Nicholas Wroblewski, Michael Stewart

Professor Robinson

ECEN 2350 Digital Logic

8 October 2019

Lab 1

**Abstract**

The goal of this lab was to introduce students to iVerilog, GTKWave, Quartus, and the DE10-Lite Board. At completion, students will have greatly increased their knowledge of all these topics. Students can work with one partner, which allows us to reinforce our team skills when working on an engineering project. Three design blocks will be completed at the end of this lab, and at the end, they will be combined into one program that will run on the board. Design blocks are used to separate each section of this project into more manageable workloads.

**Design Block 1**

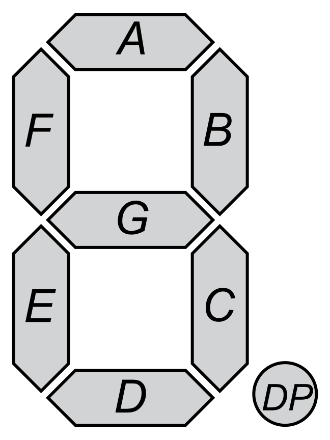
 Design block one was a simple introduction to programming the DE10 board. It had students display their and their partner’s birthdays on the seven-segment displays. The user can switch which birthday is displayed by pressing KEY0. Also, switches zero to seven must light up when they are turned on, and their inverse is shown when KEY0 is pressed. All code was written in one Verilog file, and only one module was used. We chose not to create a hex decoder for this design block since each partner’s birthday are so similar. One birthday is 04.05.00, and the other is 03.18.00. Since only HEX2, HEX3, and HEX4 need to be changed, an assignment statement for each of these HEX displays if used. These seven-segment displays take eight-bit binary numbers as an input. We had to decode the display’s drivers because they cannot simply take 8’b0000.0000 as an input of zero. Instead, zero is equivalent to 8'b1100.0000. Using figure one, eight-bit binary values for the seven-segment display registers can be created. An eight-bit input for this display follows the following template: DP,G,F,E,D,C,B,A. With the DE10 board, setting the specific segment as off is a logic high, and setting the segment on is logic low.

Figure 1: Seven Segment Display

**Design Block 2**

Design block two was the most difficult of the three, as it had students create a full bit adder and subtractor from two four-bit two’s complement inputs. Inputs one and two had to be displayed on HEX4 and HEX2 respectively, with their sign displayed on the display left of it. Their sum was displayed on HEX0, and if arithmetic overflow occurred, HEX1 and HEX0 would display 0F. KEY0 would subtract input two from input one. Students were not allowed to simply convert the inputs from binary to decimal and add them.

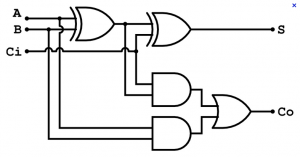
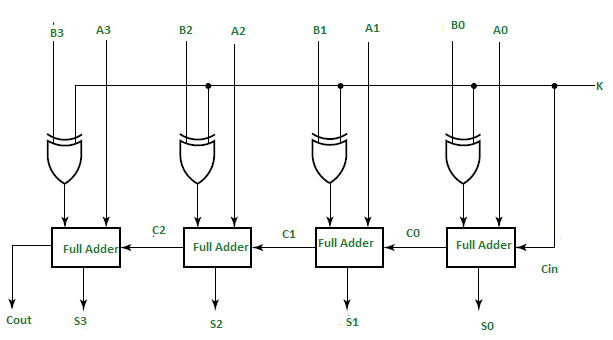
Like design block one, only one Verilog file was created. Three separate modules were created for this design block. One is a full adder, which takes in two bits, a carry bit, and outputs a carry bit and a sum bit. Another module converted a given input so it could be displayed on one of the six seven-segment displays. It takes in a four-bit number and a bit to check overflow, and outputs an eight-bit number to the display showing the sign of the number, and another eight-bit number to the value display.

Figure 2: Four-bit adder

The Verilog code was based on figures two and three. In the full bit adder module, intermediate wires are used to simplify binary arithmetic used.

Figure 3: Full Bit Adder

Creating the seven-segment display converter was by far the most difficult part of this design block. Adding overflow functionality meant that an extra input bit had to be used to check if overflow occurred. All seven-segment outputs were in a case statement, which formed the inputs into a necessary register. This register would be used to easily output a specific register, given an input. For the hex displays showing a negative, blank, or 0, the checkOverflow bit and most significant input bit were used. If overflow occurred, it displayed 0, regardless of the most significant input bit. For the seven-segment displaying the digit, the checkOverflow bit and the input bits were used. If the overflow occurred, F was displayed. The other case statements took the four bits of the input and converted them to an eight-bit register for the seven-segment displays.

**Design Block 3**

Design block three had students compare the switch inputs to see which was greater, or if they were both equal. If they were equal, LEDR[2] would turn on. If input one was greater than input2, LEDR[1] would turn on. If input one was less than input2, LEDR[0] would turn on. HEX5 and HEX4 would display input one, and input two would be displayed on HEX1 and HEX0. HEX2 and HEX3 are turned off for this module. Also if SW[9:8] = 2’b10, the values would be interpreted as unsigned. If SW[9:8] = 2’b11, the values would be interpreted as two’s complement. A case statement inside an always block decided if the inputs would be signed or unsigned. Inside the same always block, if statements were used to compare the values and illuminate the respective LEDs. Also, the same seven-segment conversion module used in design block 2 was used here to display to HEX0, HEX1, HEX4, and HEX5.

**Integration of Design Blocks**

Behind design block two, combining all three design blocks was a great challenge. Because modules cannot be called inside an if statement or case statement, we had to pick apart each module so that they would work together. Global registers for each input and output were declared, and each module had its own register for each module. For example, HEX0 had a register for modules one, two, and three. Register selection would be used by switching SW[9:8]. If SW[9:8] = 2’b00, module one would be used. If SW[9:8] = 2’b01, module two would be used. If SW[9:8] = 2’b1x, module three would be used. The only testbench file used in this lab report was created to test the integration of all three design blocks. The included test bench output screen capture shows LEDR outputs based on KEY and SW inputs.

**Conclusion**

Overall, this lab was a great start to learning the complexities of Verilog and the DE10 board. Because Verilog is a hardware description language, any line of code can be executed at any point. This was initially hard to process, as we were used to C and JavaScript, which executes code in a sequential manner. This lab also taught us about Quartus, which is an imperfect programming tool. Making sure that the correct assignment file with the correct assignments was very challenging at first. Simulation captures for each design block are included and show how the inputs are affected by an output. In the end, our team learned invaluable knowledge that will surely progress us in the field of computer engineering.