ECE 272 Lab 4
Fall 2018

SystemVerilog and Complex Projects

Phi Luu

October 31st, 2018

Grading TA: Edgar Perez

Lab Partner: Benjamin Geyer

1 Introduction

This lab focuses on implementing combinational logic using a hardware description language (HDL) called SystemVerilog. SystemVerilog uses text to define the specifications of a system. Although it is not as intuitive as describing a system with a schematic (like labs 1, 2, and 3), SystemVerilog allows the users to design more complex systems in a simpler way, especially for sequential circuits.

A simple SystemVerilog-schematic comparison can be seen in a two-input AND gate example in Figure 1 below:

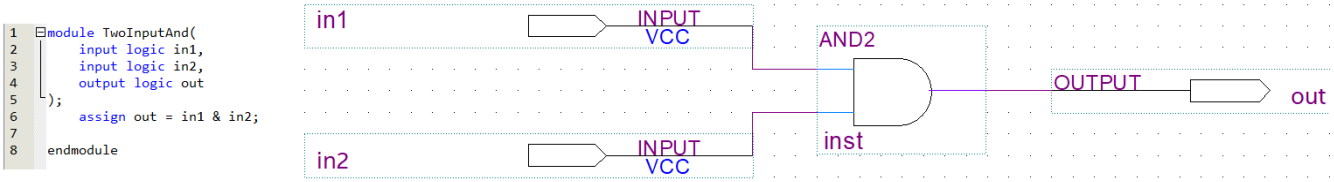

Figure 1: A high-accuracy-demand process of drawing a schematic can be simplified into 8 lines of SystemVerilog text file

During this lab, I and Ben design a seven-segment display counter and multiplier. We use 10 switches from the FPGA to represents 10-bit binary numbers, multiply them with a constant according to the input push buttons, and decode the digits to 6 seven-segment displays.

2 Design

We use all 10 switches of the FPGA to represent a 10-bit binary input and 6 seven-segment displays to represent a 6-digit decimal output. To produce the output from the inputs, we implement a 12:17 multiplexer which takes the 10-bit input of the switches as the multiplicand and 2-bit push buttons as the multiplier. The output of the multiplexer is expressed as a 17-bit binary. Next, we parse that output into a parser to split the 17-bit binary into 6 groups of 4-bit binary numbers. Each group of 4-bit binary numbers will then go to a seven-segment display decoder and produce the corresponding digit. Intuitively, the system is illustrated as in Figure 2 below:

Since the LEDs in the seven-segment displays are active-low, the output signal needs to be LOW (or 0) to activate the LEDs and to be HIGH (or 1) to deactivate the LEDs. Table 1 decodes a decimal number into a binary number and then to the signal in a seven-segment display.

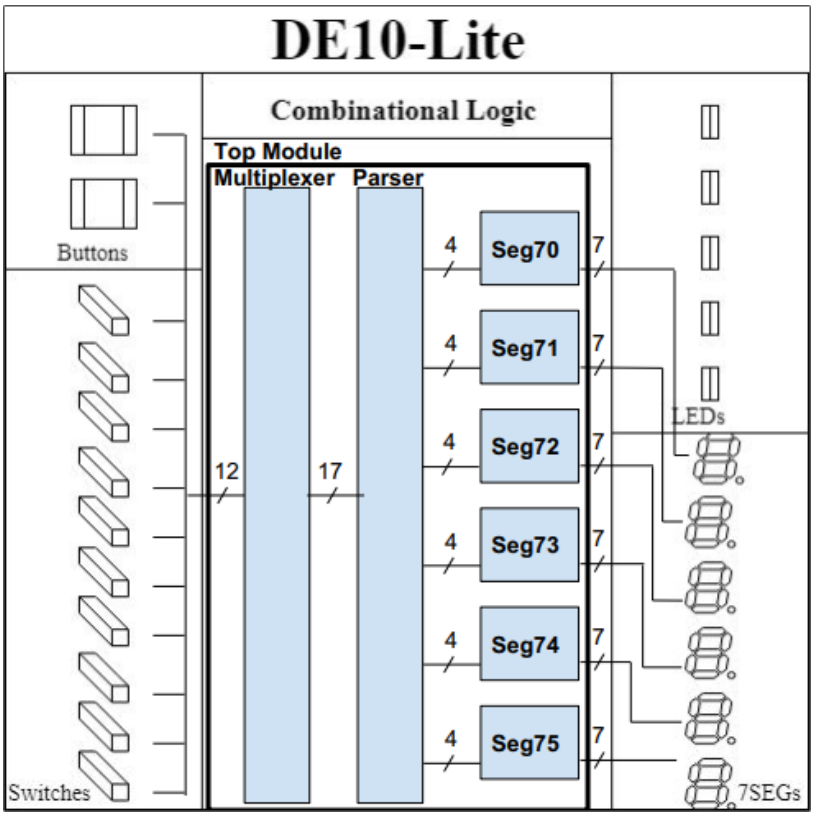

Figure 2: Block diagram

When we implemented a seven-segment display decoder in lab 3, we had to use Karnaugh maps to simplify Boolean equations. We also had to spend a lot of time connecting visual elements on the schematic of Quartus Prime. In this lab, however, we only need to type descriptions of the hardware into SystemVerilog files, and the compiler will take care of the rest. This feature will be quite convenient when we design highly complex circuits in the future.

Because we don't draw any schematic in this lab, we write SystemVerilog code that describes the connections between the elements in the circuit. The source code can be found in the Appendix section at the end of this report.

We designate the switch at the corner of the FPGA as bit 0 of the Switches array in our SystemVerilog code (the switch labeled as *SW0* on the board) and so on. We designate the seven-segment display nearest to the switches as Seg70 (the display labeled *HEX0* on the board). Finally, we designate the push button further from the switches as bit 0 of the Buttons array in our SystemVerilog code (the button labeled *KEY0* on the board). See the Appendix for more details.

Decimal	Binary	Seg_A	Seg_B	Seg_C	Seg_D	Seg_E	Seg_F	Seg_G
0	0000	0	0	0	0	0	0	1
1	0001	1	0	0	1	1	1	1
2	0010	0	0	1	0	0	1	0
3	0011	0	0	0	0	1	1	0
4	0100	1	0	0	1	1	0	0
5	0101	0	1	0	0	1	0	0
6	0110	0	1	0	0	0	0	0
7	0111	0	0	0	1	1	1	1
8	1000	0	0	0	0	0	0	0
9	1001	0	0	0	0	1	0	0

Table 1: Conversion table between decimal, 4-bit binary, and seven-segment signals

3 Results

We compile and simulate the project on ModelSim and obtain the following waveforms:

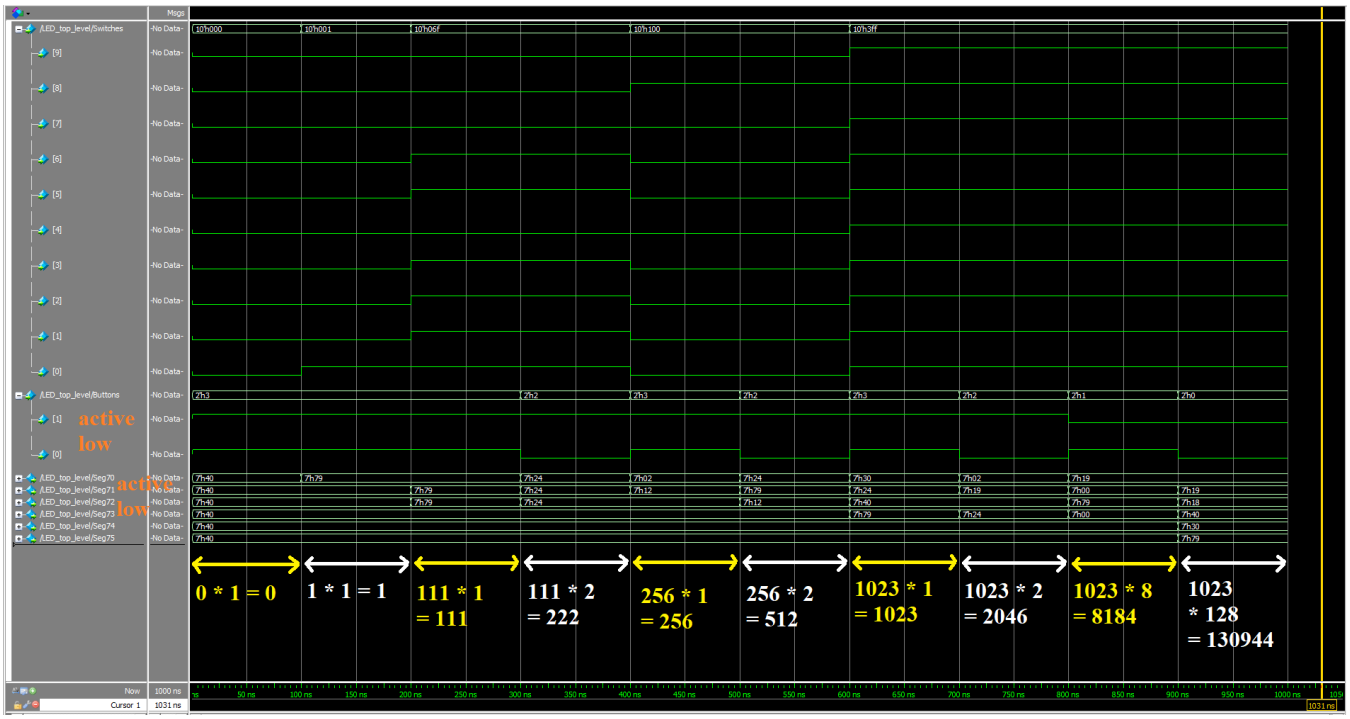

Figure 3: Simulation waveform of a few test examples. High-resolution image

Each 100-ns interval (2 columns) is a test case. Each of the test cases' inputs and output are specified with a two-way arrow. For example, from 100 ns to 200 ns (from the third column to the fourth column), the switch input is equivalent to a 1 in decimal system (*SW1* turned on and all other switches turned off), and the button input is equivalent to a 1 in decimal system (all buttons not pressed), and the output of all 6 seven-segment displays is 000001 (reading from Seg75 to Seg70).

Using the same method, we obtain the result of all test cases. Since the results are consistent with mathematical calculations, we believe the simulation of the project is a success.

Finally, we upload the project to the real FPGA. We do thorough tests on the real board, and the results are consistent with the simulations as well as mathematical laws. Therefore, we have successfully implemented a seven-segment display decoders with a multiplexer and a parser using SystemVerilog.

4 Experiment Notes

Reflection

It took me approximately 20 minutes to get used to writing SystemVerilog on Quartus Prime. The combinational logic portion of the project was quite easy. I started to prefer writing SystemVerilog to sketching schematic because I could simplify tedious graphic circuit contruction using a few lines in a *case* block of SystemVerilog.

I think this lab was an easy yet a necessary lab for students to become more familiar with SystemVerilog and how to use HDL to "write" circuits. I plan to program circuits in SystemVerilog from now on because it is more consistent than wiring the components in a graphical interface.

Study Questions

1. Describe how to control a 7-seg display using a state machine and lighting each digit one at a time.

   To control a single seven-segment display which displays decimal digits, a 10-state state machine is needed—each state for every decimal digit. Let S0 be the state corresponding to the decimal digit 0, S1 be the state corresponding to decimal digit 1, and so on. Since the FSM lights each digit one at a time, using this structure, Table 2 is constructed as the state transition table:

   10 states are encoded into 4-bit binary numbers, as follows:

   Using Table 1 as a binary-to-seven-segment conversion table, the state transition table with binary encodings and outputs is constructed in Table 4 below:

   Next, use sum-of-products form to express outputs $\text{Seg}_{G:A}$ and next states $\text{S}'_{3:0}$ in terms of current states $\text{S}_{3:0}$. Then, apply Karnaugh-maps to simplify the Boolean equations.

   Finally, use these sum-of-products Boolean equations to construct a schematic, or use Table 4 to write a hardware description module using an HDL.

Current State S	**Next State S'**
S0	S1
S1	S2
S2	S3
S3	S4
S4	S5
S5	S6
S6	S7
S7	S8
S8	S9
S9	S0

Table 2: State transition table of a 7-seg FSM

State	**Encoding $S_{3:0}$**
S0	0000
S1	0001
S2	0010
S3	0011
S4	0100
S5	0101
S6	0110
S7	0111
S8	1000
S9	1001

Table 3: State binary encoding of a 7-seg display FSM

Current State				**Outputs**							**Next State**			
S_3	S_2	S_1	S_0	Seg_G	Seg_F	Seg_E	Seg_D	Seg_C	Seg_B	Seg_A	S'_3	S'_2	S'_1	S'_0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	1	1	1	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	1	0	1	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	1	1	0	0	1	0	1	0	1
0	1	0	1	0	0	1	0	0	1	0	0	1	1	0
0	1	1	0	0	0	0	0	0	1	0	0	1	1	1
0	1	1	1	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0

Table 4: State transition and output table with binary encodings of a 7-seg FSM

Appendix

lab4.qsf (Pin Assignment)

```
set_location_assignment PIN_C14 -to Seg70[0]
set_location_assignment PIN_E15 -to Seg70[1]
set_location_assignment PIN_C15 -to Seg70[2]
set_location_assignment PIN_C16 -to Seg70[3]
set_location_assignment PIN_E16 -to Seg70[4]
set_location_assignment PIN_D17 -to Seg70[5]
set_location_assignment PIN_C17 -to Seg70[6]
set_location_assignment PIN_C18 -to Seg71[0]
set_location_assignment PIN_D18 -to Seg71[1]
set_location_assignment PIN_E18 -to Seg71[2]
set_location_assignment PIN_B16 -to Seg71[3]
set_location_assignment PIN_A17 -to Seg71[4]
set_location_assignment PIN_A18 -to Seg71[5]
set_location_assignment PIN_B17 -to Seg71[6]
set_location_assignment PIN_B20 -to Seg72[0]
set_location_assignment PIN_A20 -to Seg72[1]
set_location_assignment PIN_B19 -to Seg72[2]
set_location_assignment PIN_A21 -to Seg72[3]
set_location_assignment PIN_B21 -to Seg72[4]
set_location_assignment PIN_C22 -to Seg72[5]
set_location_assignment PIN_B22 -to Seg72[6]
set_location_assignment PIN_F21 -to Seg73[0]
set_location_assignment PIN_E22 -to Seg73[1]
set_location_assignment PIN_E21 -to Seg73[2]
set_location_assignment PIN_C19 -to Seg73[3]
set_location_assignment PIN_C20 -to Seg73[4]
set_location_assignment PIN_D19 -to Seg73[5]
set_location_assignment PIN_E17 -to Seg73[6]
set_location_assignment PIN_F18 -to Seg74[0]
set_location_assignment PIN_E20 -to Seg74[1]
set_location_assignment PIN_E19 -to Seg74[2]
set_location_assignment PIN_J18 -to Seg74[3]
set_location_assignment PIN_H19 -to Seg74[4]
set_location_assignment PIN_F19 -to Seg74[5]
set_location_assignment PIN_F20 -to Seg74[6]
set_location_assignment PIN_J20 -to Seg75[0]
set_location_assignment PIN_K20 -to Seg75[1]
set_location_assignment PIN_L18 -to Seg75[2]
set_location_assignment PIN_N18 -to Seg75[3]
set_location_assignment PIN_M20 -to Seg75[4]
set_location_assignment PIN_N19 -to Seg75[5]
set_location_assignment PIN_N20 -to Seg75[6]
```

```
set_global_assignment -name FAMILY "MAX 10"
set_global_assignment -name DEVICE 10M50DAF484C7G
set_global_assignment -name TOP_LEVEL_ENTITY LED_top_level
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 18.0.0
set_global_assignment -name PROJECT_CREATION_TIME_DATE "11:12:19  OCTOBER 24, 2018"
set_global_assignment -name LAST_QUARTUS_VERSION "18.0.0 Lite Edition"
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 256
set_global_assignment -name SYSTEMVERILOG_FILE LED_top_level.sv
set_global_assignment -name SYSTEMVERILOG_FILE Decoder.sv
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION "23 MM HEAT SINK WITH 200 LFPM
    ↪  AIRFLOW"
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE (CONSERVATIVE)"
set_global_assignment -name SYSTEMVERILOG_FILE Multiplexer.sv
set_global_assignment -name SYSTEMVERILOG_FILE Parser.sv
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL PLACEMENT_AND_ROUTING
    ↪  -section_id Top
set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top

set_location_assignment PIN_C10 -to Switches[0]
set_location_assignment PIN_C11 -to Switches[1]
set_location_assignment PIN_D12 -to Switches[2]
set_location_assignment PIN_C12 -to Switches[3]
set_location_assignment PIN_A12 -to Switches[4]
set_location_assignment PIN_B12 -to Switches[5]
set_location_assignment PIN_A13 -to Switches[6]
set_location_assignment PIN_A14 -to Switches[7]
set_location_assignment PIN_B14 -to Switches[8]
set_location_assignment PIN_F15 -to Switches[9]
set_location_assignment PIN_B8 -to Buttons[0]
set_location_assignment PIN_A7 -to Buttons[1]
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -section_id Top
```

LED_top_level.sv

```
module LED_top_level( //Every SV File starts with module and ends with endmodule
    input logic [9:0] Switches, //after the name of the module, all its inputs and
    ↪  outputs are declared in the ()
    input logic [1:0] Buttons,
    output logic [6:0] Seg70, Seg71, Seg72, Seg73, Seg74, Seg75
);
    /*******************************/
```

```
    /* Set internal variables here */
    /*******************************/
    logic [16:0] m_out;
    logic [3:0] p_out0, p_out1, p_out2, p_out3, p_out4, p_out5;

    /*********************************************/
    /* Instanciate and Connect all modules here */
    /*********************************************/

    Multiplexer M(
        .switches(Switches),
        .buttons(Buttons),
        .out(m_out)
    );

    Parser P(
        .in(m_out),
        .out0(p_out0),
        .out1(p_out1),
        .out2(p_out2),
        .out3(p_out3),
        .out4(p_out4),
        .out5(p_out5)
    );

    Decoder D0(
        .Num(p_out0),
        .segments(Seg70)
    );
    Decoder D1(
        .Num(p_out1),
        .segments(Seg71)
    );
    Decoder D2(
        .Num(p_out2),
        .segments(Seg72)
    );
    Decoder D3(
        .Num(p_out3),
        .segments(Seg73)
    );
    Decoder D4(
        .Num(p_out4),
        .segments(Seg74)
    );
    Decoder D5(
        .Num(p_out5),
```

```
        .segments(Seg75)
    );

endmodule
```

Multiplexer.sv

```
module Multiplexer(
    input logic [9:0] switches,
    input logic [1:0] buttons,
    output logic [16:0] out
);

    always_comb
        case(buttons)
            0: out = switches * 128;
            1: out = switches * 8;
            2: out = switches * 2;
            3: out = switches;
        endcase

endmodule
```

Parser.sv

```
module Parser(
    input logic [16:0] in,
    output logic [3:0] out0, out1, out2, out3, out4, out5
);

    always_comb begin
        out0 = in % 10;
        out1 = (in / 10) % 10;
        out2 = (in / 100) % 10;
        out3 = (in / 1000) % 10;
        out4 = (in / 10000) % 10;
        out5 = (in / 100000) % 10;
    end

endmodule
```

Decoder.sv

```
module Decoder(
    input logic [3:0] Num,
    output logic [6:0] segments
);
    always_ff @(*)
        case(Num)
            0: segments=7'b100_0_000;
            1: segments=7'b111_1_001;
            2: segments=7'b010_0_100;
            3: segments=7'b011_0_000;
            4: segments=7'b001_1_001;
            5: segments=7'b001_0_010;
            6: segments=7'b000_0_010;
            7: segments=7'b111_1_000;
            8: segments=7'b000_0_000;
            9: segments=7'b001_1_000;
            default: segments= 7'b111_1111;
        endcase

endmodule
```