Python Based Verilog Code Generator

Akshat Mangal | Gaurav Sonkusle | Mohamed Shamir | Mohmmad Aslam

Writing hardware code is time-consuming. Especially when you don’t have much experience working with hardware codes. We wish to eliminate this gap between the experienced and the non-experienced. Our idea was to make some user-friendly interface for verilog code generation, so even if someone is very new to the domain of hardware language programming, they can simply generate the codes for the modules & testbenches as per their requirements by just providing specific input to our program. Our project aims to generate Verilog code for sequential circuit modules and the respective testbenches using python programming language. We generated the code for the counters, registers, flip flops, floating point arithmetic operation unit modules and we also implemented FSM modules which convert input state table into verilog code for the moore and mealy implementations.

Most challenging part of our project was parametrizing and generalizing circuits and making it functional for n-bit operations. To verify the generated modules and testbenches we simulated it in the EDAplayground and checked the waveform. The output Verilog code can be simulated in EDAplayground and deployed on FPGA boards to test and evaluate the effectiveness of the Verilog code generated. We implemented the following modules and are able to generate the verilog code for the same according to the given user inputs.

1. Counters -- Up counters, Down counters, Ring counters, Johnson counters
2. Registers -- SISO, SIPO, PISO, Shift left/right , Circular left/right
3. Flip Flops -- JK, T, D, SR
4. Floating point arithmetic operations -- Adder, Subtractor, Multiplier, Divider.
5. FSM : State Tables into Verilog Code* -- Moore and Mealy Implementation

Counters: In counters we are taking number of bits as an input from the user and what kind of implementation they need, i.e., behavioural or structural. Then we are using Jinja2 templates and flask in backend to generate the code as the user requirement and display on our website. We are also providing them with a basic testbench so that they can check their results on either Vivado or Edaplayground. In counters we also implemented the **Custom Counter** in which we gave users more options like choosing start and end points of the counter (like BCD counter, 2-18 counter).

*FSM module has been implemented as bonus to the originally proposed work for the project

Registers: In registers also we are taking number of bits as an input from users and for shift & circular registers one additional input feature is, what kind of shift/circular movement they want left/right. After that using our backend code, we generate the appropriate verilog code as per user requirement with a basic testbench template.

FlipFlops: In flipflops we don't require any input from the user because they have fix input/outputs.

Floating Point Arithmetic Operations: Modules are made for addition, subtraction, multiplication and division operations. The user can give input in the decimal form. Inside the program, the input number would be converted into the IEEE754 representation. And the code outputs the result in IEEE754 format for floating numbers.

FSM : State Tables into Verilog Code: A module where the user can give a state table as input and can get the verilog code as output. The verilog code can be generated for both Moore and Mealy Implementation for N different states.

The web application can be accessed at https://python-verilog-gen.herokuapp.com/. The github repository for the same is available at https://github.com/mshamir11/python-verilog-code-gen.

The main challenge involved in the project was the parametrization of each module. The program should generate verilog code on its own for various input parameters. We approached the problem by writing verilog code for the respective module. We identified various changes in the code for different user inputs and needs. Using Jinja2 Templating Language, we generated verilog code as per the user input to the module. We used the EDA playground for the design and simulation of each verilog module. We used flask microframework, for displaying our work.
HTML, CSS and JavaScript are used for the development of the webpage and finally Jinja2 Templating Language is used for the parametrization of each verilog module. The web app developed is deployed on heroku platform for the demo purpose.

Future Work:
The project can be extended to many other modules and can be used as a verilog library. More flexibility can be provided to the users on deciding what components to use or alternate methods in the verilog implementation.