About Pulpino

Aamir Sultan,
Associate Design Engineer,
Lampromellon.

November, 2020

In this document we will see the finding and knowledge obtained during the study of Pulpino micro-controller in this week.

1. What is Pulpino?

Pulpino is a microcontroller developed at ETH Zurich university[1]. This microcontroller is using single-core which can configured for RISCY or Zero-Riscy core.

1.1 RISCY properties

Properties of the RISCY core are given as follows:

- Pipeline stages 4, In order, IPC nearly 1.
- Support for RV32IMC, can also be configure for the RV32F.
- Other supported ISA extensions.
    - ⋆ hardware loops
    - ⋆ post-incrementing load and store instructions
    - ⋆ bit-manipulation instructions
    - ⋆ MAC operations
    - ⋆ support fixed-point operations
    - ⋆ packed-SIMD instructions
    - ⋆ dot product

1.2 Zero-Riscy Properties

Zero-Riscy is an In order, single issue with 2 pipeline stages. It supports RV32IMCE of RISC-V architecture extensions. Other properties of the Zero-Riscy are as folows:

- ⋆ Ultra Low Power.
- ⋆ Ultra Low Area.
- ⋆ Switch to Ultra Low power mode while IDLE.
- ⋆ Wake up unit in-case of interrupt or event.

2. Building the Pulpino

Following is the command list that is used for the building of the core form the Git repository[1].

1. One can check for the packages using the following command which looks for packages in *PyPI* with yaml in the short description. That reveals various packages, including *PyYaml*, *yamltools*, and *PySyck*, among others (Note that *PySyck* docs recommend using *PyYaml*, since *syck* is out of date). Now you know a specific package name, you can install it.

```
$ pip search yaml
```

2. If the pip is not installed you can install it via the following code.

```
$ sudo apt install python-pip
```

3. First of all install the pyyaml for python2 incase if it is used in the building process of the SoC.

```
$ pip install pyyaml
```

4. In the second step install the yaml library with pyhton3 using the following command.

```
$ pip3 install pyyaml
```

5. Make a build folder inside the sw folder of the pulpino main directory. We will use it for keeping the settings of the core that we want to generate. Following command will create the build folder.

```
$ mkdir ./sw/build
```

6. For copying the configuration file we use the copy command as follows.

```
$ cp ./sw/cmake_configure.zeroriscy.gcc.sh ./sw/build
```

   the configuration files are placed in sw folder. The other configuration files can be used for building also. One has to repeat the step 6 and change the *cmake_configure.zeroriscy.gcc.sh* by other configuration files that are given as follows.

   - cmake_configure.zeroriscy.gcc.sh
   - cmake_configure.riscvfloat.gcc.sh
   - cmake_configure.riscv.gcc.sh
   - cmake_configure.microriscy.gcc.sh

7. We need to install a special makefile utility by the name of Cmake. CMake uses a simple platform and compiler-independent configuration files to control the software compilation process. We the follow the following commands[2]:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo snap install cmake --classic
```

   Check the version of the cmake by using the the follwoing command if it is correctly installed then it will show the version of the Cmake installed.

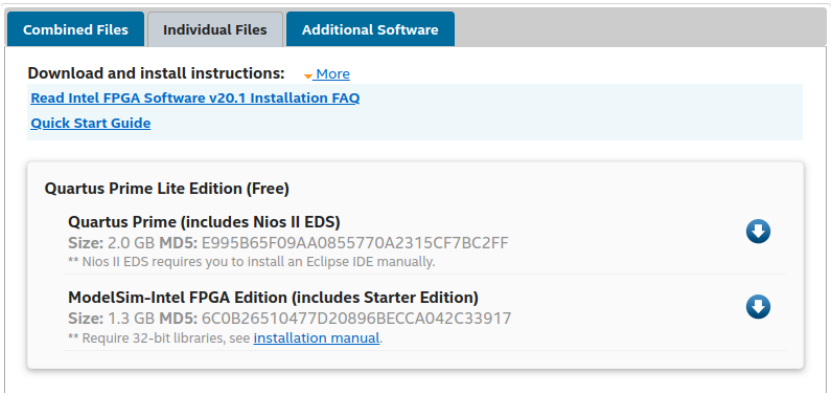

Figure 1: Download Options

```
$ cmake --version
```

8. We also need the modelsim for the simulation of the SoC [3]. However the modelsim should be installed on the system but incase if is not installed you can send a request for download at `https://fpgasoftware.intel.com/20.1/?edition=lite&platform=linux`. On this page goto individual files and click on the down arrow for downloading the Modelsim Starter Pack, Figure 1.

   Note : If ModelSim is already installed then you skip to step number 15.

   Once the ModelSim is downloaded it will in the form of *.run* file. To install goto the download location of the modelsim *.run* file and run the following commands.

```
$ chmod +x ModelSimSetup-*.run
```

9. Next we will install the ModelSim by using the following commands.

```
$ ./ModelSimSetup-*.run
```

10. Change your directory to *"ModelSim-Installation-Directory/intelFPGA/20.*/modelsim_ase/linuxaloem"* using the command.

```
$ cd ModelSim-Installation-
Directory/intelFPGA/20.*/modelsim_ase/linuxaloem
```

11. Type the following command to run modelsim.

```
$ cd ./vsim
```

12. if the ModelSim is executed its fine otherwise you will have add few libraries. if the error is given as *./vsim: No such file or directory* then use following commands. *Note:* In case the model-sim is invoked successfully you can skip to step number 15.

13. Type the following command to run modelsim.

```
$ sudo dpkg --add-architecture i386
$ sudo apt-get update
$ sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
$ sudo apt-get install lib32z1 lib32ncurses5 lib32bz2-1.0
```

14. try again if executed otherwise it will possibly give the following *error-libXft.so.2: cannot open shared object file.*

```
$ sudo apt-get install libxft2 libxft2:i386 lib32ncurses5
```

Another error that will pop *libXext.so.6: cannot open shared object file. libXext.so.6 not found.* Use following command.

```
$ sudo apt install libxext6
$ sudo apt install libxext6:i386
```

Hopefully this will solve the problem.

15. We need Verillator for evaluation. We can install the Verillator using following command.

```
$ apt-get install verilator
```

16. We need to clone the RISC-V toolchain for the compilation and running of the SoC. We will clone the toolchain from pulp platform using the following commands, [4].

```
$ git clone --recursive
https://github.com/pulp-platform/pulp-riscv-gnu-toolchain
```

17. Setting and building the tool-chain using command.

```
$ sudo apt-get install autoconf automake autotools-dev curl
libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison
flex texinfo gperf libtool patchutils bc zlib1g-dev
```

18. To build the Newlib cross-compiler for all pulp variants, pick an install path. If you choose, say, /opt/riscv, then add /opt/riscv/bin to your PATH now. Then, simply run the following command in the cloned directory:

```
$ ./configure --prefix=/opt/riscv –with-arch=rv32gc
–with-cmodel=medlow –enable-multilib
$ sudo make
```

19. To set the path to the binaries use following command.

```
$ export PATH=$PATH:/opt/riscv/bin
```

20. To build the Newlib cross-compiler, pick an install path. If you choose, say, /opt/riscv, then add /opt/riscv/bin to your PATH now. Then, simply run the following command:

```
$ ./configure –prefix=/opt/riscv
$ sudo make
```

21. Setting the Environment paths.

```
$ export PULP_RISCV_GCC_TOOLCHAIN=/opt/riscv/bin
```

22. Change the directory to the */pulpino/sw/build* and use the following command.

```
$ cmake ..
```

It will create the *makefile* and other files for further executions.It will create the *makefile* and other files for further executions.

23. If it gives error then try installing the essentials for correct working using the following command.

```
$ sudo apt-get update
$sudo apt-get install -y build-essential
```

References

[1] E. Zurich, "Pulpino." `https://github.com/pulp-platform/pulpino`, 2016.

[2] fosslinux, "Cmake." `https://www.fosslinux.com/38392/how-to-install-cmake-on-ubuntu.htm`, 2020.

[3] H. H. Hung, "Modelsim installation." `https://yoloh3.github.io/linux/2016/12/24/install-modelsim-in-linux/`, 2016.

[4] E. Zurich, "pulp-platform." `https://github.com/pulp-platform`, 2016.