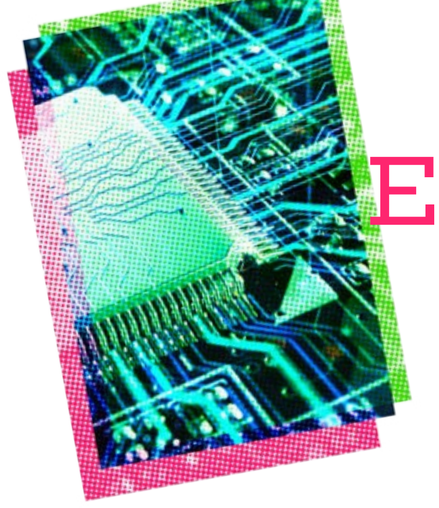

Embedded FreeBSD

Building U-boot

BY CHRISTOPHER R. BOWMAN

A reader wrote to me that he had trouble building U-boot, so I thought I'd walk through the process since I wanted to bring up a different Zynq board and would have to go through it anyway. I need to provide a disclaimer: what I've written below is accurate, but these are complex systems, and I could have gotten some details wrong. I'd be grateful to hear from you if you think I have.

As we've discussed before, U-boot is both the second- and third-stage boot loader that runs and loads the FreeBSD loader, which, in turn, loads the FreeBSD kernel itself. U-boot is an open-source community project used on a wide variety of systems to provide boot services. Documentation is available through their website: The U-boot Documentation.

U-boot is both the second- and third-stage boot loader that runs and loads the FreeBSD loader.

On AMD/Xilinx Zynq chips, the first-stage boot loader is in BootROM on the Zynq chip itself. The Zynq boot process is described in Chapter 6, "Boot and Configuration" of the Zynq 7000 SoC Technical Reference Manual. The short description is that when powering on the device, it samples some pins and, depending on their state, chooses one of several boot methods. This allows a jumper on the board (JP5 according to Fig. 2.1 of the Zybo Z7 Reference Manual) to select a boot method, one of which is loading from an SD card. If you select this method, the BootROM code looks for a file named **`boot.bin`** on a FAT16 or FAT32 partition on the SD card. This is what U-boot calls the secondary program loader or SPL. The Zynq chip contains a small amount of onboard RAM, thus limiting the size of the program that the BootROM can load. In non bare metal applications, the SPL must contain enough code to configure the Zynq chip (PLL's, memory interfaces, and more) so that it can bring up the memory system and load U-boot proper. U-boot proper is a larger version with more features (like file systems) supported.

We're quite lucky as the Zybo Z7 is already supported in U-boot. So, we just need to get the build process to work. U-boot is typically built on a host system for a target system. The U-boot documentation referenced above suggests that we should set the compiler to be used for this cross-compiling using an environment variable. We will use the GCC compiler and GNU tools to do the cross-compiling, so we need to install those packages and set the cross compiler. U-boot is also built using **`gmake`**, not the standard BSD make, so we'll need to install that, as well as some other packages:

```
pkg install gmake
pkg install arm-none-eabi-gcc
pkg install bison
pkg install gnutls
pkg install gmake
pkg install pkgconf
pkg install coreutils
pkg install dtc
pkg install gdd

setenv CROSS_COMPILE arm-none-eabi-
```

Odd that you use **arm-none-eabi-**, and not **arm-none-eabi-gcc**, but it's not a typo.

Next, we need to configure the U-boot source tree for the board we want to target. The Zybo Z7 board is supported by the **xilinx_zynq_virt_defconfig** located in the **configs** directory. This configuration supports multiple boards, one of which is the Zybo Z7. To configure the source tree, we run:

```
make xilinx_zynq_virt_defconfig
```

But we have to be careful that we pull in the GNU **make**, not the BSD **make**. To do this, I've created a directory with a symlink named **make** that points to **/usr/local/bin/gmake**, and I've set this directory to be first in my path. This seems to work well. From there, we can just call **make** and wait (I highly recommend using the -j flag if you have extra cores). Did it error out for you as it did for me?

I get this output:

```
make[1]: *** [scripts/Makefile.xpl:257: spl/U-boot-spl-align.bin] Error 1
make: *** [Makefile:2358: spl/U-boot-spl] Error 2
make: *** Deleting file 'spl/U-boot-spl'
```

The relevant lines from **scripts/Makefile.xpl** are

```
$(obj)/$(SPL_BIN)-align.bin: $(obj)/$(SPL_BIN).bin
        @dd if=$< of=$@ conv=block,sync bs=4 2>/dev/null;
```

If you remove the redirection of output to **/dev/null**, you'll see a complaint from dd:

```
dd: record operations require cbs
```

Seems FreeBSD's **dd** is not command line equivalent with the GNU version. Originally, I simply used the GNU version of **dd** by installing the package and then creating a symlink in my local bin directory, but it turns out you can simply remove "block" from the **dd** command.

Also, the **V make** variable can be set to control the verbosity of build output. If your build doesn't work, I highly recommend running again with only one processor and V=1:

```
make V=1
```

If everything builds without error, you should have a **U-boot.img** file and an **spl/boot.bin** file. These are U-boot proper and the secondary program loader. Copy these to your SD card and give it a whirl!

Wha, wait, didn't work? Huh! As I said, this configuration supports multiple boards, and its default device tree isn't for the Zybo Z7. Consulting the board-specific documentation referenced above, we can specify which device tree is the default by setting DEVICE_TREE:

```
setenv DEVICE_TREE zynq-zybo-z7
```

This will override the default DTS in the configuration file. Build it again and try it. Wait, what? Another problem? The kernel loads, but it crashes in probing? Oh right. FreeBSD DTS requirements are not the same as Linux. The compat strings required to get some hardware recognized are different, and FreeBSD seems to require some clock-frequency properties, though I'm not sure the values are used. It might make sense to add compat values to the FreeBSD drivers that match what Linux expects, but I'm not a committer. I had to add the following to the DTS file in **arch/arm/dts/zynq-zybo-z7.dts**:

```
&sdhci0 {
        compatible = "arasan,sdhci-8.9a", "xlnx,zy7_sdhci";
   U-boot,dm-pre-reloc;
        status = "okay";
};

&devcfg {
compatible = "xlnx,zynq-devcfg-1.0", "xlnx,zy7_devcfg";
status = "okay";
};

&global_timer {clock-frequency = <50000000>;};
&ttc0 {clock-frequency = <50000000>;};
&ttc1 {clock-frequency = <50000000>;};
&scutimer {clock-frequency = <50000000>;};
```

Now that we've learned to build U-boot, let's see if we can make it a port. There are a whole bunch of U-boot ports, all of which are built off the U-boot-master port. To use them, we need to include the master port **Makefile**. We have to specify the board, the model, and the config that should be used. We have a few patches for the changes we made above, and we end up with the following.

```
MASTERDIR=      ${.CURDIR}/../U-boot-master

MODEL=          zybo-z7
BOARD_CONFIG=   xilinx_zynq_virt_defconfig
FAMILY=         zynq_7000

EXTRA_PATCHES=  ${.CURDIR}/files

BUILD_DEPENDS+= gdd:sysutils/coreutils

COMMENT=        ported by Christopher R. Bowman <my_initials>@ChrisBowman.com

.include "${MASTERDIR}/Makefile"
```

I hope you've found these columns useful. I'd appreciate your comments or feedback. You can contact me at articles@ChrisBowman.com.

CHRISTOPHER R. BOWMAN first used BSD back in 1989 on a VAX 11/785 while working two floors below ground level at the Johns Hopkins University Applied Physics Laboratory. He later used FreeBSD in the mid 90's to design his first 2 Micron CMOS chip at the University of Maryland. He's been a FreeBSD user ever since and is interested in hard-ware design and the software that drives it. He has worked in the semiconductor design automation industry for the last 20 years.