Installing Embedded Linux on ZedBoard

Version 1.0

Clément Foucher (homepage)
Clement.Foucher@laas.fr

LAAS–CNRS
Laboratoire d'analyse et d'architecture des systèmes

September 3, 2015

Contents

1 Before starting **3**

1.1 Document purpose . . . 3
1.2 Disclaimer . . . 3
1.3 Tools revisions and OS . . . 3
1.4 Administrator permissions . . . 4
1.5 Conventions and directories . . . 4
1.6 Environment packages and libraries . . . 4
1.7 Downloading required sources . . . 5
1.8 Projects . . . 5
1.9 Reinitializing the SD card to factory state . . . 5
1.9.1 Restore partition scheme . . . 5
1.9.2 Restore files . . . 5

2 Preparing the environment **6**

2.1 Configuring the scripts . . . 6
2.2 Creating the project . . . 6

3 Hardware layout **7**

4 Generating the software **8**

4.1 Generating the kernel . . . 8
4.1.1 Default configuration using script . . . 8
4.1.2 Detailed process . . . 8
4.2 Generating the file system . . . 9
4.2.1 Default configuration using script . . . 9
4.2.2 Detailed process . . . 9
4.3 Generating the bootloader . . . 10
4.3.1 Ethernet configuration . . . 10
4.3.2 Default configuration using script . . . 10
4.3.3 Detailed process . . . 10
4.4 Generating the First Stage Boot Loader . . . 11
4.4.1 Detailed process . . . 11

Chapter 1

Before starting

Please read this chapter carefully before starting, as it contains valuable information that you'll require all along this document.

1.1 Document purpose

This document is a tutorial describing how to build an Embedded Linux system for use on a ZedBoard development board. Following this procedure, you'll obtain an Embedded Linux running on a persistent file system, which you can use as a base for your further developements. This tutorial describes every needed step, from scratch to a running system, and provides scripts to automate every step.

All needed tools are open-source and can be downloaded using the scripts provided with this tutorial, except for Xilinx Vivado which requires a license.

You should have obtained this document inside an archive containing the scripts and other files used in the procedure. If you don't have this archive, you can download it at this address: ....

TODO: Address of document

1.2 Disclaimer

The procedure depicted in this document is intended to help you build an Embedded Linux system whatever your knowledge of the Linux system is.

It should be relatively safe, but some steps require advanced manipulations on your host system, ZedBoard platform and related peripherals, including the SD card provided with ZedBoard.

The author or its institution cannot be held responsible for any harm caused to your host system, ZedBoard platform or any element manipulated by the following tutorial.

Moreover, the author can't be held responsible for english mispelling that is probably present in this document ;-). Plase inform me of any error using my e-mail on the front page.

1.3 Tools revisions and OS

The procedure described in this document has been conducted using Xilinx Vivado 2015.2 on Fedora 22 Workstation 64 bits. Some steps of the procedure are closely related to tools version,

so I cannot guarantee this tutorial will work using a different version of Vivado or any other tool used in this tutorial.

If your system does not run a native Linux OS, you can install for free a Linux virtual machine. The hardware generation steps, involving Xilinx tools, can be run using Windows in order to avoid running heavy tools in a virtual environment. Nevertheless, in such a case, you'll still neeed a Vivado installation on your virtual Linux to provide the ARM cross-compilation toolchain.

1.4 Administrator permissions

Some of the manipulations described in this document require root privileges. When so, we use the `sudo` command in order to get administrator permissions for the current command.

These commands require entering your password to work, and the current used must be registered in the sudo permissions list.

1.5 Conventions and directories

TODO: Reformuler cette section: on a peut-re plus bnesoin de quoi que ce soit à part basedir.

Texts beginning by `$` are shell commands to be typed in the console. Other texts written using `monospaced characters` are to be typed as it. Paths are displayed in `monospaced blue`. Paths ending with a `/` are directories (E.g., `<$BASEDIR>/scripts/`), while the others are files (E.g., `<$BASEDIR>/scripts/initialize_project.sh`).

The *host* system is the system used to generate the files, while the *target* system is the Embedded Linux system that will be generated.

Some specific directories will be used on the host system during the tutorial:

- `$BASEDIR` is the base working directory provided in the archive coming with this tutorial. The current file is located in `<$BASEDIR>/doc/`.
- `$MOUNTDIR` is a directory used to mount elements on the host system. While it can be any empty directory, `/mnt/` is usually chosen for this purpose. Anyway, any **empty** folder can be used as a mount directory.

In the current tutorial, we use the syntax `<$DIRECTORY>` to refer to paths defined above. These are to be replaced by the user when doing the manipulation using matching directory. E.g., if the base directory is located in `/home/user/ZedBoard/`, user has to understand occurrences of `<$BASEDIR>/scripts/` as `/home/user/ZedBoard/scripts/`.

1.6 Environment packages and libraries

Many packages are required on the host system to execute this procedure. Here are the main ones, but some other may be required depending on your configuration:

- gcc
- gcc-c++
- git

Before starting, type the following for each library in order to ensure all needed libraries are available on your system.

```
$ sudo dnf install <library>
```

TODO: provide a complete command line

1.7 Downloading required sources

Apart from the libraries required by the host, some sources are used in this procedure which are not provided in the archive (E.g. Linux kernel, boot loader, etc.). The first time a script is run, these elements will be downloaded. If you just want to download some or all sources in order to use them manually or for any other reason, I provide a dedicated script you can use as follows:

```
$ cd <$BASEDIR>/scripts
$ ./download_sources.sh
```

And follow the instructions from there.

Note that this script will not download already existing sources. If for any reason you want to force source re-download, please delete or move existing source before launching this script.

1.8 Projects

A project is a set of both hardware design and software environment generation. Projects are located in `<$BASEDIR>/projects/`. The base folder of project <project_name> will be `<$BASEDIR>/projects/<project_name>/`, and will be referred to as `<$PROJECT_ROOT>`.

1.9 Reinitializing the SD card to factory state

This section is only to be followed if you need to restore the SD card filesystem to its original state. It is not needed as part of this process, but it can be used to revert your SD card to its original state, as we modify its partition scheme during this procedure.

1.9.1 Restore partition scheme

Launch GParted. In the jumplist, select the entry matching your SD card (E.g., `/dev/sdc/`). On each existing partitions, right-click, and select Unmount. When all partitions are unmounted, select menu Device ⟩ Create Partition Table..., make sure MS-DOS type is selected and click on *Apply*.

Right-click on the empty space, select *New*, choose a *fat32* file system and select *Add*. Select *Apply All Operations* and validate using *Apply*. When finished, exit GParted.

1.9.2 Restore files

The card partition scheme is now ready, all you need now is to put back the original files on it. These files are contained in the archive provided at this address: `http://www.digilentinc.com/Data/Products/ZEDBOARD/ZedBoard_OOB_Design.zip`. Download it, expand it, and copy the content of folder `sd_image/` back to the card.

Your card is now ready as new.

Chapter 2

Preparing the environment

2.1 Configuring the scripts

First, file <$BASEDIR>/scripts/environment/project_root.sh must be edited. It contains a variable called **BASEDIR**, which must refer to the path of the base directory as defined in 1.5.

When done, also edit file <$BASEDIR>/scripts/environment/environment.sh. This file containes many variables that are to be used by other scripts. Each variable purpose is explained in the file, and must be set according to your host system configuration.

Please take some time to ensure ALL variables in these files are correct, or scripts in this document will not work. Incorrect configuration may even result in harming your system for commands with administrator permissions, so be very careful.

2.2 Creating the project

First, choose a project name (E.g., *test_project*). This name will have to be used in replacement of all <*project_name*> occurrences in this document.

Open a shell and type:

```
$ cd <$BASEDIR>/scripts
$ ./initialize_project.sh <project_name>
```

This will create the root folder of your project, as well as some sub-folders and files that will be used in the following steps.

Chapter 3

Hardware layout

TODO

Chapter 4

Generating the software

TODO: Edit intro

To boot, the board needs a .bin file, which is made of the bitstream, a boot loader and a first stage boot loader. In this chapter, we will generate the latter two.

4.1 Generating the kernel

The kernel contains the OS core, which is the very minimum to run on the board.

4.1.1 Default configuration using script

Open a shell and type:

```
$ cd <$BASEDIR>/scripts
$ ./generate_linux_kernel_default.sh <project_name>
```

When done, the kernel image is located in <$PROJECT_ROOT>/output/linux_kernel/uImage.

4.1.2 Detailed process

The procedure consists in getting the latest kernel version, and regressing to the correct branch, as follows:

```
$ git clone https://github.com/Xilinx/linux-xlnx.git linux_kernel
$ cd linux_kernel
$ git checkout xilinx-v2015.2.03
```

Then, copy the default device tree:

```
$ cp -f $PROJECT_ROOT/device_tree/zynq_devicetree.dts .
```

Finally, configure the kernel:

```
$ make distclean
$ make ARCH=arm xilinx_zynq_defconfig
$ make ARCH=arm xconfig
```

Then, set the configuration as you want. I personally set the following configuration to make sure system is very light by removing elements I don't need:

General setup	→ Initial RAM filesystem and RAM disk (initramfs/initrd) support		→ NO
Device drivers	→ Block devices	→ RAM block device support	→ NO
	→ Network device support	→ Wireless lan	→ NO
Networking support	→ Wireless	→ NO	
File systems	→ Network file system	→ NO	

Save, then build the image using the following command:

```
$ make ARCH=arm UIMAGE_LOADADDR=0x8000 uImage
```

TODO: COPIER LE RESULTAT

4.2 Generating the file system

4.2.1 Default configuration using script

Open a shell and type:

```
$ cd <$BASEDIR>/scripts
$ ./generate_file_system_default.sh <project_name>
```

When done, the generated file system is located in <$PROJECT_ROOT>/output/file_system/rootfs.ext4.

4.2.2 Detailed process

The procedure consists in downloading Buildroot, configuring it, and generating the file system.

First, download Buildroot using the following commands:

git clone $BUILDROOTURL$FSSOURCES cd $FSLSOURCES$gitcheckout$BUILDROOTGITTAB

```
$ git clone git://git.buildroot.net/buildroot buildroot
$ cd buildroot
$ git checkout 2015.08
```

Then, launch the configuration tool using the following command:

```
$ make xconfig
```

From there, set the configuration as you want. I personally set the following configuration for my needs:

TODO: séparer config nécessaire et config additionnelle

Target Architecture		→ ARM (little endian)	
Target Architecture Variant		→ cortex-A9	
Enable NEON SIMD extension support			→ YES
Build options	→ Show packages that are deprecated or obsolete		→ YES
	→ development files in target filesystem		→ YES
Toolchain	→ Kernel Headers	→ Linux 3.8.x kernel headers	
	→ GCC compiler Version	→ gcc 4.6.x	
	→ Use software floating point by default		→ NO
	→ Enable large file (files > 2 GB) support		→ YES
	→ Enable WCHAR support		→ YES
	→ Thread library implementation		→ linuxthreads (stable/old)
System configuration	→ System hostname:		→ zeboard-00
	→ Port to run a getty (login prompt) on:		→ ttyPS0
Package Selection for the target			
	→ Development tools	→ gcc	→ YES
	→ Libraries → JSON/XML	→ libxml2	→ YES
	→ Networking applications	→ openssh	→ YES
Filesystem images	→ ext2/3/4 root filesystem	→ YES	
		→ ext2/3/4 variant	→ ext4
	→ tar the root filesystem	→ NO	

Save, close, and type:

```
$ make
```

TODO: COPIER LE RESULTAT

4.3 Generating the bootloader

4.3.1 Ethernet configuration

If the board has to be used on a Ethernet network, you'll need to define a MAC address for the board. Open file `<$PROJECT_ROOT>/bootloader/zynq_common.h`, edit line 211 to introduce the MAC address desired for the board, and save the file.

If you have no use of the network, just ignore this step.

4.3.2 Default configuration using script

Open a shell and type:

```
$ cd <$BASEDIR>/scripts
$ ./generate_bootloader.sh <project_name>
```

4.3.3 Detailed process

TODO: Voir pour fichier zync_common.h

4.4 Generating the First Stage Boot Loader

The First Stage Boot Loader (FSBL) is in charge of the early board configuration.

Open the PlanAhead project. Select File » Export » Export Hardware for SDK. In the window, check both *Export Hardware* and *Launch SDK*, and validate using *OK*.

In XSDK, select File » New » Application Project. Set project name *FSBL*, make sure *OS Platform* is standalone and *Language* is C, then click Next. Select template *Zynq FSBL* and then Finish.

Right-click on the *FSBL* project, and select Build Configuration » Set Active » Release. Right-click again and select Clean Project.

4.4.1 Detailed process