Mechtron 3TB4: Lab 1

Describe Circuits using Verilog

Reports Due:
Prelab: 11:59PM of the day of your lab session
Post-lab: 11:59PM of the day of your next tutorial session

Note: Connect to the VPN prior to starting Quartus Prime, otherwise compilation will be unavailable.

Goals

In this part of the lab, you will continue to practice using Quartus Prime, to learn Verilog HDL and to practice programming the DE1-SoC board. This lab session requires you to describe some simple circuits using Verilog, and to simulate them. You are also required to implement a seven-segment decoder and use it to work with the HEX LED on the DE1-SOC board.

Seven-segment Displays

Seven-segment displays remain some of the most ubiquitous display devices. In their basic form, they are used to display decimal numbers. Each display consists of seven segments (hence the name). Each segment is an LED that can be turned on or off to display various patterns. Each segment is controlled independently, although they work together to display a pattern. Segments are numbered from a to g or from 0 to 6 as shown in Figure 1. The figure also shows how the 16 hexadecimal digits can be represented using a seven-segment display. **Please note that to make a segment on the DE1-SoC board light up, logic 0 has to be applied to its input**. For example, to display the number 1 on a seven-segment display, one should apply logic 0 to segments 1 and 2, and logic 1 to all other segments.

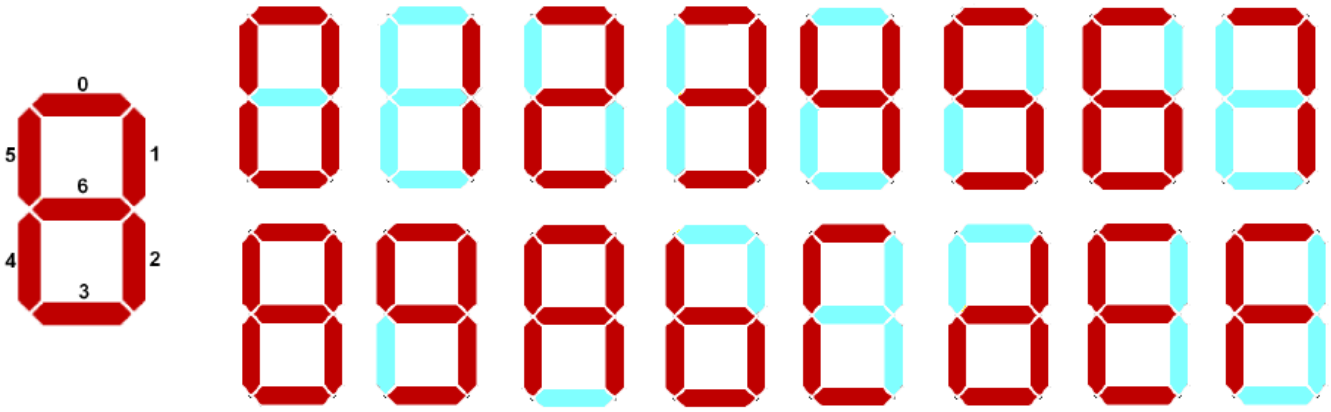

Figure 1: Seven-segment Display

As you already know, all hexadecimal digits can be encoded using four bits. If we have a four-bit number stored somewhere in the system, and we wish to display it on a seven-segment display, we need a code converter circuit that will drive each of the seven segments based on the four-bit value. Such a circuit has one four-bit input, and seven outputs, corresponding to segments 0 through 6, and is sometimes referred

to as a seven-segment decoder.

Activities

Pre-lab [50]

The following activities must be completed and submitted as part 1 of the lab report (the prelab) to Avenue 1 minute prior to the start of your next lab session.

1. **[10 pts]** Submit your report related to the tutorial for Lab 1 as described in the relevant document.
2. Study Verilog HDL. You can start with some web tutorial materials such as the tutorial at: http://www.asic-world.com/verilog/. It would be most useful to pay attention to the following topics:
   - Basic elements such as modules, ports, port directions, data types (wire vs. reg), operators, control statements (if-else, case, loop...).
   - Port connections and rules for port connections.
   - Parameterized modules
   - Continuous assignment, blocking assignment and nonblocking assignment.
   - Combinational logic, sequential logic.
   - Always block, sensitivity list, edge sensitive, level sensitive.
   - Using procedural code to implement combinational logic, using procedural code to implement sequential logic.
   - How to avoid inferred latches when using an if-else statement or when using a case statement.
   - New features in 2001 Verilog such as signed shift operator, multi-dimensional array.
3. **[10 pts]** Submit your brief answers for the following questions:
   - What is the `reg` data type and what is the `wire` data type in Verilog?
   - Can the `wire` data type be used on the left side of the assignment statement in a procedural block?
   - What are the rules for module port connection?
   - What are continuous assignment, blocking assignment and nonblocking assignment?
   - What is the difference in procedural coding when implementing combinational logic and sequential logic?
   - How does one avoid inferred latches when using Verilog to describe circuits?
   - What is the difference between the operators "<<" and "<<<"?
   - How to declare an array of 6 elements of a 7-bit wire?
4. **[10 pts]** Refresh your knowledge for the basic logic blocks, specifically: flip-flops, latches, registers, counters, multiplexers and decoders. Write Verilog modules to describe: One bit data width D flip-flop, one bit data width D flip-flop with active low synchronous reset, one bit data width D flip-flop with active low synchronous reset and active low enable, a D latch with synchronous enable control, a 4-to-1 multiplexer and a 4-bit counter with reset and enable controls.

5. [**5 pts**] Write a truth table for the seven-segment decoder, that provides the following functionality:

   - The decoder has a four-bit input and a seven-bit output.
   - For the first 10 binary combinations of the input (0000 to 1001), the decoder should provide signals to display decimal digits 0 to 9 on the seven-segment display, as shown in Figure 1.
   - For binary combinations 1010 to 1110, the decoder should produce signals to display the first five letters of your first name. If your first name is shorter than five letters, use additional letters from your last name. Keep in mind that you need to drive logic 0 to turn a segment on. It may not be possible to display some letters in a meaningful way on the seven-segment display. You should use approximations if you require a letter that cannot be displayed easily on the seven-segment display.
   - For the binary combination 1111, the decoder should provide signals to turn off the seven-segment display (nothing is displayed).

   You should produce the complete truth table corresponding to what you are required to display, as specified above.

6. [**5 pts**] Using either K-maps or algebraic manipulation, derive logic expressions for segments 0 and 1 in the truth table you created in step 5. Note that the logic expressions need not be minimal.

7. [**10 pts**] Describe in Verilog a circuit that implements the truth table created in step 5. Use the "`assign`" statements to specify the functionality of segments 0 and 1 and behavioral descriptions to specify the functionality of other segments. Use the following skeleton code for your module:

```
module seven_seg_decoder(input [3:0] x, output [6:0] hex_LEDs);
        reg [6:0] reg_LEDs;

        assign hex_LEDs[0]= /* expression for segment 0 */;
        assign hex_LEDs[1]= /* expression for segment 1 */;

        assign hex_LEDs[6:2]=reg_LEDs[6:2];

        always @(*)
        begin
                case (x)
                        4'b0000:  reg_LEDs[6:2]=5'b10000;  //7'b1000000    decimal 0
                        4'b0001:  reg_LEDs[6:2]=5'b11110;  //7'b1111001    decimal 1
                        4'b0010:  reg_LEDs[6:2]=5'b01001; //7'b0100100     decimal 2

                        /* finish the case block */

                endcase
        end

endmodule
```

In the lab [40]

You and your group members may have different solutions from your pre-lab (pre-labs should be completed individually). In the lab you need to demonstrate only one working program/circuit per group.

1. [**10 pts**] Create a new Quartus project, name the project `lab1part1`. Input your Verilog modules for the basic logic blocks prepared in pre-lab step 4 . Compile and simulate them, and verify the functional simulation result. To simulate each module, you need to set it as the "Top-Level Entity". To set a module as the Top-Level Entity, right click the module name in the Files tab in the Project Navigator window, then select the command "Set as Top-Level Entity". Take screenshots of the simulation results and include them in your lab report.

   The correct behavior of a circuit can be checked using functional and timing simulations. For the current version of Quartus, timing simulation is not supported for Cyclone V, which is used on the DE1-SoC board.

2. [**10 pts**] Create another new Quartus project to test your seven-segment decoder. Name this project `lab1part2`. In this project, you will feed the values of four switches on the DE1-SoC board directly to your seven-segment decoder module and display the decoder's result on one of the HEX LEDs on the board.

   You need to create a a top level module `lab1part2.v`, which takes at least four of the ten SWs of the DE1-SoC board as input, and outputs signals to one of the six HEXs of the DE1-SoC board. In the top level module, instantiate your module of the seven-segment decoder.

   You can use the DE1-SoC peripheral names as they are used in `DE1_SoC.qsf`, the pin assignment file provided by Intel/Altera. In this way, you can assign pins for your project by importing the pin assignment file `DE1_SoC.qsf`. (`Assignment | Import Assignment...`, then browse to the file `DE1_SoC.qsf` and click Open to import it.)

   The following sample top level module declaration uses SW0, SW1, SW2 and SW3 as input and HEX0 as output:

   ```
   module lab1part2 input [3:0] SW,  output [6:0] HEX0,);
   /* fill in your code here */
   endmodule
   ```

   Compile your project and program your DE1-SoC board to test your seven-segment decoder. Demonstrate your working project to one of the TAs. Before compiling, make sure that all unused pins are reserved as **Input tri-stated**. This option is available under `Assignments | Device > Device and Pin Options > Unused Pins`.

3. [**20 pts**] Create another new Quartus project and name it `lab1part3`. The requirements for this project are:

   - When the program starts, all HEX displays are off, except HEX0.
   - If SW9 and SW8 are both on, HEX0 is off. Otherwise it will display the results from your seven-segment decoder.

- If SW9 and SW8 are both off, your seven-segment decoder takes input from {SW3:SW0}.
- If SW9 is off and SW8 is on, your seven-segment decoder takes input from a 4-bit counter. This 4-bit counter counts the times that KEY3 of the DE1-SOC is pressed.
- If SW9 is on and SW8 is off, your seven-segment decoder takes input from the most significant four bits of a 30-bit counter. This 30-bit counter counts the pulses of CLOCK_50 of the DE1-SOC board.
- Anytime KEY0 is pressed, the 4-bit and the 30-bit counters will be reset to 0.
- You need to use a counter (either parameterized or not) and multiplexer modules, and instantiate and bind them in your top level module. An example of a parameterized counter with reset_n and enable control is as follows:

```
module counter(input clk, reset_n, enable, output reg[WIDTH-1:0] result);

parameter WIDTH=4;
//2's power of 26 is 67,108,864.
// that is  1 second (226/50e6 = 1.34), if count is using CLOCK_50

always@(posedge clk, negedge reset_n)
    if(!reset_n)begin
        result<=30'b0;
    end else if(enable)begin
        result<=result+1'b1;
    end
endmodule
```

3.1. Use Netlist Viewers from the Quartus tools menu to look at different RTL views of your circuit for `lab1part3`. Take screenshots and include them in your report.

3.2. Use `Assignment | Pin Planner` to confirm that pins have been correctly assigned for your project `lab1part3`. Take a screenshot and include it in your report.

3.3. Download your design to the FPGA and test the behavior of your circuit. Demonstrate its operation to one of the TAs.

Lab Reports (Part 2) [10]

This part of the lab concerns your presentation and report writing skills. Describe what you did in this lab, include the code used as well as screenshots taken. Answer the following questions.

1. In two to three sentences explain the role of the `DE1-SoC.qsf` file that you imported into your design.
2. Open the compilation report in Quartus, and report the following numbers:
    - Total number of logic elements used by your circuit.
    - Total number of registers.
    - Total number of pins.
    - The maximum number of logic elements that can fit on the FPGA you used.