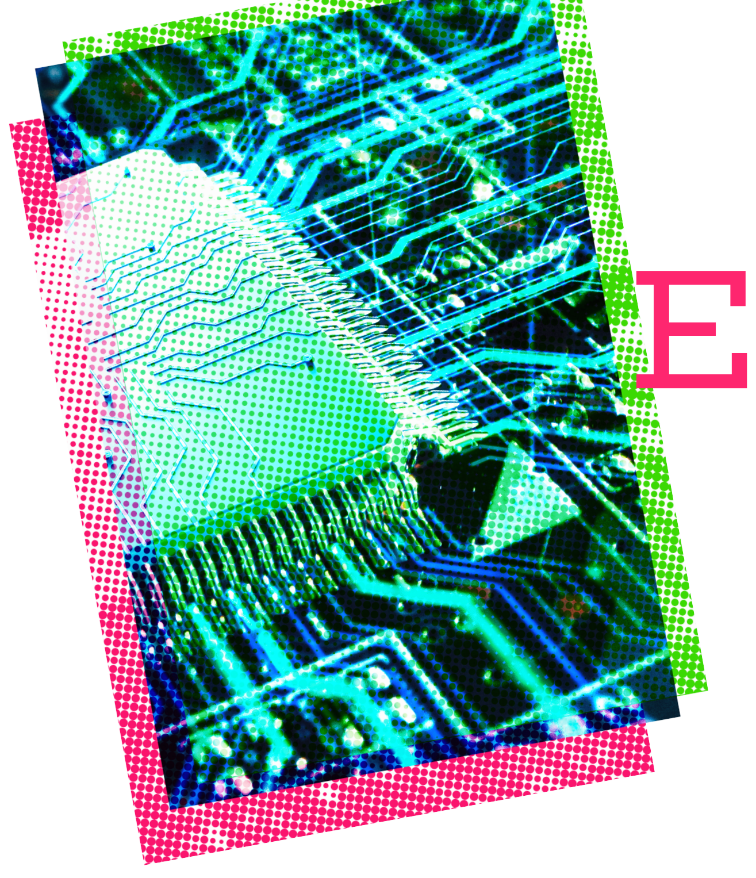

Embedded FreeBSD

Rolling Your Own Images

BY CHRISTOPHER R. BOWMAN

In the last column, I talked a little about the board I've been using, the Digilent ARTYZ7. In this one, I'm going to talk about rolling your own images. At some point you're going to want an image slightly different from the one you have, and so you'll want to build from source and create an image to write to an SD card.

Let's start by understanding exactly how the ARTYZ7 boots. The Zynq-7000 SoC Technical Reference Manual is a gold mine of technical information, and in Chapter 6, it talks about how the chip and thus all Zynq boards boot. There are a ton of technical details, but based on the way the jumpers are configured by default, the ARTYZ7 boots from the SD card. You may have downloaded and poked around the image I provided in the previous column. If you did, you'll see that it's formatted in MBR mode with a FAT partition first and a second MBR partition with a FreeBSD slice that has UFS on it. The processor will look for an MBR and look for the FAT16 or FAT32 partition. It will look for a file **`boot.bin`** on the FAT partition. If it finds one, it will load that into memory and begin executing that. If you wanted to program to the bare metal, you could write your application and call it **`boot.bin`**. That's a little much for me. Instead, when booting FreeBSD a first stage boot loader is used, and like many embedded boards we use Das U-boot. U-boot for short is an open-source community-maintained boot loader. In our case U-boot is used twice. First, U-boot is compiled as a minimal First Stage Boot Loader (FSBL) which sets up the hardware and then looks for a second stage boot loader which is also U-boot but compiled with a much richer functionality. In our usage, the second stage U-boot lives in the file named **`U-boot.img`** on the FAT partition. This U-boot second stage loader loads the lua loader from the file **`EFI/BOOT/bootarm.efi`** on the FAT partition. The lua loader will then load the kernel, among other things, into memory and execute it.

At some point you're going to want an image slightly different from the one you have.

So, if we're going to build an image, we're going to need to create the partitions and get U-boot installed as well as FreeBSD on an SD card.

Building U-boot is relatively straightforward. While there are many ports for U-boot for various boards, there isn't one for the ARTYZ7. I've created one which you can find HERE. While I haven't managed to get this into the FreeBSD ports tree, you can simply drop this into the **/usr/ports/sysutils** directory in a recent ports tree. Chapter 4.5 of the FreeBSD handbook has really excellent instructions for installing ports via git and building a port. Once you've added the port to your ports tree a simple **make** in **sysinstall/u-boot-artyz7** should cause U-boot to be downloaded and built automatically. After **make install** the files **boot.bin** and **U-boot.img** should appear in **/usr/local/share/U-boot/U-boot-artyz7**. Note: I used to be able to use high "**-j**" values with **make** to use multiple cores to build but that seem to fail on the most recent ports tree (2024Q3)

Building from source is also well documented in the FreeBSD handbook. In Chapter 26.6. Updating FreeBSD from Source you'll find a lot of information on downloading the FreeBSD sources and building from them. If you have the FreeBSD sources install in **/usr/src** you can simply go there and run:

```
# make buildworld
# make buildkernel KERNCONF=ARTYZ7
```

While this should work just fine on the ARTYZ7 boards, after all it has a full FreeBSD install, you should be prepared for it to take a long, um **long** time. As PC hardware has become so incredibly powerful and inexpensive, I use an AMD64 system to host all my files, development environment, and do all my building. FreeBSD has built-in support for cross compiling and building. On my PC I use the following to get my PC to build from sources for my ARM based ARTYZ7 board:

```
# make buildworld TARGET=arm TARGET_ARCH=armv7 -j32
# make buildkernel KERNCONF=ARTYZ7 TARGET=arm \
    TARGET_ARCH=armv7 -j32
```

This will build a cross compiler from AMD64 to ARMv7 and use that compiler to build everything.

For a kernel config file, I've copied the ZEDBOARD config in **src/sys/arm/conf/ZEDBOARD** to **src/sys/arm/conf/ARTYZ7** and I've changed the name in there to match. The **-j32** switch causes the compile processes to use up to 32 processes while doing the builds. On an AMD 5950x PC with a fast SSD it takes about 10 minutes to build world and another 60 seconds or so to build the kernel. I couldn't imagine how many days it would take on the ARTYZ7.

Once you build everything from source the process can diverge in a couple of ways:

1. You can mount an SD card on your host development system and install directly onto the SD card from the host.
2. You can use the **mdconfig** command to create a file-backed memory device. This allows you to treat a file as if it were a block device. You can use all the standard FreeBSD tools to partition the device and mount the partitions in the filesystem. From there you can install as if it were a physical device and, at the end, you're left with a file that is suitable for use with **dd** to copy to an SD card or provide to others
3. The final method, and the one I'm using and will discuss here, is to install to a directory on my host PC and then use **mkimg** and **makefs** to build an image from the host directory which is again suitable for copying to an SD card using **dd**.

Let's look a little more closely at method 3. I typically create an msdos directory and a ufs directory to represent the two partitions I'm going to need on my SD card:

```
# mkdir msdos ufs
```

Next, I do an install to the **ufs** directory:

```
# make installworld installkernel TARGET=arm \
    TARGET_ARCH=armv7 -j32 DESTDIR=ufs
```

You'll also need to run the distribution target which creates all the default configuration files in **/etc**. When doing a source upgrade of a working box, you wouldn't normally run this, as it would overwrite your configuration files, but when building a system from scratch, you do want default configuration files:

```
# make distribution TARGET=arm \
    TARGET_ARCH=armv7 DESTDIR=ufs -j32
```

At this point, I have a complete install in **ufs** and I can go in and do any customization I want to my image. For instance, I can customize **ufs/etc/rc.conf** to automatically bring up the ethernet interface, **cgem0** using DHCP. I can install **ssh** keys into **ufs/etc/ssh** so that the system has the same **ssh** keys all the time instead of having new ones generated on first boot. Since I'm using a host system to do all my building and host all my files, I like to configure **/etc/fstab** to nfs mount my home directory.

I've found it very handy to create a user account so I can log in immediately:

```
# echo 'xxxx' | pw -R ${ufs} useradd -n crb -m -u 1001 \
    -d /homes/crb -g crb -G 1001,wheel,operator\
    -c "Christopher R. Bowman" -s /bin/tcsh -H 0
```

The **-R** option to pw causes pw to edit the password files in the **ufs/etc** instead of my host system. The **-H0** option allows me to use **echo** to pipe my password to pw without having to type it in interactively (you'll need to use the encoded password entry from your host system here instead of **xxxx**). You may also find it handy to modify the root account so that it has a password instead of no password.

Now that I've got the **ufs** directory customized as I want it to appear in my image, let's turn our attention to the FAT partition.

I need to install **boot.bin** and **U-boot.img**:

```
# cp /usr/local/share/U-boot/U-boot-artyz7/boot.bin msdos
# cp /usr/local/share/U-boot/U-boot-artyz7/U-boot.img msdos
```

boot.bin loads **U-boot.img** which emulates EFI firmware for the FreeBSD loader. EFI systems look for a file on a FAT partition called **EFI/BOOT/bootarm.efi** so we'll copy the FreeBSD lua loader there:

```
# mkdir -p msdos/ EFI/BOOT
# cp ufs/boot/loader_lua.efi msdos/EFI/BOOT/bootarm.efi
```

Note I copied the ARM version from our build not the host version which would be AMD64 code.

Now, we've got our two directories **msdos** and **ufs** which contain the files we want on the SD card we just need to create the image file. This is a 4-step process:

```
makefs -t msdos \
  -o fat_type=16 \
  -o sectors_per_cluster=1 \
  -o volume_label=EFISYS \
  -s 32m \
  efi.part msdos

makefs -B little \
  -o label=rootfs \
  -o version=2 \
  -o softupdates=1 \
  -s 3g \
  rootfs.ufs ufs

mkimg -s bsd \
  -p freebsd-ufs:=rootfs.ufs \
  -p freebsd-swap::1G \
  -o freebsd.part

mkimg -s mbr -f raw -a 1\
  -p fat16b:=efi.part \
  -p freebsd:=freebsd.part \
  -o selfbuilt.img
```

The first command builds a file system image file (**efi.part**) from the **msdos** directory. The second command builds a file system image file (**rootfs.ufs**) from the **ufs** directory. The third command takes the **rootfs.ufs** file and assembles it into a FreeBSD slice with a 1 gigabyte swap partition. The final command bundles our **efi.part** and **freebsd.part** files into a single image (**selfbuilt.img**).

If I insert my SD card into my host I get a **/dev/da0** device and a simple **dd** will copy the image onto the SD card:

```
# dd if=selfbuilt.img of=/dev/da0 bs=1m status=progress
```

At this point all that's left to do is insert the SD card into the ARTYZ7 board, hit the reset button and watch the glorious boot process. Because of the configuration I did in the ufs partition, I'm able to **ssh** into my board as soon as it's done booting and my home directory is nfs mounted. At this point it's time for world domination! Or a beer — it's always time for beer.

CHRISTOPHER R. BOWMAN first used BSD back in 1989 on a VAX 11/785 while working 2 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 hardware design and the software that drives it. He has worked in the semiconductor design automation industry for the last 20 years.