How to build bootable ARM and AArch64 images

This article describes, step by step, how to build bootable ARM and AArch64 images as part of a NetBSD release using the `build.sh` script. Note that because `build.sh` handles cross-platform builds of NetBSD, the process described here is platform-independent (except for details such as filesystem paths).

Contents

Introduction

Release builds for ARM and AArch64 platforms include compressed system images in `${RELEASEDIR}/${RELEASEMACHINEDIR}/binary/gzimg`. However, they may not be bootable. Consequently, boot blocks must be manually installed in the images, which is an extra barrier for testing systems or adopting NetBSD. This has prompted creation of external repositories, e.g., armbsd.org, to host a collection of bootable images. However, this does not ease the burden on developers compiling their own systems; for them, manual installation of boot blocks has generally been required.

For ARM and AArch64 platforms, `/usr/src/etc/etc.evbarm/Makefile.inc` contains the commands used to build system images. Because installboot(8) can write boot blocks directly to system images, it is possible for `build.sh` to build bootable images during the normal process of building a release.

In the case of many ARM and AArch64 platforms, installboot(8) uses U-Boot boot blocks, which are not part of the NetBSD source code. Developers can, however, install as many U-Boot boot blocks as needed, either with pkgsrc in the default location of `/usr/pkg/share/u-boot` or in an arbitrary set of directories. For each board with an available boot block, a board-specific bootable image can be built by `build.sh` in `${RELEASEDIR}/${RELEASEMACHINEDIR}/binary/gzimg`. If a boot block is not available, no additional images will be built.

This facility creates opportunities to build bootable images for any number of boards within the scope of a standard release build. The following steps are required to build bootable ARM and AArch64 images as part of a normal NetBSD release.

- Install U-Boot boot blocks.
- Set environment variables (optional).
- Run `build.sh`.
- Copy image(s) to bootable media.

Conventions

There are of course many different ways that the NetBSD source code and pkgsrc may be set up. As a result, specific filesystem paths may differ. To be concrete, this tutorial assumes the following setup, which is typical for many NetBSD systems:

- `/usr/pkg` -- location of installed pkgsrc files
- `/usr/pkgsrc` -- location of pkgsrc source files
- `/usr/src` -- location of NetBSD source files
- `${RELEASEDIR}` -- location of the NetBSD release files, which may be set with the `build.sh -R` option.
- `${RELEASEMACHINEDIR}` -- location (within `${RELEASEDIR}`) of release files for a specific machine, e.g., `evbarm-earmv7hf`.
- `make` -- make(1) command used to build NetBSD and pkgsrc; this should be BSD make (not GNU make), which may be invoked as `bmake` on some systems.

If your setup differs, then substitute the corresponding paths from your system into the text or commands referred to throughout this document.

Install U-Boot boot blocks

Boot blocks must be available in order to build bootable ARM and AArch64 images. Although they may be downloaded from any suitable site and installed in a location accessible to `build.sh`, pkgsrc includes a large number of U-Boot packages covering a variety of different boards; for a complete list, see the sysutils page from the pkgsrc wiki. As a result, the appropriate boot blocks can be built and installed using pkgsrc. For example, the following commands make the U-Boot boot blocks for the Beaglebone Black board:

```
# cd /usr/pkgsrc/sysutils/u-boot-beagleboneblack
# make install
```

Note that each U-Boot boot block generally consists of multiple files. Consequently, all the files composing one U-Boot boot block are installed within a single boot block-specific directory. For example, the commands above place all the files within the directory `/usr/pkg/share/u-boot/beagleboneblack`.

Set environment variables

This step is optional as the relevant environment variables can also be set with the `-V` option when `build.sh` is run (see below).

Two environment variables must be set so that `build.sh` will build bootable ARM and AArch64 images; both are described in `/usr/src/BUILDING`.

- `INSTALLBOOT_BOARDS` -- A (space-separated) list of boards to build bootable images for. If corresponding U-Boot packages are installed, bootable images are built as part of a release. See the `-o "board="` option of installboot(8). The following command will display all the supported boards (the `-v` option additionally identifies the corresponding U-Boot packages):

```
# installboot -m evbarm
```

- `INSTALLBOOT_UBOOT_PATHS` -- A colon-separated list of search paths used by installboot(8) to find U-Boot packages. Note that the search paths defined here correspond to directories that contain individual boot block-specific directories; often, only a single path is required regardless of the number of boot blocks to be used.

For the Bourne shell, the environment may be set as follows to build bootable images for the TI AM335x BeagleBone Black and Pine64 boards:

```
# export INSTALLBOOT_BOARDS="ti,am335x-bone-black pine64,pine64"
# export INSTALLBOOT_UBOOT_PATHS=/usr/pkg/share/u-boot
```

Other shells may require different syntax to define environment variables.

Run `build.sh`

The `build.sh` script is run the same as when not creating bootable ARM and AArch64 images. If the environment is set as described above and appropriate boot blocks are available, bootable images will be built; otherwise, only the default set of images will be built.

An alternative to setting environment variables as described in the previous section is to use the `-V` option when running the `build.sh` script. For example, the following has the same effect as setting the environment variables:

```
# cd /usr/src
# ./build.sh -V INSTALLBOOT_BOARDS="ti,am335x-bone-black pine64,pine64" \
             -V INSTALLBOOT_UBOOT_PATHS=/usr/pkg/share/u-boot release
```

Note that in practice additional options are normally selected when running the `build.sh` script; see `/usr/src/BUILDING` for details.

If all is well, the output of the `build.sh` script will identify the bootable images as they are made. For example, with the boards selected above, the following lines should appear in the output

```
===> Making bootable image gzimg/armv7-ti,am335x-bone-black.img.gz
===> Making bootable image gzimg/armv7-pine64,pine64.img.gz
```

and the following files should be built:

```
# cd ${RELEASEDIR}/${RELEASEMACHINEDIR}/binary/gzimg
# ls -1      # note: -1 = hyphen-one (not el)
armv7-pine64,pine64.img.gz
armv7-ti,am335x-bone-black.img.gz
armv7.img.gz
```

The first two of these files are the bootable ARM and AArch64 images; the last is the only (unbootable) system image that is built by default.

If images for boards listed in the environment variable `INSTALLBOOT_BOARDS` are missing, the most likely explanation is that the corresponding U-Boot boot blocks are not available within any of the directories listed in the environment variable `INSTALLBOOT_UBOOT_PATHS`.

Copy image to bootable media

The command `build.sh release` will build a full release of NetBSD, which includes a set of system images in the directory `${RELEASEDIR}/${RELEASEMACHINEDIR}/binary/gzimg`. When the environment is set up correctly, the system images will include, in addition to the default image, e.g., `armv7.img.gz`, board-specific images, e.g., `armv7-ti,am335x-bone-black.img.gz`; these are the bootable images with U-Boot boot blocks. Copy the appropriate file (uncompressed) to bootable media. For example, the following commands copy the Beaglebone Black image to `/dev/rsd0d`:

```
# cd ${RELEASEDIR}/${REELASEMACHINEDIR}/binary/gzimg
# gzcat armv7-ti,am335x-bone-black.img.gz | dd of=/dev/rsd0d bs=1m conv=sync
```

Be certain to select an appropriate output device, as it will be overwritten by the system image and data loss will occur.

Conclusion

Building bootable ARM and AArch64 images as a normal part of a NetBSD release can be fully and easily automated. One-time setup includes two-steps: (i) defining two environment variables, either within the build environment or using `-V` options to `build.sh`, and (ii) installing the needed U-Boot boot blocks. In fact, if boot blocks are not available, `build.sh` will build only the familiar set of release files, even if the variables are set. Thus, defining these variables as part of system setup enables `build.sh` to build bootable system images once U-Boot boot blocks are installed in the future. This process can simplify greatly the task of developing NetBSD for ARM and AArch64 boards, because it automates what would otherwise be manual steps to build bootable images from the single (unbootable) system image that is traditionally built.