Lab 2-1 Prepare Boot Image of Xilinx ZC702

Wen-Lin Sun

2018-9-21

In this lab, we will prepare the things that we need when booting up Linux on Xilinx ZC702. The files are first-stage boot loader(FSBL), second-stage boot loader(e.g. U-boot), device tree, kernel image and root filesystem. Make sure you have these files after you finish this lab:

- boot.bin (fsbl.elf + u-boot.elf)
- uImage
- uramdisk.image.gz
- devicetree.dtb

And put them into a directory (create it yourself).

1 Environment Setup

Before building all of these files, we should install several tools.

- Install 32-bit libraries:
  Update the database before we install anything.

```
SHELL> sudo apt-get update
SHELL> sudo apt-get install lib32z1 lib32ncurses5 lib32ncurses5-dev lib32stdc++6
libbz2-1.0:i386
```

- Download & Install Xilinx Vivado Design Suite (you may need to register):
  Recommend: Install it under your home directory.
  Official website: https://www.xilinx.com/support/download.html

- Setup Environment Variables:

```
SHELL> export CROSS_COMPILE=arm-linux-gnueabihf-
SHELL> source [xilinx_directory]/Xilinx/SDK/2018.2/settings64.sh
```

- Install Git:

```
SHELL> sudo apt-get install git
```

- Build & Install device tree compiler:

```
SHELL> git clone https://git.kernel.org/pub/scm/utils/dtc/dtc.git
SHELL> cd dtc
SHELL> sudo apt-get install flex bison swig python-dev
SHELL> make
SHELL> export PATH=‘pwd‘:$PATH
```

2 Generate FSBL

You should download the hardware definition file `system.hdf from E3 first.`

- Create a link gmake to make:
  Formally, we should install gmake, but we can also use make instead here.

```
SHELL> sudo ln -s /usr/bin/make /usr/bin/gmake
SHELL> cd <hdf_dir>
SHELL> hsi
hsi% set hwdsgn [open_hw_design system.hdf]
hsi% generate_app -hw $hwdsgn -os standalone -proc ps7_cortexa9_0 -app zynq_fsbl
-compile -sw fsbl -dir <fsbl_dir>
hsi% exit
```

  The file fsbl will be placed at <fsbl_dir> as you set in the command. Just rename it to fsbl.elf.

3 Build U-boot

In this lab, we use u-boot as our second-stage boot loader, for helping us boot from Linux. Since the tool mkimage we use when building kernel image will also be generated when building u-boot, we should make sure this step is done correctly before we continue.

- Build u-boot:

```
SHELL> git clone https://github.com/Xilinx/u-boot-xlnx.git
SHELL> cd u-boot
SHELL> make zynq_zc702_defconfig
SHELL> make
```

  The file u-boot will be generated in this directory and the mkimage will be placed under the directory tools. Just rename it to u-boot.elf.

- Export mkimage tools to PATH

```
SHELL> cd tools
SHELL> export PATH=‘pwd‘:$PATH
```

4 Build uImage & Device Tree

In this section, we will build kernel image and attach it with u-boot recognizable header. Alternatively, we will build the the corresponding device tree.

- Download & Configure Linux kernel:
  In this lab, we use the version 2016.4.

```
SHELL> git clone https://github.com/Xilinx/linux-xlnx.git
SHELL> git checkout tags/xilinx-v2016.4
SHELL> make ARCH=arm xilinx_zynq_defconfig
SHELL> make ARCH=arm menuconfig
```

- Build kernel image with u-boot header (uImage):

```
SHELL> make ARCH=arm UIMAGE_LOADADDR=0x8000 uImage
```

  The uImage will be placed under arch/arm/boot.

- Build device tree:

```
SHELL> make ARCH=arm zynq-zc702.dtb
```

  The device tree zynq-zc702.dtb will be placed under arch/arm/boot/dts. Just rename it to devicetree.dtb.

5 Modify Root Filesystem

In this lab, we will use BusyBox as our root filesystem.

- Download & Configure BusyBox:
  We use the branch 1_29_stable for example.

```
SHELL> git clone git://git.busybox.net/busybox -b 1_29_stable
SHELL> make ARCH=arm menuconfig
```

  Modify the configuration as you need. Make sure [Setting] -> [Build Options] -> [Build static binary (no shared libs)] is selected. And then, build it.

- Build & Generate Busybox and related files:

```
SHELL> make ARCH=arm
```

  After that, use make install to gather the compiled Busybox and symlink into the directory _install.

```
SHELL> make ARCH=arm install
```

- Modify the filesystem:

  - Manually add missing directories of root filesystem to _install. (/dev, /sys, /proc, /root and /etc/init.d)

```
SHELL> cd _install
SHELL> mkdir -p dev sys proc root etc/init.d
```

- Manually create initial script rcS under etc/init.d and make it executable.

```
#!/bin/sh

mount -t proc none /proc
mount -t sysfs none /sys
/sbin/mdev -s
```

- Manually create inittab under etc.

```
#!/bin/sh

# Init script
::sysinit:/etc/init.d/rcS

# Start shell on the serial ports
::respawn:/sbin/getty -L ttyPS0 115200 vt100

# Execute /sbin/init when restarting the init process
::restart:/sbin/init

# Umount everything before rebooting
::shutdown:/bin/umount -a -r
```

- Set etc/passwd to make root be able to login without password.

```
root::0:0:root:/root:/bin/sh
```

- Create soft link to /init to avoid that kernel cannot found the init.

```
SHELL> ln -s /sbin/init init
```

- After changing all the directory to be owned by root, you can pack the root filesystem with cpio format. And then, compress it with gzip format. The output file will be named ramdisk.cpio.gz and placed under _install.

```
SHELL> cd ..
SHELL> sudo chown -R root:root _install
SHELL> cd _install
SHELL> find .  | cpio -H newc -o | gzip -9 > ../ramdisk.cpio.gz
```

- Transform the ramdisk.cpio.gz into u-boot format with mkimage.

```
SHELL> mkimage -A arm -T ramdisk -C gzip -d ramdisk.cpio.gz uramdisk.image.gz
```

Now you have the compressed root filesystem file uramdisk.image.gz under busybox.

6 Prepare Boot Image

In this section, we will generate the boot image boot.bin. Before doing this step, make sure you have fsbl.elf and u-boot.elf.

- Put them into a directory.
- Enter the directory and create a file boot.bif with following content (sample):

```
image : {
        [bootloader]fsbl.elf
        u-boot.elf
}
```

- Generate the boot image.
  bootgen merges fsbl.edf and u-boot.elf to boot image boot.bin.

```
SHELL> bootgen -image boot.bif -o i boot.bin
```