

FPGA-ZCU Quick Start Tutorial

Document Version 1.0a

Tuo Li, Sri Parameswaran

School of CSE, University of New South Wales

`tuoli@unsw.edu.au`, `sri.parameswaran@unsw.edu.au`

December 6, 2018

About This Tutorial

This document will provide a step-by-step walkthrough with the fpga-zcu (the ZCU102 port of Rocket-chip) project, towards running operating system on rocket-chip System-on-Chip (SoC) on ZCU102 FPGA board. This tutorial assumes using SD card to boot and program the FPGA. This tutorial assumes that Vivado 2017.1 has been installed.

1 Getting Started

1. Clone the main repository using git command from github:
`git clone https://github.com/li3tuo4/fpga-zcu.git`
 Alternatively, you can directly download the zip file from github.
2. Enter the `fpga-zcu/zcu102` directory and use Makefile to initiate the submodules/sub-repositories:
`make init-submodules`
3. Enter `fpga-zcu/rocket-chip` directory and follow README to install the rocket-chip dependencies (submodules). Make sure riscv-linux-gnu is installed when building compiler. This is required for making riscv-linux.

2 Hardware Generation

1. Enter `fpga-zcu/zcu102` directory and use Makefile to compile chisel source code:
`make rocket`
2. Implement and build bitstream using Makefile:
`make bitstream`
3. Export the hardware from the vivado workspace (the directory created during synthesis and implementation):
`cp zcu102_rocketchip.ZynqConfig.runs/impl_1/rocketchip_wrapper.sysdef`
`soft_config/rocketchip_wrapper.hdf`
`cp zcu102_rocketchip.ZynqConfig.runs/impl_1/rocketchip_wrapper.bit`
`soft_config/rocketchip_wrapper.bit`
 These files are used by software generation.

3 Software Generation

In this step, we will create the software stacks for ARM and RISC-V. The ARM operating system will run on the host (ARM cores), which calls RISC-V via frontend server (fesvr). Executing frontend server with Berkeley boot loader (bbl), built with RISC-V Linux kernel, on ARM, will boot up RISC-V core to run RISC-V Linux operating system.

3.1 ARM

1. Install petalinux, which creates software stacks for ARM. Follow the “Installation Steps” on Page 11 in: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1144-petalinux-tools-reference-guide.pdf

2. Download board support package (bsp) file of ZCU102 from: <https://www.xilinx.com/member/forms/download/xef.html?filename=xilinx-zcu102-v2017.1-final.bsp>. Xilinx account is required, which can be registered straightforwardly. Put the downloaded file `xilinx-zcu102-v2017.1-final.bsp` into `zcu102/soft_config` directory.
3. Export PETALINUX environment variable and set up petalinux by:
`export PETALINUX=/path/to/petalinux && make pl_setup`
4. Enter `fpga-zcu/zcu102`. Use Makefile to automatically create and build petalinux project for ARM Linux image:
`make kernel_image`
 Makefile will automatically create a petalinux project named `petalinux_proj`, in `fpga-zcu/zcu102`. The `BOOT.BIN` and `image.ub` in `petalinux_proj/images/linux` are the image files created. `initramfs.cpio.gz` is the root file system.
5. Copy these three files to `zcu102/fpga-images-zcu102` directory. Note this step will replace the initial image files in this directory. Then, use Makefile to open, modify, and close the root filesystem:
`make rootfs-open`
`make rootfs-close`

3.2 RISC-V

1. Obtain `freedom-u-sdk` from <https://github.com/sifive/freedom-u-sdk> and follow README to set up.
2. Copy the configuration file from `soft_config` to the work directory of RISC-V Linux and in the work directory of RISC-V Linux execute build:
`cp /path/to/soft_config/Linux_config /path/to/freedom-u-sdk/work/linux/.config`
`./build`
 The kernel image `vmlinux` will be generated this directory.
3. Follow the `riscv-tools` README to create Berkeley boot loader (bbl) with RISC-V Linux included: `cd <riscv-pk>/build`
`rm -rf *`
`../configure --prefix=$RISCV --host=riscv64-unknown-linux-gnu`
`--with-payload=<riscv-linux>/vmlinux`
`make`
`make install`
4. Use Makefile to create `fesvr-zynq` binary for ZCU102:
`make fesvr-zynq`
5. Put `fesvr-zynq`, `libfesvr.so` and `bbl` into the root filesystem by using the commands in Step 4 in Section 3.1.

4 FPGA Board Setup and SD Card Preparation

1. Prepare the board following Xilinx quick start guide in https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/xtp426-zcu102-quickstart.pdf and Chapter 2: Board Setup and Configuration in Xilinx UG1182 in https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf. Note Switch SW6 must be configured to “off, off, off, on” for allowing SD card boot up.
2. Prepare SD card following README in the sub-repository `fpga-images-zcu102` in <https://github.com/li3tuo4/fpga-images-zcu102>.
3. Enter `/path/to/zcu102` directory and use Makefile to load SD card:

```
make load-sd
```

 Now it's ready to run the system on FPGA.

5 FPGA Emulation

Make sure USB cable's driver is installed properly on the host PC. Xilinx UG344 might be helpful: https://www.xilinx.com/support/documentation/user_guides/ug344.pdf

1. Set up and open minicom or your preferred terminal: `minicom -D /dev/ttyUSB0`
 Make sure the serial port in minicom is configured to “115200 8N1”, which means “speed = 115200, parity bit = none, data bits = 8”.
2. Insert the SD card and power on the FPGA board. The minicom terminal will show the ARM Linux boot up output. By default, ARM Linux's login username and password are both “root”.
3. After login, execute `fesvr` with `bbl` as argument to wake up RISC-V and let it boot up Linux:

```
LD_LIBRARY_PATH=./ ./fesvr-zynq bbl
```

 By default, the login username and password of RISC-V Linux are “root” and “sifive”.

Hint: If you have ethernet connection, which is already available in the default system build, using SCP to transfer files such as `fesvr` and `bbl` directly from Host PC to FPGA after ARM Linux login can save the time in opening, modifying and closing root filesystem.