ELECTRICAL & COMPUTER ENGINEERING

Lab 4: FPGA INTERACTIONS AND SYSTEM VERILOG

The University of Washington | The Remote Hub Lab | Last Revised: March 2022

Summary

Building on the background knowledge gained from the previous labs, Lab 4 introduces the Hardware Description Language (HDL) SystemVerilog. In this lab, you will learn about 1) Quartus for SystemVerilog synthesis, 2) ModelSim for testing and simulation, and 3) Writing code to Field Programmable Gate Arrays (FPGAs).

Table of Contents

What is a Field Programmable Gate Array (FPGA)?

FPGAs are hardware chips with programmable logic cells -- electrical components which a designer can use to customize and configure circuits for specific purposes. Digital circuits typically consist of a large amount of logic components which necessitates a way to reconfigure circuits on scale, and this is why FPGAs are used.

To program an FPGA, we need to use a Hardware Description Language (HDL). This lab series uses an HDL called SystemVerilog, but note that there are other HDLs such as VHDL. After writing and synthesizing the code, it is good practice to simulate the output. This allows designers to check if the code results in what they expect before sending it to the FPGA. Sending a design to an FPGA means converting the HDL code into bitstreams (0s and 1s) that are understandable by the machine. This workflow is demonstrated in Figure 1.

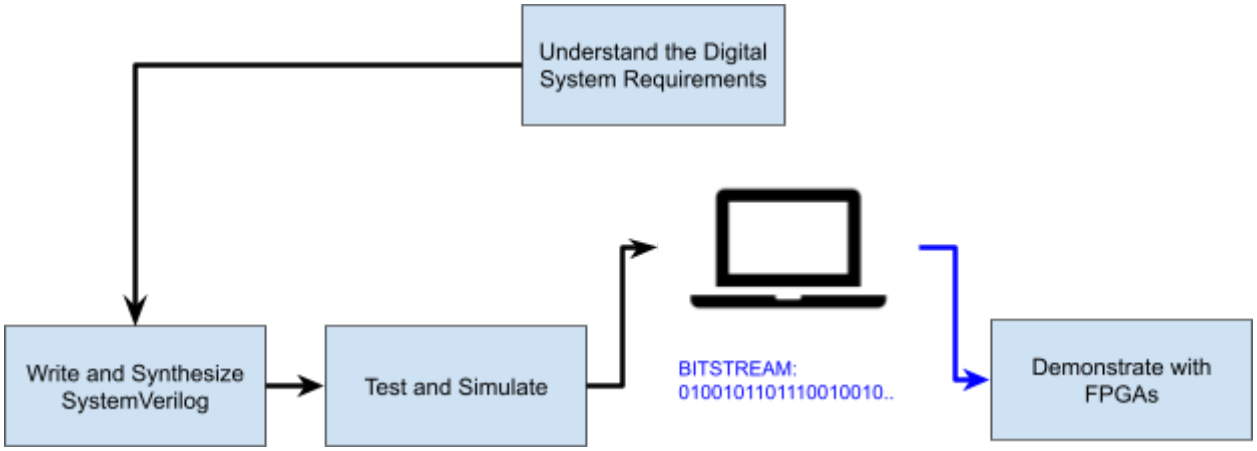

Figure 1: Understanding → SystemVerilog → Synthesis → Simulation→ FPGA Workflow

In this lab, we'll walk through each step of the workflow as depicted in Figure 1. As in the Digital Trainer Activity, we'll use LabsLand to access physical FPGAs remotely. We'll also use ModelSim for simulation and Quartus Prime Software to write and synthesize code. But what code should we write and how do we write it? To answer this question, let's look at the basics of SystemVerilog.

Implementing Digital Logic Using a Hardware Description Language

Description

In Labs 0-3, you gained skills to define various digital systems on paper. How do we do this on a computer? That's where a Hardware Description Language (HDL) like SystemVerilog comes in. Creating a digital design involves three steps: Describing the design using HDL with a software like Quartus, verifying the design with a tool like ModelSim, and then implementing the verified design on an FPGA. In this section, we will introduce the major SystemVerilog Syntax through a full adder example which will suffice for the needs of this lab. Interested readers can also find more resources about SystemVerilog on the Internet.

A full adder performs bitwise addition using two bits (A, B) and a carry-in value (cin), resulting in sum and carry-out (cout) outputs. Figure 2 shows an example of a SystemVerilog program which creates a full adder unit.

```
1  /* RHLab: Lab 4
2     The fullAdder module adds together two 1-bit numbers. */
3
4  module fullAdder (A, B, cin, sum, cout);
5
6     input logic A, B, cin;           // given: 1-bit A, B, and cin
7     output logic sum, cout;          // result: 1-bit sum and 1-bit carry-out
8
9     assign sum = A ^ B ^ cin;        // A XOR B XOR cin
10    assign cout = A&B | cin & (A^B); // (A AND B) OR cin AND (A XOR B)
11
12 endmodule
```

Figure 2: Full Adder Module

SystemVerilog Syntax and Main Components

general notes

- Notice how most lines in SystemVerilog except for "endmodule" end with a semicolon (;).
- Notice the indents and white space left between lines for readability.

modules

- Think of this like the "container" for your code. A module begins with the starting line syntax "module [name()]". A module ends with the line "endmodule."
- See lines 4 and 12 in Figure 2.

input and output logic

- These lines are akin to declaring your variables. You list your input signals after the syntax “input logic” , and you list your output signals after the syntax “output logic” respectively.
- See lines 6 and 7 in Figure 2.

assign statements

- assign statements define the variables’ meaning with Boolean expressions.
- See Figure 2’s lines 9-10 which use assign statements as supposed to lines 6-7’s variable declarations.

comments

- Leave comments in your code for readability. Use one double forward slash at the beginning of in-line comments (//) and a single forward slash and star at the beginning and end of multi-line comments (/*...*/).
- In Figure 2, see lines 6-10 for in-line comments and lines 1-2 for multi-line comments.

Defining Logic functions with Bitwise operators

- Bitwise operators are used to define logic functions such as the sum and cout equations in lines 9 and 10 of figure 2. Table 1 summarizes the bitwise operators used in SystemVerilog

***Table 1**: Bitwise Operators*

Operator	Syntax Symbol	Boolean Expression → SystemVerilog Example
AND	&	$A\ AND\ B \rightarrow A\ \&\ B$
OR	|	$A\ OR\ B \rightarrow A \mid B$
NOT	~	$\overline{A} \rightarrow \sim A$
NOR	(combination)	$A\ NOR\ B \rightarrow \sim(A \mid B)$
NAND	(combination)	$A\ NAND\ B \rightarrow \sim(A\ \&\ B)$
XOR	^	$A\ XOR\ B \rightarrow A\ \hat{}\ B$

Note: Do not use ‘+’ for the OR operator, and do not use ‘x’ for the AND operator in SystemVerilog code. They mean different things in SystemVerilog.

Quartus, ModelSim, and LabsLand Activity

Now that you have some background on Quartus, ModelSim, and LabsLand, we can jump into a practice exercise. The following steps will provide a walkthrough of the Understanding → SystemVerilog → Synthesis → Simulation → FPGA workflow using the full adder example.

1. **Understand the system requirements.**
    a) Before coding, we need to understand the system we will be implementing.
    b) One way to accomplish this first step is on paper (as we've seen in Labs 0-3). Another way is to draw a schematic diagram of the system.
    c) To help you understand the full adder:
        i) Open the project skeleton you created in Lab 0.
        ii) Follow this video tutorial to create a schematic diagram of a full adder's components and logic gates: https://youtu.be/qn6ggwxpDjQ?t=86.
        iii) Note that the video skips to timestamp 1:26 since you already created a project skeleton.

2. **Practice writing SystemVerilog in Quartus, synthesizing your system in Quartus, and simulating your system in ModelSim.**
    a) Follow the instructions in this video tutorial to design and simulate the full adder in Quartus and ModelSim: https://www.youtube.com/watch?v=BcvclrqZ2fc.
    b) This tutorial will help you implement the SystemVerilog code in Figure 2 in addition to writing a verification code to simulate the design in ModelSim.
    c) Note that it is intentional that the code in figure 2 is not "copy-and-paste-able" to Quartus; you must type out the syntax of the code to gain familiarity with the (HDL).

3. **Prepare your code for the LabsLand FPGA.**
    a) Follow the instructions in this video tutorial to create and simulate a SystemVerilog module called "DE1_SoC" for your FPGA: https://www.youtube.com/watch?v=mnZt2iNNfp4.
    b) Refer to Figure 3 to view the DE1_SoC code for both the design module and testbench.
    c) Note that this tutorial helps you make another module which instantiates the full adder system so we can interact with FPGA inputs and outputs. This hierarchical structure is a common feature of HDL. Because we are making another module, we must synthesize and simulate at this higher level as well.

```
/* RHLab: Lab 4
   The DE1_SoC module communicates to the physical FPGA board. */

module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);

   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;   // outputs on board: Six 7-seg HEX displays, 10 LEDRs
   input logic [3:0] KEY;     // inputs on board: 6 Keys, 10 Switches
   input logic [9:0] SW;

   fullAdder FA (.A(SW[2]), .B(SW[1]), .cin(SW[0]), .sum(LEDR[0]), .cout(LEDR[1]));

   // All HEX displays should be off (HEX segments are active 'low' when the bit == 0)
   assign HEX0 = 7'b1111111;
   assign HEX1 = 7'b1111111;
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;


endmodule

module DE1_SoC_testbench();

   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic [9:0] LEDR;
   logic [3:0] KEY;
   logic [9:0] SW;

   DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW);

   integer i;
   initial begin
   SW[9] = 1'b0;
   SW[8] = 1'b0;
      for (i = 0; i <2**8; i++) begin
         SW[7:0] = i; #10;
      end
   end

endmodule

```

Figure 3: DE1_SoC Module and Testbench

4. **Demonstrate the digital system on an FPGA.**
   a) Login to LabsLand, navigate to your main dashboard with the RHLab activities, and choose “Intel DE1-SoC”. Click the “Access this Lab” button.

Figure 4: Intel DE1-SoC Activity

b) Locate “DE1 IDE SystemVerilog” and click the “Access” button below it. You will be directed to a new page called “SystemVerilog IDE for DE1-Soc”.

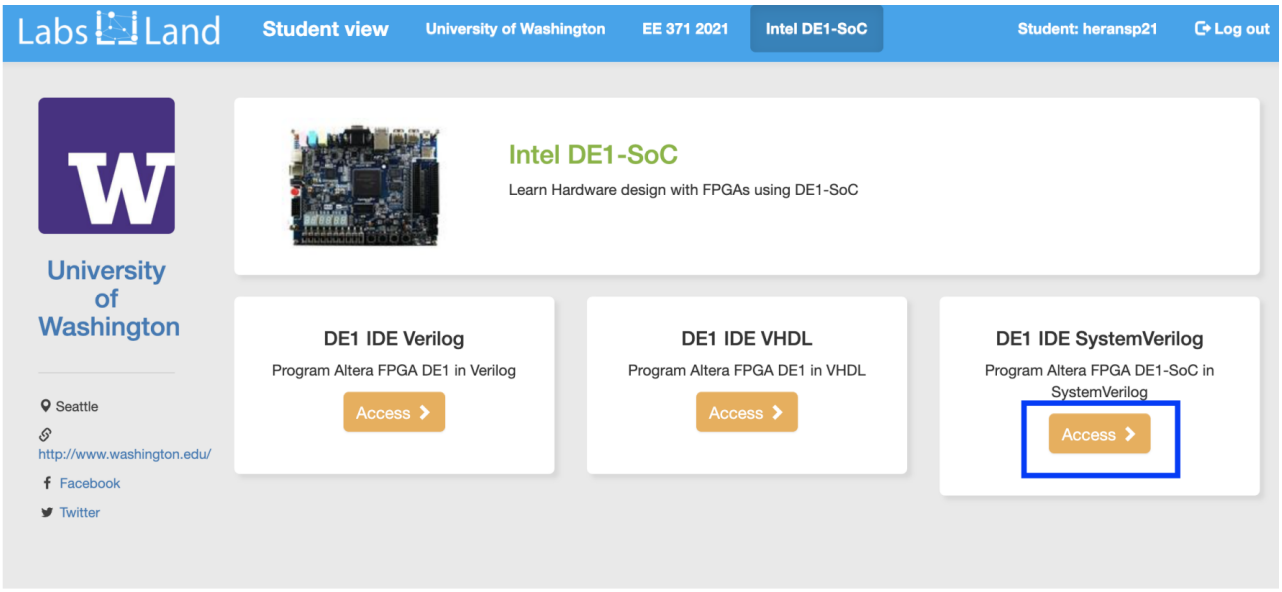

Figure 5: DE1 SystemVerilog IDE

c) In the following page, select the “Add” button to import the top-module “DE1-SoC.sv” and file “full_adder.sv” that you created earlier into LabsLand.

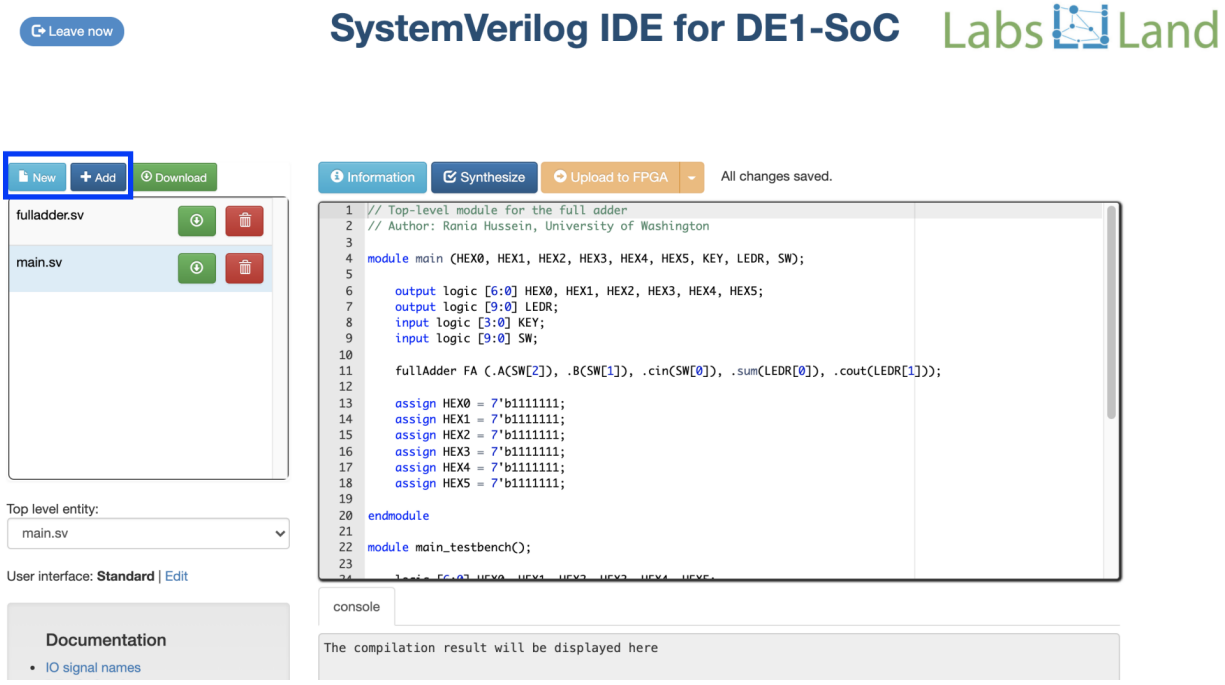

Figure 6: Add Modules into LabsLand

d) Choose the top-module using the dropdown menu under “Top level entity” (boxed in red in Figure 7). Make sure you select “DE1-SoC.sv” as the top-module.

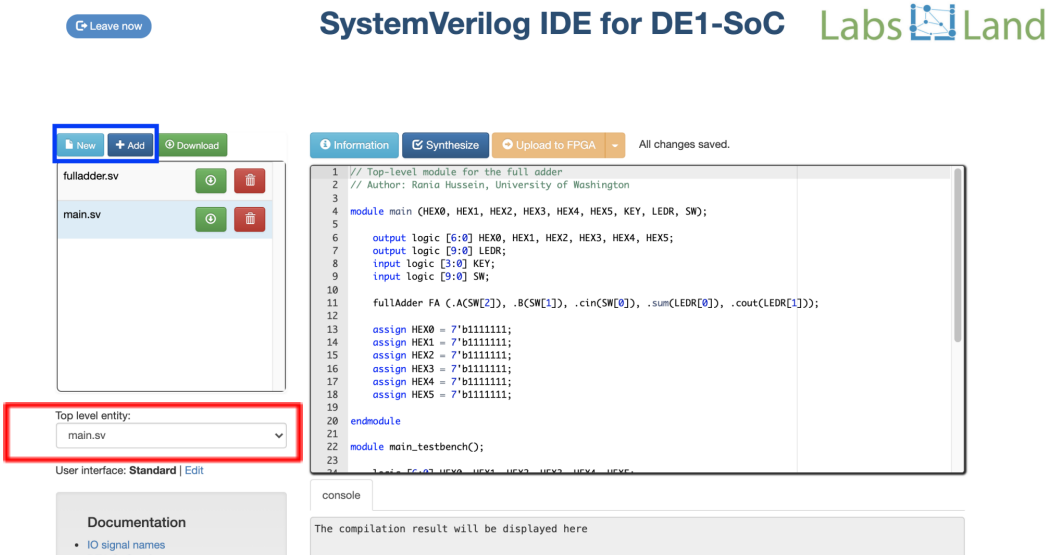

Figure 7: Setting the Top Level Module

(Alternatively, you may also create new files by clicking “New” and copy the provided full adder example under “Examples” found in the bottom left corner of the interface into the corresponding new files. The top-module is named “main.sv” in this case, so make sure to adjust the settings accordingly.)

e) You will then be able to synthesize the code using the button “Synthesize”. Once the synthesis is complete and succeeds without errors, you can click on “Upload to FPGA” to load your design onto an FPGA.

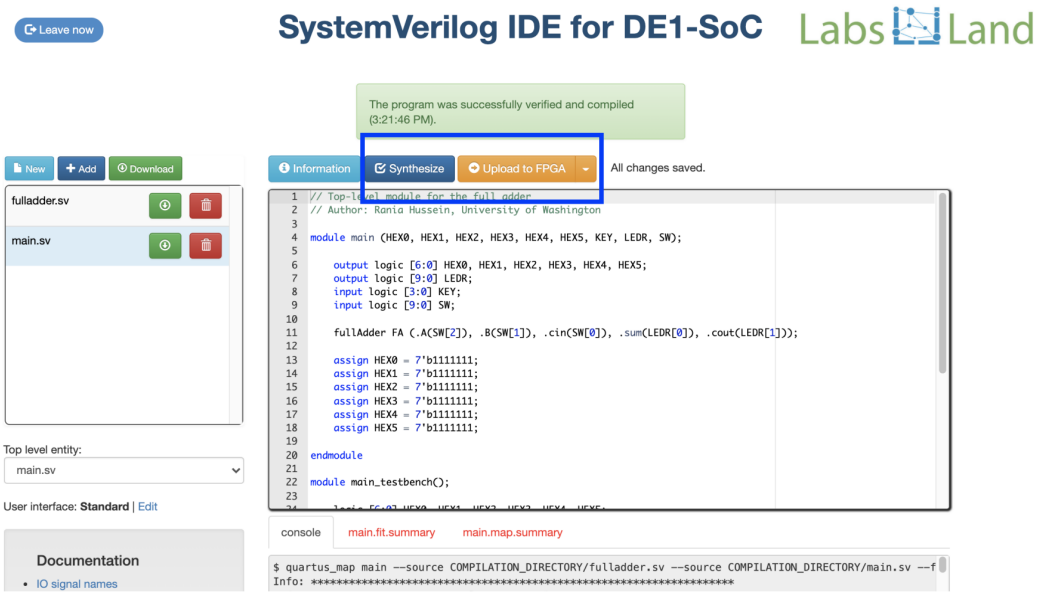

Figure 8: Synthesizing on LabsLand

f) After connecting to the remote FPGA, you will see a webpage like Figure 9's.

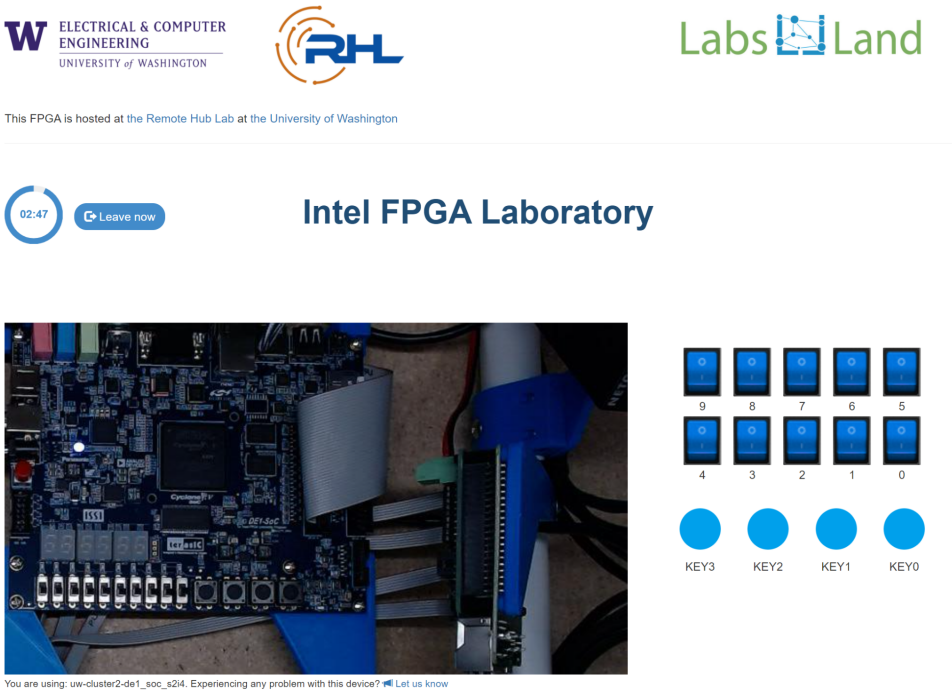

Figure 9: An FPGA on LabsLand

The right part of the page shows the buttons and keys of the FPGA. You can click on the buttons and keys accordingly as inputs. It is important to note that 'KEYS' need to be held down, as they do not function like switches.

g) Toggle Switch2 (A), Switch1 (B), and Switch0 (Cin) to test different input combinations. Check that your system in LabsLand outputs signals using LEDR1 (Cout) and LEDR0 (Sum) which are consistent with the Truth Table in Table 2.

Table 2: Full Adder Truth Table

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Reflection and Observations

Write down your initial thoughts about the Understanding → SystemVerilog → Synthesis → Simulation → FPGA workflow. What did you observe throughout the process?

What did you notice about the SystemVerilog code that you'd like to learn more about?

What questions do you have, if any?