FPGA-ZCU Quick Start Tutorial

Document Version 1.0d

Tuo Li, Sri Parameswaran
School of CSE, University of New South Wales
tuoli@unsw.edu.au, sri.parameswaran@unsw.edu.au

March 22, 2019

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 and SDK 2018.2 have 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/zcu102_rocketchip_ZynqConfig.runs/impl_1/
       rocketchip_wrapper.sysdef soft_config/rocketchip_wrapper.hdf
   cp zcu102_rocketchip_ZynqConfig/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. This tutorial uses `linux_u500vc707devkit_confg` branch (commit `db77bd7a8779d776d3f67b8a76ab5973e93604e6`).

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/config_freedom /path/to/freedom-u-sdk/work/linux/.config`
   `./build`

   In the latest version of freedom SDK, build is extracted into simple Makefile, in this case, type:
   `make -jN ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu vmlinux`
   instead. The kernel image `vmlinux` will be generated in 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. Make sure Xilinx SDK is installed and has been added to the PATH to successfully generate the binary. 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.

6 Possible Build Errors and Solutions

6.1 Error compiling sbt component 'compiler-interface'

Example error message after `make rocket`:

```
sajid2@sajid2-HP-Compaq-Elite-8300-SFF:~/fpga-zynq/zc706$ make rocket
mkdir -p /home/sajid2/fpga-zynq/common/build
cd /home/sajid2/fpga-zynq/common && java -Xmx2G -Xss8M -XX:MaxPermSize=256M -jar /home/
    sajid2/fpga-zynq/rocket-chip/sbt-launch.jar "run /home/sajid2/fpga-zynq/common/build
    zynq Top zynq ZynqFPGAConfig"
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=256M; support was
    removed in 8.0
[ERROR] Failed to construct terminal; falling back to unsupported
java.lang.NumberFormatException: For input string: "0x100"
       at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
.
.
.
.
.

[info] Resolving org.scala-sbt#interface;0.13.16 ...
[error] (compile:compileIncremental) Error compiling sbt component 'compiler-interface'
Project loading failed: (r)etry, (q)uit, (l)ast, or (i)gnore? q
```

Solution:

1. Build/compile the chisel and verilator according to the instruction given here https://github.com/freechipsproject/chisel3 including `publish_local` and adding librarydependencies into build.sbt file which locates into rocketchip directory. According to the following instructions:

   (a) To publish your version of Chisel to the local Ivy (sbt's dependency manager) repository, run: `sbt publishLocal`

   (b) In order to have your projects use this version of Chisel, you should update the library-Dependencies setting in your project's build.sbt file to:

   ```
   libraryDependencies += "edu.berkeley.cs" %% "chisel3" % "3.2-SNAPSHOT"
   ```

2. Install Java 8.

   (a) Downladed the java version 8 from https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

   (b) Extract to the folder

   (c) Add the extracted folder bin path into path by

   ```
   export ........./bin  :$PATH
   ```

   Check the java version before adding the path and after adding the path to confirm it. Use command `which java`:

   ```
   sajid2@sajid2-HP-Compaq-Elite-8300-SFF:~/rc-fpga-zcu/rocket-chip/riscv-tools$
       which java
   /usr/bin/java
   sajid2@sajid2-HP-Compaq-Elite-8300-SFF:~/rc-fpga-zcu/rocket-chip/riscv-tools$
       export PATH=/home/sajid2/jdk-8u191-linux-x64/jdk1.8.0_191/bin:$PATH
   ```

```
sajid2@sajid2-HP-Compaq-Elite-8300-SFF:~/rc-fpga-zcu/rocket-chip/riscv-tools$
    which java
/home/sajid2/jdk-8u191-linux-x64/jdk1.8.0_191/bin/java
```

3. Now make rocket. Finally it is working now and successfully completing

6.2 tcmalloc failure during vivado synthesis

Error message:

```
--------------------------------------------------------------------------------
Start Technology Mapping
--------------------------------------------------------------------------------
src/central_freelist.cc:333] tcmalloc: allocation failed 16384
TclStackFree: incorrect freePtr. Call out of sequence?
[Tue Jan 15 17:23:56 2019] synth_1 finished
wait_on_run: Time (s): cpu = 00:00:00.33 ; elapsed = 00:20:41 . Memory (MB): peak =
    1493.863 ; gain = 0.000 ; free physical = 1893 ; free virtual = 3117
# launch_runs impl_1 -to_step write_bitstream
ERROR: [Common 17-70] Application Exception: Failed to launch run 'impl_1' due to
    failures in the following run(s):
synth_1
These failed run(s) need to be reset prior to launching 'impl_1' again.
```

Solution:
It's caused by insufficient memory in the host machine. According to https://www.xilinx.com/products/design-tools/vivado/memory.html, synthesizing for xczu9eg (zcu102) by vivado requires 10 to 14 GB memory.