GSoC 2019: Cycle-accurate Verilog Design Simulation Integration

Bowen Hu

Self introduction

1 Basic information

Name: Bowen Hu

University: Fudan University, Shanghai, China

Email: bowenhu19@gmail.com

Github: https://github.com/B0WEN-HU/

Contacts: Email, Skype, Google Hangouts

Residence: Shanghai, China (UTC+8)

2 Background

My name is Bowen Hu. I am a first-grade graduate student majoring in microelectronics at Fudan University, Shanghai, China. My undergraduate major is electronic information science and technology.

The main work of my major now, is to design digital integrated circuits with Verilog and many other tools. By far, I've used several programming languages including C++, Python, Verilog and MATLAB. Even though I have not used Verilator before, I do have experience of how to do Verilog simulation with ModelSim and Vivado, which are both the simulation tools commonly used in the industry. I will not get credits for the GSoC project.

I never been involved in any open source projects before. But I am still interested in SDR, even though I chose IC design as my major.

I have uploaded some of the code I have written. Please check the links below.

- C++

  https://github.com/B0WEN-HU/Plugin-demo

  https://github.com/B0WEN-HU/Fractal_Multithreading_demo

  https://github.com/B0WEN-HU/Channel-coding-simulations

  https://github.com/B0WEN-HU/Sparse_matrix_addition

  https://github.com/B0WEN-HU/Finite-field-systolic-array-multiplier/tree/master/Simulation

  https://github.com/B0WEN-HU/64-point-FFT/tree/master/Simulation

- Python

  https://github.com/B0WEN-HU/Cryptology_web_demo

- Verilog

  https://github.com/B0WEN-HU/Finite-field-systolic-array-multiplier

  https://github.com/B0WEN-HU/64-point-FFT

I am currently fluent in Chinese and English.

I heard about GNU Radio about a year ago, when I was an undergraduate. I built a wireless communication demo (it still needs clock wire though, so technically it is not *wireless* ;).) with the flowgraph of GNU Radio. I want to explore the potential of hardware in SDR system. So, it is best to stay in close contact with GNU Radio community, even if the GSoC project is over. Actually, I think this is an excellent opportunity to learn more about GNU Radio, by contributing to it.

Topic background

1 Introduction

Hardware accelerators are necessary or at least desirable in many SDR systems. GNU Radio provides an open and free platform for designing real time SDR system. This proposal elaborates the architecture of Verilog design simulation integration, utilizing the existing tool, Verilator.

2 Reasons for integrating hardware simulation

In order to achieve a fast and efficient real-time SDR system, Hardware accelerators are commonly used. These accelerators generally are designed with HDL, like Verilog, and implemented on FPGA, a programmable hardware device. The typical workflow to design a FPGA accelerator is shown in Figure 1.

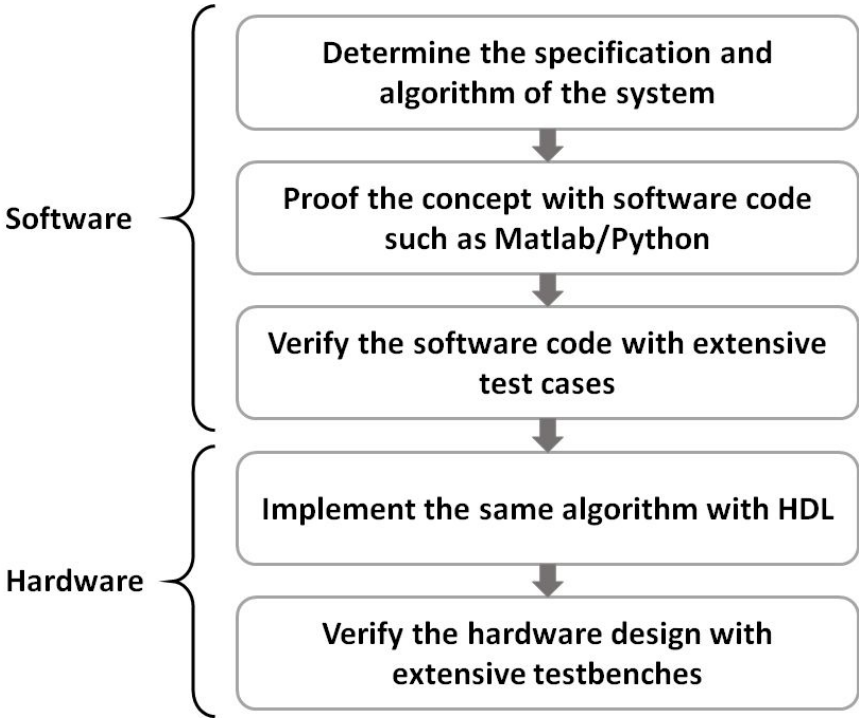

Figure 1. Typical workflow of hardware accelerator design

This development workflow is complicated and hard to cooperate with the existing SDR system. However, with Verilator, Verilog code could be simulated and tested within GNU Radio flow, which could save a lot of effort writing test cases, greatly simplify the development process, and test the hardware design in a real system scenario. Verilator is a Verilog simulator. Verilator can compile synthesizable Verilog code into C++ code, which is especially well suited to generate executable hardware model for software environment.

3 Architecture of integrated Verilog design simulation

Considering the limitation that synthesizable Verilog codes only accept sets of logic symbols(Boolean vectors) as input or output, just like real hardware devices do, various data types in GNU Radio have to be converted into the suitable type for Verilog code.

The executable hardware model generated by Verilator is cycle-accurate. Thus, there have to be a control module which generates control signals that make the executable hardware model works as designed.

Because it is unreasonable to compile and restart the GNU Radio after the Verilog module was changed, the whole process should work at runtime.

In order to solve these problems, this project should be divided into the following parts. The schematic diagram is shown in Figure 2. There are more details in the next section.

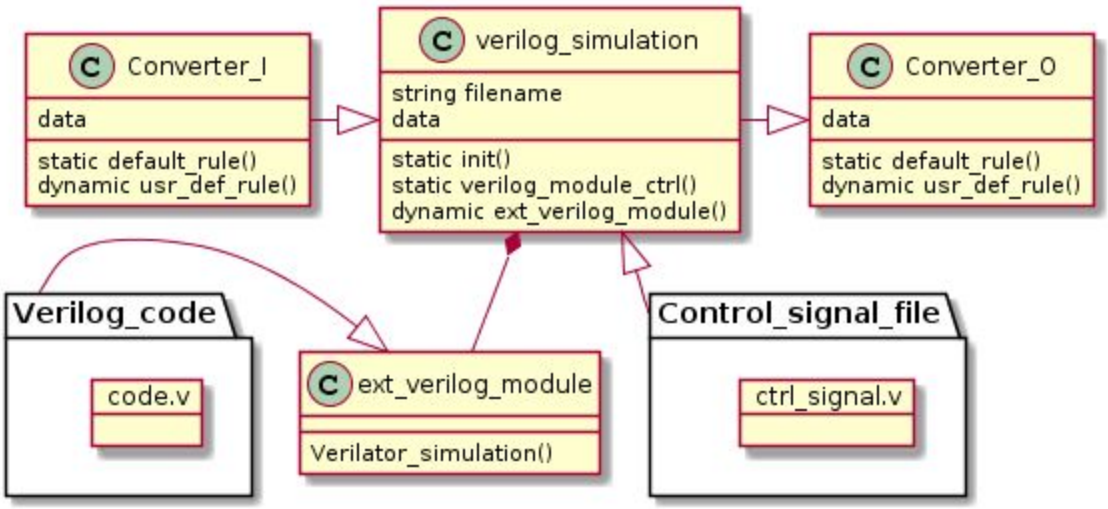

Figure 2. Schematic diagram of the architecture

- Converter for executable hardware model: It is important to build the suitable data type converter for Verilog to work with other GNU Radio blocks.
- Verilog/C++ parsing tool: Because the Verilog code comes from an external file, it is necessary to parse the Verilog code or C++ code generated by Verilator to get the ports in Verilog.
- Control signal module: The control signal module should give right control signal, clock or reset signal for example, to the Verilog module and handle the cycle of input and output.
- Plugin structure for external Verilog code: The external Verilog code should be compiled and executed at runtime. I have written a demo of plugin structure, please check the code here(https://github.com/B0WEN-HU/Plugin-demo).

The Deliverables

Work deliverables and details are listed below.

- Deliverables
    - The functional code that can make external Verilog code compiled and executed at run time, convert the data type to make the Verilog module work with the GNU Radio properly, and give the right timing control signal to control the Verilog module as designed.
    - Examples and software test cases for fundamental blocks, for example, a FIFO and an integer squarer.
    - Documentations for the code.

- Details
    - Converter: The converter module will convert the data type suitable for the Verilog module. It will only be implemented for several default rules of data type conversion, like Integer 32 and Float 32. Users might want to specify the rules of data conversion in case that the default rules won't work properly. This part may be implemented by the very same plugin structure as the main verilog simulation module.
    - Control signal module: The control signal module will generate clock, reset and any other essential signals to control the Verilog module. moreover, it will handle the input and output with correct timing, make the Verilog module works as intended. However, Due to cycle-accurate Verilog

simulation by Verilator, the timing of input and output may vary in different modules. For example, there might be uncertain number of cycles before the output data is valid(However, the author of the module should know about it).Or, some modules would have to take its time to do the calculation, and won't respond to the input data for several cycles, which means if we put the data to the input port during these cycles, the data would just be ignored. This information should be informed by user through some mechanisms. As Figure 2 shows, a control signal file might be a practical way to tell the program how to give the proper signals to the module.
    - Plugin structure: The plugin structure makes the external Verilog code be compiled and executed at runtime. It should be implemented through a dynamic link library (shared library) with fixed interface, and called at runtime.

License

The code submitted will be GPLv3 licensed. Cyberspectrum is the best spectrum.

Schedule

I plan to spend 40 hours a week writing code during GSoC period. I might have about one week off in mid-June in order to prepare for the final exam. However, I will make sure that I can catch up.

Timeline

- **May 6 - May 26** - Learn related knowledge and Discuss with the mentor.
- **May 27 - May 31** - Coding - Build the structure of the whole module.
- **June 1 - June 7** - Coding - Build the default rules and implementation of converter.
- **June 8 - June 14** - Coding - Build the parsing tool which can get the ports in Verilog.
- **June 15 - June 21** - Coding - QA test for the converter and parsing tool.
- **June 22 - June 28** - Submit Phase 1 Evaluation
- **June 28 - July 5** - Coding - Build the control module.
- **July 6 - July 12** - Coding - Build some simple Verilog test cases.

- **July 13 - July 19** - Coding - Combine all modules, and test it.
- **July 20 - July 26** - Submit Phase 2 Evaluation
- **July 27 - August 2** - Integrate the module to GNU Radio.
- **August 3 - August 9** - Coding - Add more features and improve the code.
- **August 10- August 16** - Add documentations.
- **August 17- August 26** - Submit final work product and final evaluation

References

https://wiki.gnuradio.org/index.php/GSoCIdeas

https://wiki.gnuradio.org/index.php/Guided_Tutorial_GNU_Radio_in_C%2B%2B

https://www.veripool.org/projects/verilator/wiki/Manual-verilator