: No status update
reading u-boot.img
reading u-boot.img
```

has support for several filesystems as well, including FAT32, ext2, ext3, ext4 and Cramfs built in to it. It also has a shell where it interactively can take input from users, and it supports scripting. It is distributed under the GPLv2 license. U-Boot is a stage-2 bootloader.

The U-Boot project also includes the x-loader. The x-loader is a small stage-1 bootloader for ARM. Most modern chips have the ability to read a FAT32 filesystem built in to the ROM. The x-loader loads the U-Boot into memory and transfers control to it. U-Boot is a pretty advanced

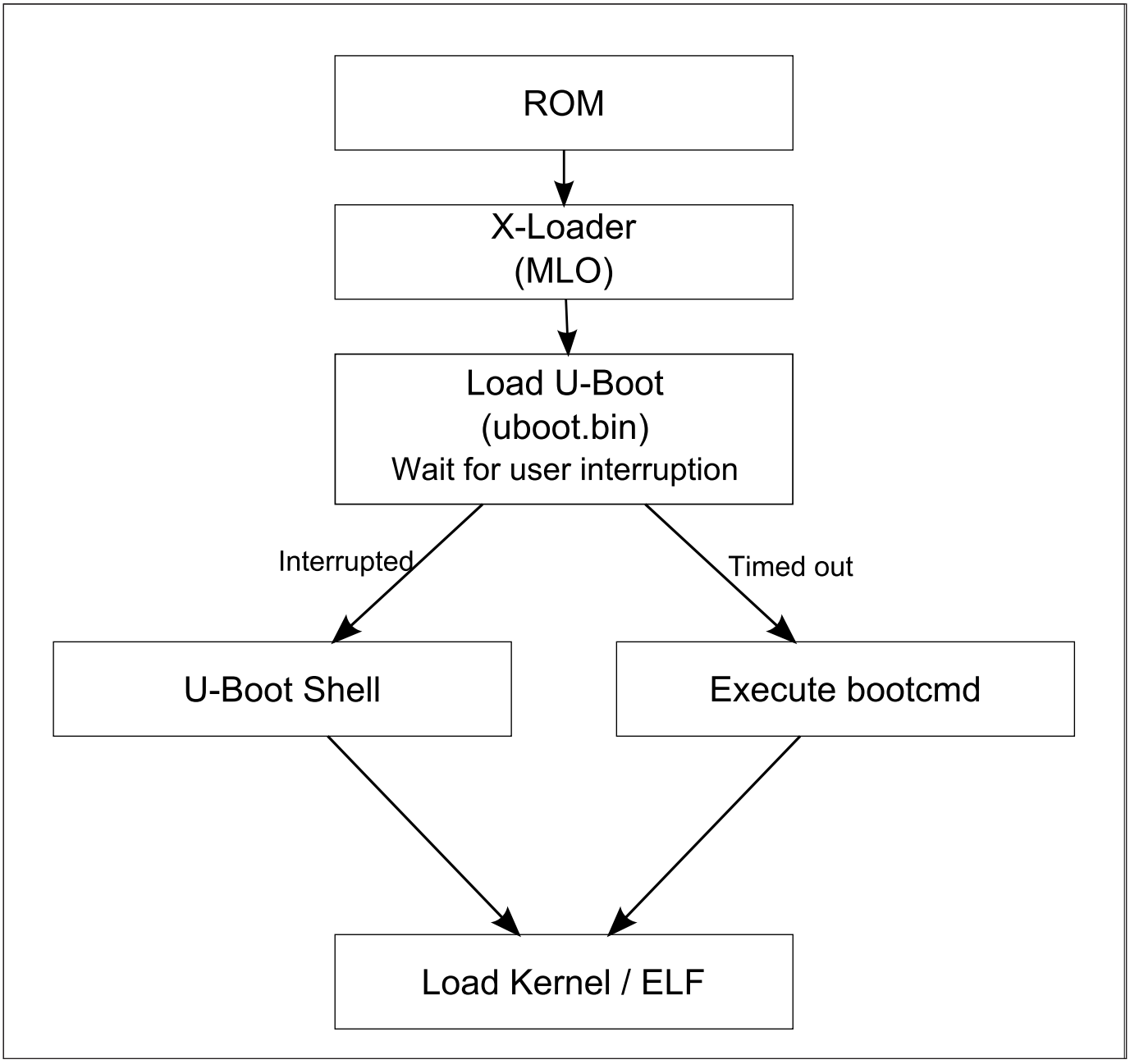

Figure 1. Boot Sequence

bootloader that is capable of loading the kernel and ramdisk image from the NAND, SD card, USB drive and even the Ethernet via bootp, DHCP and TFTP.

Figure 1 shows the default boot sequence of the BeagleBone Black. This sequence is more or less applicable to most embedded systems. The x-loader and U-Boot executables

Listing 2. The Serial Console Output from the Stage-2 Bootloader

```
U-Boot 2013.04-rc1-14237-g90639fe-dirty (Apr 13 2013 - 13:57:11)

I2C:   ready
DRAM:  512 MiB
WARNING: Caches not enabled
NAND:  No NAND device found!!!
0 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - readenv() failed, using default environment

musb-hdrc: ConfigData=0xde (UTMI-8, dyn FIFOs, HB-ISO Rx,
 ➥HB-ISO Tx, SoftConn)
musb-hdrc: MHDRC RTL version 2.0
musb-hdrc: setup fifo_mode 4
musb-hdrc: 28/31 max ep, 16384/16384 memory
USB Peripheral mode controller at 47401000 using PIO, IRQ 0
musb-hdrc: ConfigData=0xde (UTMI-8, dyn FIFOs, HB-ISO Rx,
 ➥HB-ISO Tx, SoftConn)
musb-hdrc: MHDRC RTL version 2.0
musb-hdrc: setup fifo_mode 4
musb-hdrc: 28/31 max ep, 16384/16384 memory
USB Host mode controller at 47401800 using PIO, IRQ 0
Net:   <ethaddr> not set. Validating first E-fuse MAC
cpsw, usb_ether
Hit any key to stop autoboot:  0
```

are stored in the files called MLO and uboot.img, respectively. These files are stored in a FAT32 partition. The serial port outputs of the BeagleBone are shown in Listings 1–3. The x-loader is responsible for the output shown in Listing 1. Once the execution is handed over to U-Boot, it offers you

Listing 3. The Serial Console Output from the Stage-2 Bootloader and Kernel

```
gpio: pin 53 (gpio 53) value is 1

Card did not respond to voltage select!

.

.

.

gpio: pin 54 (gpio 54) value is 1

SD/MMC found on device 1

reading uEnv.txt

58 bytes read in 4 ms (13.7 KiB/s)

Loaded environment from uEnv.txt

Importing environment from mmc ...

Running uenvcmd ...

Booting the bone from emmc...

gpio: pin 55 (gpio 55) value is 1

4215264 bytes read in 778 ms (5.2 MiB/s)

gpio: pin 56 (gpio 56) value is 1

22780 bytes read in 40 ms (555.7 KiB/s)

Booting from mmc ...

## Booting kernel from Legacy Image at 80007fc0 ...

   Image Name:   Angstrom/3.8.6/beaglebone

   Image Type:   ARM Linux Kernel Image (uncompressed)

   Data Size:    4215200 Bytes = 4 MiB

   Load Address: 80008000

   Entry Point:  80008000

   Verifying Checksum ... OK

## Flattened Device Tree blob at 80f80000

   Booting using the fdt blob at 0x80f80000

   XIP Kernel Image ... OK

OK

   Using Device Tree in place at 80f80000, end 80f888fb


Starting kernel ...


Uncompressing Linux... done, booting the kernel.

[    0.106033] pinctrl-single 44e10800.pinmux: prop pinctrl-0

 ➥index 0 invalid phandle

.

.

.

[    9.638448] net eth0: phy 4a101000.mdio:01 not found on slave 1


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

The Angstrom Distribution beaglebone ttyO0


Angstrom v2012.12 - Kernel 3.8.6


beaglebone login:
```

a few seconds to interrupt the boot sequence, as shown in Listing 2. If you choose not to interrupt, U-Boot executes an environment variable called `bootcmd`. `bootcmd` holds the search sequence for a file called `uImage`. This is the kernel image. The kernel image is loaded into the

Listing 4. Well Formatted Content of the Variable bootcmd

```
01 gpio set 53;
02 i2c mw 0x24 1 0x3e;
03 run findfdt;
04 mmc dev 0;
05 if mmc rescan ;
06 then
07     echo micro SD card found;
08     setenv mmcdev 0;
09 else
10     echo No micro SD card found, setting mmcdev to 1;
11     setenv mmcdev 1;
12 fi;
13 setenv bootpart ${mmcdev}:2;
14 mmc dev ${mmcdev};
15 if mmc rescan;
16 then
17     gpio set 54;
18     echo SD/MMC found on device ${mmcdev};
19     if run loadbootenv;
20     then
21         echo Loaded environment from ${bootenv};
22         run importbootenv;
23     fi;
24     if test -n $uenvcmd;
25     then
26         echo Running uenvcmd ...;
27         run uenvcmd;
28     fi;
29     gpio set 55;
30     if run loaduimage;
31     then
32         gpio set 56;
33         run loadfdt;
34         run mmcboot;
35     fi;
36 fi;
```

memory, and the execution finally is transferred to the kernel, as shown in Listing 3.

The search sequence defined in the `bootcmd` variable and the filename (uImage) are hard-coded in the U-Boot source code (see Listing 9). Listing 4 shows the formatted content of the environment variable `bootcmd`. The interesting parts of `bootcmd` are lines 19–28. This part of the script checks for the existence of a file called uEnv.txt. If the file is found, the file is loaded into the memory (line 19). Then, it is imported to the environment ready to be read or executed (line 22). After this, the script checks to see if the variable `uenvcmd` is defined (line 24). If it is defined, the script in the variable is executed. The uEnv.txt file is a method for users to insert scripts into the environment. Here, we'll use this to override the default search sequence and load the kernel image or an ELF file from the TFTP server.

For better insight into the workings of U-Boot, I recommend interrupting the execution and dropping to the U-Boot shell. At the shell, you can see a list of supported commands by typing `help` or `?`. You can list all defined environment variables with the `env print` command. These environment variables are a powerful tool for scripting. To resume the boot sequence, you either can issue the `boot` command or `run bootcmd`. A good way to understand what the `bootcmd` is doing is to execute each command one at a time from the U-Boot shell and see its effect. You may replace the `if...then...else...fi` blocks by executing the conditional statement without the `if` part and checking its output by typing `echo $?`.

The uEnv.txt file is a method for users to insert scripts into the environment.

DHCP

The DHCP (Dynamic Host Configuration Protocol) is a protocol to provide hosts with the necessary information to access the network on demand. This includes the IP address for the host, the DNS servers, the gateway server, the time servers, the TFTP server and so on. The DHCP server also can provide the name of the file containing the kernel

image that the host must get from the TFTP server to continue booting. The DHCP server can be set up to provide a configuration either for the entire network or on a per-host basis. Configuring the filename (Listing 5) for the entire network is not a good idea, as one kernel image or ELF file will execute only on the architecture for which it was built. For instance, the vmlinuz image built for an x86_64 will not work on a system with an ARM-based processor.

The Ubuntu apt repository offers two DHCP servers: `isc-dhcp-server` and `dhcpcd`. I prefer to use `isc-dhcp-server`.

The `isc-dhcpd-server` from the Ubuntu repository is pretty advanced and implements all the necessary features. I recommend using Webmin to configure it. Webmin is a Web-based configuration tool that supports configuring several Linux-based services and dæmons. I recommend installing Webmin from the apt repository. See the Webmin documentation for instructions for adding the Webmin apt repository to Ubuntu (**http://www.webmin.com/deb.html**).

Once you have your DHCP server

Listing 5. The Host Configuration Section for a DHCP Server

```
subnet 192.168.0.0 netmask 255.255.0.0 {
    next-server 192.168.146.1;
    option domain-name-servers 192.168.146.1;
    option routers 192.168.146.1;
    range 192.168.145.1 192.168.145.254;

        # The BeagleBone Black 1
    host BBB-1 {
        next-server 192.168.146.1;
        filename "/BI/uImage";
        hardware ethernet C8:A0:30:B0:88:EB;
        fixed-address 192.168.146.4;
    }
}
```

IMPORTANT NOTE:

Be extremely careful while using the DHCP server. A network must not have more than a single DHCP server. A second DHCP server will cause serious problems on the network. Other users will lose network access. If you are on a corporate or a university network, you will generate a high-priority incident inviting the IT department to come looking for you.

installed, you need to configure a subnet and select a pool of IP addresses to be dished out to hosts on the network on request. After this, add the lines corresponding to the host from Listing 5 into your /etc/dhcp/dhcpcd.conf file, or do the equivalent from Webmin's intuitive interface. In Listing 5, `C8:A0:30:B0:88:EB` corresponds to the BeagleBone's Ethernet address. The `next-server` is the address of the TFTP server from which to fetch the kernel image of ELF. The `/BI/uImage` filename is the name of the kernel image. Rename the image to whatever you use.

TFTP

TFTP (Trivial File Transfer Protocol) is a lightweight file-transfer protocol. It does not support authentication methods. Anyone can connect and download any file by name from the server or upload any file to the server. You can, however, protect your server to some extent by setting firewall rules to deny IP addresses out of a particular range. You also can make the TFTP home directory read-only to the world. This should prevent any malicious uploads to the server. The Ubuntu apt repository has two different TFTP servers: `atftp` and `tftp-hpa`. I recommend `tftp-hpa`, as development of `atftp` has seized since 2004.

tftpd-hpa is more or less ready to run just after installation. The default file store is usually /var/lib/tftpboot/, and the configuration files for tftp-can may be found in /etc/default/tftpd-hpa. You can change the location of the default file store to any other location of your choice by changing the `TFTP_DIRECTORY` option. The TFTP installation creates a user and a group called tftp. The tftp server runs as this user. I recommend adding yourself to the tftp group and changing permissions on the tftp data directory to 775. This will let you read and write to the tftp data directory without switching to root each time. Moreover, if files in the tftp data directory are owned by root, the tftp server will not be able to read and serve them over the network. You can test your server by placing a file there and attempting to get it using the tftp client:

```
$ tftp 192.168.146.1 -c get uImage[COMMAND]
```

Some common problems you may face include errors due to permission. Make sure that the files are readable by the tftp user or whichever user the

Listing 6. An Example of the uenvcmd Variable for DHCP Booting

```
echo Booting the BeagleBone Black from LAN (DHCP)...
dhcp ${kloadaddr}
tftpboot ${fdtaddr} /BI/${fdtfile}
setenv bootargs console=${console} ${optargs} root=${mmcroot}
 ➥rootfstype=${mmcrootfstype} optargs=quiet
bootm ${kloadaddr} - ${fdtaddr}
```

Listing 7. An Example of uenvcmd Variable for TFTP Booting

```
echo Booting the BeagleBone Black from LAN (TFTP)...
env set ipaddr 192.168.146.10
env set serverip 192.168.146.1
tftpboot ${kloadaddr} /BI/${bootfile}
tftpboot ${fdtaddr} /BI/${fdtfile}
setenv bootargs console=${console} ${optargs} root=${mmcroot}
 ➥rootfstype=${mmcrootfstype} optargs=quiet
bootm ${kloadaddr} - ${fdtaddr}
```

tftpd runs as. Additionally, directories must have execute permission, or tftp will not be able to descend and read the content of that directory, and you'll see a "Permission denied" error when you attempt to get the file.

U-Boot Scripting

Now that you have your DHCP and TFTP servers working, let's write a U-Boot script that will fetch the kernel image and boot it. I'm going to present two ways of doing this: using DHCP and using only TFTP. As I mentioned before, running a poorly configured DHCP server will cause a network-wide disruption of services. However, if you know what you are doing and have prior experience with setting up network services, this is the simplest way to boot the board.

A DHCP boot can be initiated simply by adding or modifying the `uenvcmd` variable in the uEnv.txt file, as shown in Listing 6. uEnv.txt is found in the FAT32 partition of the BeagleBone Black. This partition is available to be mounted when the

BeagleBone Black is connected to your computer via USB cable.

For a TFTP-only boot, you manually specify an IP address for the development board and the TFTP server. This is a much safer process, and you incur very little risk of interfering with other users on the network. As in the case of configuring to boot with DHCP, you must modify the `uenvcmd` variable in the uEnv.txt file. The script shown in Listing 7 is an example of how to set up your BeagleBone Black to get a kernel image from the TFTP server and pass on the execution to it.

Both Listing 6 and 7 are formatted to give a clear understanding of the process. The actual uEnv.txt file should look something like the script shown in Listing 8. For more information about U-Boot scripting, refer to the U-Boot FAQ (**http://www.denx.de/wiki/DULG/Faq**) and U-Boot Manual (**http://www.denx.de/wiki/DULG/Manual**). The various commands in the `uenvcmd` variable must be on the same line separated by a semicolon. You may notice that I place my script in `uenvcmdx` instead of `uenvcmd`. This is because `test -n` throws an error to the console based on the content of the variable it is testing. Certain variable contents, especially long complicated scripts, cause the `test -n` to fail with an error message to the console. Therefore, I put a simple command to `run uenvcmdx` in `uenvcmd`. If you find that your script from the uEnv.txt is not being executed, look for an error on the serial console like this:

```
test - minimal test like /bin/sh

Usage:
test [args..]
```

Listing 8. An Example of uEnv.txt for TFTP Booting

```
optargs=quiet
uenvcmdx=echo Booting the bone from emmc...; env set ipaddr
 ➥192.168.146.10; env set serverip 192.168.146.1; tftpboot
 ➥${kloadaddr} /BI/${bootfile}; tftpboot ${fdtaddr}
 ➥/BI/${fdtfile}; setenv bootargs console=${console}
 ➥${optargs} root=${mmcroot} rootfstype=${mmcrootfstype}
 ➥optargs=quiet; bootm ${kloadaddr} - ${fdtaddr}
uenvcmd=run uenvcmdx
```

On some development boards like the BeagleBoard xM (**http://beagleboard.org/Products/BeagleBoard-xM**), the Ethernet port is implemented on the USB bus. Therefore, it is necessary to start the USB subsystem before attempting any network-based boot. If your development board does not hold a Flash memory on board, it may not have a MAC address either. In this case, you will have to set a MAC address before you can issue any network requests. You can do that by setting the environment variable `ethaddr` along with the rest of the uEnv.txt script.

An alternative but cumbersome way to change the default boot sequence is to modify the U-Boot source code. Modifying the source code gives you greater versatility for booting your development board. When you interrupt the U-Boot boot sequence, drop to the U-Boot shell and issue the `env print` command, you'll see a lot of environment variables that are defined by default. These environment variables are defined as macros in the source

Listing 9. Part of the u-Boot/include/configs/am335x_evm.h File Responsible for the Default Script in the bootcmd Variable

```
#define CONFIG_BOOTCOMMAND \
  "mmc dev ${mmcdev}; if mmc rescan; then " \
    "echo SD/MMC found on device ${mmcdev};" \
    "if run loadbootenv; then " \
      "echo Loaded environment from ${bootenv};" \
      "run importbootenv;" \
    "fi;" \
    "if test -n $uenvcmd; then " \
      "echo Running uenvcmd ...;" \
      "run uenvcmd;" \
    "fi;" \
    "if run loaduimage; then " \
      "run mmcboot;" \
    "fi;" \
  "fi;" \
```

code. Modifying the source code aims at modifying these variables. As shown in Figure 1, U-Boot begins loading the kernel by executing the script in `bootcmd`. Hence, this is the variable that must be modified.

To begin, you'll need the source code to U-Boot from the git repository:

```
$ git clone git://git.denx.de/u-boot.git
```

Before making any modifications, I recommend compiling the unmodified source code as a sanity check:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- am335x_evm_config

$ make -j 8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

This most likely will work without a hitch. Now you can modify the u-Boot/include/configs/am335x_evm.h file. In this file, you'll find code similar to Listing 9. Modify this as you please and re-compile. Depending on your target board, you will have to modify a different file. The files to some common target platforms are:

- Panda Board (**http://pandaboard.org**): u-Boot/include/configs/omap4_common.h
- BeagleBoard: u-Boot/include/configs/omap3_beagle.h

Conclusion

I hope the instructions provided here help you create a system to develop and deploy bare-metal programs and kernel images quickly. You also may want to look into u-boot-v2, also known as Barebox (**http://barebox.org**). The most helpful code modification that I suggest here is to compile the U-Boot with an elaborate boot sequence that you can tailor to your needs with the least modifications. You can try out some fancy scripts to check and update firmware over LAN—I would consider that really cool. Write to me at bharath (you-know-what) lohray (you-know-what) com.■

Bharath Bhushan Lohray is a PhD student working on his dissertation on image compression techniques at the Department of Electrical and Computer Engineering, Texas Tech University. He is interested in machine learning and considers building a terminator the echelon of human advancement—at least until we build the terminator.

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.