Linux on ZEDBoard

Zinching Dang, Christian Brugger

June 30, 2015

1 Pre-Requirements

- ZEDBoard, SD-card, micro-USB cable
- running Linux system with root access (a virtual machine will do the job, if it can access the SD-card)
- putty (Windows) or screen (Linux) to access the serial port
- some basic Linux knowledge (kernel compiling needs some more knowledge to understand what is going on, but not neccessray, and some cups of coffee)

2 Prepare SD-card

2.1 Format SD-card

- format SD-card using any partitioning tool, e.g. gparted
- delete any existing partitions and create two partitions
  1. 50MB: primary partition, filesystem: fat16, label: boot, flags: boot
  2. remaining space: primary partition, filesystem: ext4, label: root, no flags
- apply changes to SD-card

2.2 Write files on SD-card

- get latest linaro server version from
  `http://releases.linaro.org/latest/ubuntu/raring-images/server`
- download tar.gz file, optionally check md5 or sha1 check sum
- the following instructions need root privileges: remove and re-insert SD-card, and mount the filesystems to `/mnt/boot/` and `/mnt/root`. If the filesystems are mounted automatically, use the auto-mounted paths. If not use following commands:
  ```
  mkdir /mnt/boot /mnt/root
  mount /dev/mmcblk0p1 /mnt/boot
  mount /dev/mmcblk0p2 /mnt/root
  ```
- copy the Linux kernel and boot files to the boot partition:
  ```
  cd <path-to-kernel-and-boot-files>
  cp uImage devicetree.dtb BOOT.BIN /mnt/boot/
  ```
  The Cross-Compiling part explains how to build these files yourself.
- extract the server image:
  ```
  tar --strip-components=1 -C /mnt/root -xzpf ↩
  <path-to-image-file>/linaro-raring-server-<version>.tar.gz
  ```

- unmount SD-card:
  `sync`
  `umount /mnt/boot`
  `umount /mnt/root/`

3 System setup

3.1 First Boot

- insert SD-card into ZEDBoard, connect microUSB cable to USB-UART port and plug power connection in
- wait until blue LED turns on and connect to the ZEDBoard via serial console using putty (RAW connection on COM<X>, X is the connected board) or using screen (`screen /dev/USBttyACM<X> 115200`, see ls /dev/ttyACM*). Make sure you set the baude rate to 115200 bauds.
- if after ca. 1 min no blue LED comes up, try to connect anyway. Depending on the Boot-Image, no LED will light up.
- hit any key to interrupt autoboot
- load default configuration:
  `env default -a`
- change MAC-address to defined address to avoid MAC collisions:
  `setenv ethaddr 00:0a:35:01:01:3<X>`
  with X as your board number (e.g. `00:0a:35:01:01:38` for board #8)
- save changes:
  `saveenv`
- boot from SD-card:
  `run sdboot`
- after some seconds you should have a root shell, hit <RETURN> multiple times, if the console seems to got stuck.

3.2 Network interface

- determine device name (usually `eth0`, but sometimes it changes, especially when changing the MAC-address):
  `ip link`
  The first entry is irrelevant (it is the Loopback interface), the second one is the network interface and should look like `ethX` with `x` as some integer
- edit the network interface (NIC) configuration:
  `/etc/network/interfaces` and insert the following two lines:
  `auto ethX`
  `iface ethX inet dhcp`
- start interfaces:
  `ifup eth0`
- edit the hostname file: `/etc/hostname`

3.3 User configuration & SSH access

- set date:
  `ntpdate`
- update system:
  `apt-get update`
  `apt-get dist-upgrade`
  and install some useful tools:
  `apt-get install ntp ssh vim git build-essentials screen`
- set root password:
  `passwd`
- delete default user and set up new user:
  `deluser linaro`
  `adduser <USERNAME>`
  (The debian-specific tool `adduser` is interactive and more comfortable than the standard linux tool `useradd`)
- put user in groups:
  `usermod -aG <GROUP> <USERNAME>`
- delete old SSH keys and create new ones (for security reasons):
  `rm /etc/ssh/ssh_host_*`
  `dpkg-reconfigure openssh-server`
- optionally put your public SSH key to `/root/.ssh/authorized_keys`. The directory maybe have to be created first. Set the file permission accordingly:
  `chmod -r 600 /root/.ssh/`

4 Building Kernel drivers

If you need to build your own modules (device drivers), you need the linux kernel source. If you built the kernel yourself, you can use the sources that you already got. They have to be copied to `/lib/modules`. To build the modules, these steps are required:

- change to the directory containing the kernel sources and build the modules:
  `make modules`
- install them, so that the operating system is aware of them:
  `make modules_install`

5 Dynamic Reconfiguration

It is possible to reconfigure parts or the whole FPGA during runtime from Linux.

- Find the bitfile of your Vivado project, usually under:
  `<project>/<project>.runs/impl_1`

- Convert a Design bitfile into a `*.bin` file. The program promgen is part of the ISE toolchain:
  `promgen -b -w -p bin -data_width 32 ↩`
  `-u 0 design_1_wrapper.bit -o design_1_wrapper.bit.bin`
- Copy `*.bin` file to the Zynq over ethenet.
- Grant yourself access to the configuration device:
  `sudo chown root:admin /dev/xdevcfg;`
  `sudo chmod 660 /dev/xdevcfg`
- Load the bitstream:
  `cat design_1_wrapper.bin > /dev/xdevcfg`
- After a few milliseconds the new configuration has been loaded.

A Recreate Kernel and Boot-Image

This chapter explains how to cross-compiling the Linux kernel and how to create the build Boot-Image.

A.1 Toolchain

- get and install Xilinx Tools: SDK (tested with Xilinx Vivado 2013.2)
- install some packages needed: build-essentials, git
  `apt-get --no-install-recommends install build-essentials git`
- clone necessary Xilinx git trees: linux-xlnx, u-boot-xlnx, device-tree:
  `git clone git://github.com/Xilinx/<Repository Name>.git`
- grab a coffee, depending on the Server and your PC it can take a while
- before compiling, set some variables and source Xilinx paths (64 for 64bit Linux, else 32) and make sure these variables and paths are set for the following instructions:
  `export ARCH=arm`
  `export export CROSS_COMPILE=arm-xilinx-linux-gnueabi-`
  `sh <PATH-TO-VIVADO>/settings<64 OR 32>.sh`

A.2 Building Das U-Boot

- change to Xilinx U-Boot directory:
  `cd u-boot-xlnx`
- edit a configuration file: '`include/configs/zynq_common.h`' and search for the string '`sdboot`' (not '`rsa_sdboot`')
- delete the line containing the string '`${ramdisk_image}`'
- make sure the last line of this entry looks like this:
  `"bootm 0x3000000 - 0x2A00000; " \`
  It is important, that there is no address specified between the Kernel-Image (0x3000000) and the device tree (0x2A00000)
- now build Das U-Boot:
  `make -j4 zynq_zed`
  (specify the number of CPUs with parameter '-j', e.g. -j4 if you have 4 CPUs (virtual and physical))
- add the `tools` directory to your path:
  `export PATH=$PATH:<Path to current directory>/tools`

A.3 Building Linux Kernel

- change to Xilinx Linux Kernel and build kernel:
  `cd ../linux-xlnx`
  `make -j4 xilinx_zynq_defconfig`
  `make -j4 uImage LOADADDR=0x00008000`

- grab another coffee, this can also take a while
- the outputfile is located at `arch/arm/boot/` as `uImage`

A.4 Building the Device Tree

- stay in the Linux Kernel directory
  `make zynq-zed.dtb`
- the outputfile is located at `arch/arm/boot/dts/` as `zynq-zed.dtb`

A.5 Boot-Image

- generate a FSBL (First Stage Boot Loader) using SDK (`xsdk`)
  - start SDK
  - create new Application Project as a standalone application and create a new Board Support Package
- switch back to the terminal and create a new directory `bootimage` for building the Boot-Image
- copy `fsbl.elf`, `u-boot.elf` and `uImage.bin` to this new directory
  - `fsbl.elf` is located in your SDK project directory at `Debug`
  - `u-boot.elf` has just been compiled and is located in the U-Boot directory at `<u-boot-xlnx>/u-boot`. Just rename the file to `u-boot.elf`
  - `uImage.bin` is the Kernel that has been compiled in the previous step. This file also has to be renamed.
- create a new file `boot.bif` with the following content:

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

- build the Boot-Image:
  `bootgen -image boot.bif -o i BOOT.BIN`