# Ataraxia – A Web-Enabled FPGA-Based Verilog Simulator

Niraj Manikantadas Menon

ECE 49600

Purdue University

**Author Note**

Correspondence regarding this article must be addressed to Niraj Manikantadas Menon, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907.

University email: menon18@purdue.edu

Personal email: nirajmmenon@gmail.com

**Table of Contents**

**Abstract**

This paper details the design and implementation of software intended to provide an easy, free and accessible way to simulate and synthesize Verilog through a website with near-zero client-side setup time. With Verilog code intended to model a given hardware design for an example breakout board on an ice40HX8K FPGA, a user may perform manual verification of the design by interaction with input buttons on said example breakout board. It also has support for indicating various errors on corresponding lines of code, which includes basic syntax and synthesis errors. The simulator is different from other alternatives like EDA Playground in that it adds a virtual breakout board theoretically connected to the FPGA and adds customized errors in place of the generic misunderstood warnings generated by simulation tools like CVC and synthesis tools like Yosys. With these differences, it provides a far more direct and intuitive interface for users new to digital design with Verilog. By synthesizing their code and simulating the resulting gate-level structural code, a user could get results extremely close to what they would get on the real FPGA and viewing them on a breakout board more natural for new users than having to analyze waveforms.

This paper is also intended to serve as an end-of-semester report on the work done on the software as part of the ECE 49600 requirements. The instructor in charge was Rick, and the course was worth 3 credit hours.

*Keywords*: Verilog, SystemVerilog, FPGA, CPLD, Synthesis, Simulation, CVC, Yosys

# Ataraxia – A Web-Enabled FPGA-Based Verilog Simulator

Students in ECE 270 at Purdue University were finding it quite difficult to access lab computers to be able to test their Verilog designs on the lab evaluation board (an ispMACH 4256ZE CPLD with an attached breakout board consisting of DIP switches, pushbuttons and various LEDs and character displays) due to the sheer number of students already waiting for spots during designated open lab hours and therefore the lack of available computers. This was unfair to the students whose Verilog designs were due early on in the week and were only able to use the board effectively for a few hours, or sometimes not at all. This necessitated the development of **software that would let students be able to closely emulate the experience of writing and synthesizing Verilog for the physical board in lab**. I did this project with the valuable guidance of Rick, my instructor from ECE 362 who taught ECE 270 in Fall 2019.

## Simulator design

### Establishing requirements

It was established that developing such software should satisfy the following criteria:

- It should be **intuitive**, since the core idea was to offer students a second way to work on their Verilog designs in a way that was identical to their lab experience, but easier.
- It should be **interfaceable**, so that future extensions like a SystemVerilog testbench could be used to test hundreds of students' code at once.
- It should be **extendable,** allowing students who finish the course and are interested in working on the simulator, to implement their own additions and extensions to the simulator without too much familiarity of the original backend code.

**Setting up the framework**

Over the course of the last few years, increasingly more open source software has been developed to enable access to Verilog simulation and synthesis tools for free to the public. Some of these include IcarusVerilog, CVC [4] and Yosys [5]. It was found that with a good parser of the potentially vague error messages produced by the compilers, one could build an excellent front end interface for these compilers that exhibited simplicity and ease of use without sacrificing the essence of said compilers.

After extensive testing with all of the above options, **CVC** was found to be a great option for basic Verilog simulation. In addition to having sufficient detail in its error reporting as opposed to the spurious error reporting offered by IcarusVerilog, CVC has support for timing analysis, which could be added to the simulator in order to provide more realistic simulations.

For synthesis, the **Yosys Open SYnthesis Suite (Yosys)** was chosen. This choice was made mainly due to the fact that there were no other viable open source solutions for Verilog synthesis on ice40 FPGAs.

For the framework that would couple these tools together, **node.js** was chosen as a viable option. Node.js is a framework that lets a user write JavaScript to be executed on the server. In addition to the usual features of JavaScript as a programming language, node.js adds support for program execution and file system support on the server, which I could utilize to do things like run the simulation software for received code from a student via a webpage. A powerful application that is used prominently in industry [1][2][3], node.js is very lightweight and has great support for some of the capability required server-side, such as implementing a WebSocket server and running command line programs, including CVC and Yosys.

**Determining how to deploy**

At first, the idea of having standalone software that could be installed on to student's own computers was discussed as a possible option, but it became clear soon enough that such an option would be impractical. Having students install desktop software to run simulations would eliminate any latency issues since the software and required tools would work on the student's own laptop, but I would have to face an endless barrage of patch requests, in addition to maintaining cross-platform compatibility and issuing a new version every time something broke in the software, which was expected with such a large number of students who were likely to use the software in ways that were never intended, and therefore break it.

Therefore, I decided to deploy the simulator as a website. The fastest and easiest way to deploy software has been to deploy it online, which requires basic knowledge of HTML, CSS and JavaScript at the very least. Deploying the simulator as a website made it extremely easy to update things on the fly and allowed us to focus more on how to integrate the synthesis and simulation tools more easily.

**Intended Design**

In June, I determined that the FPGA simulator should have the following workflow:

- Upon visiting the site, a student will be prompted to enter their credentials. Upon successful authentication, the page will load. Upon failure, an "Access denied" page will appear.
- The page will have a text box for entry of code, an interactive SVG that will look and behave like the actual FPGA breakout board given in lab, with buttons that can send inputs to a running simulation and animated circles/rectangles that will emulate the seven segment displays and LEDs on the actual board.

- The student will enter their code into the box, and type Simulate.
- A WebSocket (a persistent HTTP connection) is established back to the server, and on establishment, code and the current inputs from the pushbuttons (set by toggling them on the interactive SVG for the FPGA) is sent back through the WebSocket to the server.
- The submission will first undergo edge case checks like no code, no Verilog, and/or missing a top module. These are easily checked with Regex and will immediately report errors to the webpage, which avoids wasting CPU time on the server since there is no need to simulate or synthesize code that will definitely not run if it fails these cases.
- If it passes these cases, the code is quickly checked for syntax errors with CVC. If CVC returns errors, they are sent back to the webpage, where they will be displayed by line number. If not, the process continues.
- The code is then synthesized with Yosys, which produces an internal structural design that can be written out to a file as Verilog that instantiates behavioral logic as structural instances. This file is essentially a gate-level netlist in the form of Verilog, and provides a way to more accurately simulate the code as if it had been flashed to the FPGA. If any errors occur during synthesis, they will be sent back and the WebSocket will be closed. Otherwise, the process continues.
- CVC will then begin gate-level simulation of the code. In addition, it will use a testbench that will simulate a 100 Hz clock and instantiate the top module in the aforementioned code, along with the necessary libraries used by Yosys in synthesizing the code. At this point, any new messages issued from the webpage will

be piped into the simulation (since the webpage will only send the encoded inputs from the buttons on the FPGA graphic) and the output of the simulation will be sent back to the page as JSON, which is parsed by the client JS to update the FPGA graphic outputs accordingly.

- The simulation can stop if:
    - The user closes the page or disconnects from the Internet,
    - the Freeze/Stop button is clicked, or
    - an error occurs during simulation.

## Implementation

### Results

The simulator was set up on https://verilog.ecn.purdue.edu, where it should still be running. With Rick's help, I obtained a Dell OptiPlex 5050 to sit in the ECE 270 lab (unbeknownst to students) and run the simulator on an 8-core Intel CPU and 64 GB of RAM. The high-end specifications were required to keep the simulator stable during periods of intense activity, e.g. before lab sessions, or during lab practical week.

By the time the simulator was introduced in class in ECE 270 around week 6 of the Fall 2019 semester, I had achieved the intended design, as well as added some additional useful features. Those features included:

- **A Redis database for user account management**: Each student received a username and password to access the simulator, owing to the issue that Purdue authentication protocols could not be easily ported to the machine, preventing us from using BoilerKey

authentication or basic HTTP authentication using the student's ECN login to verify the student's credentials.

- **Process Monitor 2:** A node.js module that gave me tools to monitor the simulator. It has features like automatic process restart, error-based email notifications, and a web interface where I can just login and read error details if needed.
- **Better Verilog error logging:** I decided that we need to keep track of different kinds of errors produced by Verilog code simulated by students. The data helped us keep track of what students were confused with and which students were having errors that they couldn't fix.

**Statistics**

As of 10th December 2019, the following statistics were recorded on the simulator for the 211 students in ECE 270:

- Overall, students simulated 107415 times.
- Students visited the page 7856 times.
- Student code generated a total of 3984 errors.
- The highest number of simulations made by one student over the course of four months was 1424.

Clearly, students were very highly motivated to use the simulator, and judging by the lack of people in office hours this semester, as opposed to previous semesters, indicated that students were doing just fine without having to attend office hours to use the FPGA.

### Future considerations

Rick suggested that I add timing violation handling to the simulator to be able to more accurately simulate setup and hold violations on the FPGA. Currently, CVC has support for SDF files, which are used to specify the timing parameters of flip flop models. When given a valid SDF file, CVC will report timing violations and prevent the flip flop from registering new values on data if setup/hold violations occurred. This was crucial since we had a couple of labs where students did not understand the purpose of using synchronizers to reliably use pushbutton input to clock modules that would use the value of the pushbuttons.

The simulator code, as-is, is not easily maintainable. I plan to rewrite the code as modules so that future students who may wish to work on the simulator can more easily understand the different parts that make the simulator work, without having to study my code fully.

## Conclusion

The simulator has achieved its main goal of making Verilog simulation and synthesis tools for the specific board used in the ECE 270 lab easily available to students with the help of a web-enabled FPGA simulator, available at https://verilog.ecn.purdue.edu. Starting next semester, I will attempt to add the features discussed in future considerations with Rick's help, as well as any features that are discussed as a result of what the class may need in the future.

**Acknowledgements**

**Rick**, without whose extremely valuable support, this entire project would not have been possible.

**The open source community** for making all the software I used in my experiments and final project free and publicly available.

**My peers, fellow UTAs and students in ECE 270** who gave incredibly important perspective and feedback on the project.

## References

[1] “How Uber Uses Node.js to Scale Their Business,” *Node.js foundation*. [Online]. Available: https://foundation.nodejs.org/wp-content/uploads/sites/50/2017/09/Nodejs-at-Uber.pdf. S. Info. [Accessed: 15-Jun-2019]

[2] “8 Top Companies That Rely on Node.js,” *Simply Technologies*, 19-Apr-2018. [Online]. Available: https://www.simplytechnologies.net/blog/2018/4/18/8-top-companies-that-rely-on-nodejs. [Accessed: 2-Oct-2019].

[3] PayPal, “Node.js at PayPal,” *Medium*, 09-Jul-2018. [Online]. Available: https://medium.com/paypal-engineering/node-js-at-paypal-4e2d1d08ce4f. [Accessed: 15-Dec-2019].

[4] “What is CVC?,” *Tachyon DesignAutomation What is CVC?* [Online]. Available: http://www.tachyon-da.com/what-is-cvc/. [Accessed: 12-Aug-2019].

[5] C. Wolf and J. Glaser, “Yosys - A Free Verilog Synthesis Suite,” *Proceedings of Austrochip 2013*, 2013. [Accessed: 20-Aug-2019]