Intro to Digital Logic, Lab 2
Digital Design using FPGAs

Lab Objectives

Now that you know how to develop Verilog designs and load them into the DE1 SoC board, we can now start looking at more complex designs.

Implementing designs directly in schematics or structural (gate-level) Verilog can give you the best control, and often the smallest designs. But, sometimes it can be a real pain to optimize all the way down at that level. An alternative is high-level (RTL) Verilog, where you tell the CAD tools what you what the output to look like, and it automatically does the Boolean Algebra for you!

Design Problem – Multi-level logic on the DE-1 FPGA.

Before going through the lab, you should review the Verilog tutorial on the website up through (but not including) Register Transfer Level (RTL) Code.

In order to speed up processing of returns at Nordstrom's, the customer service department wants an electronic detector device. Its goal is to both determine those items that have been discounted, so that the proper rebate can be calculated, as well as help find shoplifters returning their ill-gotten gains.

There are six products being sold. The UPC code for each is shown, as well as whether it was ever on sale (i.e. sold at a discounted price), and whether it is expensive, and thus is marked when sold.

Item Name	UPC Code	Discounted?	Expensive?
Shoes	0 0 0	No	Yes
Costume Jewelry	0 0 1	No	No
Christmas Ornament	0 1 1	Yes	No
Business Suit	1 0 0	No	Yes
Winter Coat	1 0 1	Yes	Yes
Socks	1 1 0	Yes	No

Note that since there are only 6 items, not all UPC codes are used. The behavior of your circuit for these situations is unimportant (i.e. Don't Care).

You will be given the UPC code of the item under test (signals "U", "P", and "C" for simplicity), and a detector will check for a secret mark ("M"). Your circuit should have one "discounted" light that lights up whenever a discounted item's UPC code is applied, as well as a "stolen" light that lights up whenever a theft is detected.

Nordstrom's has a special method for finding shoplifters. Whenever they sell an expensive item, they put a secret mark onto it. Thus, expensive items that are purchased are specially marked, while stolen expensive items will not be so marked (inexpensive items are never marked, since it is too expensive to mark everything sold). When there is a return, we want to make sure someone didn't steal the item, then return the stolen item for cash.

With the rules given, there are four cases for the stolen light logic to consider:

- An expensive item with the mark is not stolen.
- An expensive item without the mark is stolen.
- A non-expensive item without the mark is not stolen.
- A non-expensive item with the mark will never occur, so is a Don't Care.

For this circuit, create a design by hand for each of the outputs. **You will need to use K-Maps or Boolean Algebra to optimize the design (K-Maps will likely be the best way).** Then, write the corresponding code in Verilog in Quartus II and simulate it. Finally, download the design to the FPGA, and use the switches and lights on the board to connect to the circuit.

Once you have the design working on the FPGA, **draw the schematic of the design**. This will be used for evaluating the quality of your design.

Please use switch Sw9, Sw8, Sw7 for U, P, C, respectively, and Sw0 for the secret Mark.

Note that with this, and all labs, keep your files when you are done with the lab – they will often get reused in subsequent labs!

Assigned Task – Seven-segment display

In the class notes we presented a seven-segment display driver. RTL code for that seven-segment display is given below. Add the code below into a new System Verilog file in the same project as the first design problem. Then, create a new module that instantiates the seg7 code twice – one taking an input from SW3 – SW0 and displaying on Hex0 of the board, another taking input from SW7 – SW4 and displaying on Hex1 of the board.

```
module seg7 (bcd, leds);
   input  logic  [3:0] bcd;
   output logic  [6:0] leds;

   always_comb begin
      case (bcd)
         //          Light: 6543210
         4'b0000: leds = 7'b0111111; // 0
         4'b0001: leds = 7'b0000110; // 1
         4'b0010: leds = 7'b1011011; // 2
         4'b0011: leds = 7'b1001111; // 3
         4'b0100: leds = 7'b1100110; // 4
         4'b0101: leds = 7'b1101101; // 5
         4'b0110: leds = 7'b1111101; // 6
         4'b0111: leds = 7'b0000111; // 7
         4'b1000: leds = 7'b1111111; // 8
```

```
            4'b1001: leds = 7'b1101111; // 9
            default: leds = 7'bX;
        endcase
    end
endmodule
```

Note that the 7-segment display on the DE-1 is ACTIVE LOW. That means putting a FALSE on the wire makes it light up, while a TRUE means that light is off. You will have to adjust your design accordingly.

Design Problem – UPC code to display

Review the Verilog Tutorial on the website up to (but not including) Sequential Logic.

In the first part of the lab, we built a system that took in a UPC code and output whether the item was on sale and whether it was stolen. However, Nordstrom has found that people are changing the UPC sticker on their stuff. To combat that, they'd like you to add a display on Hex5 – Hex0 that describes each product when it is entered – if the description is different from the product actually entered, they know someone is trying to cheat! Note that, since Nordstrom pays their employees embarrassingly little, they tend to not be very careful, so the Hex display should be as simple as possible.

We will use the same Items, UPC codes, On Sale values, and Expensive markings as before as seen in the table above.

Determine the Hex displays for each item. You can use any or all of the lights on the hex display you choose, upper and lower case, pictograms, whatever. So, for example you could put in "Dress" as an item and have the hex display show "drESS". Note that dresses are not often returned and are not on the list of items!

Once you have figured out what you'll display, create a high-level design for the circuit. It will be similar to the seg7 module, with 3 inputs (U, P, and C), but up to 6 7-bit outputs (for each of the 7seg displays). It will be written in RTL. Simulate it in ModelSim, then hook it to the switches and lights of your board to make sure it works. Finally, connect the proper inputs and outputs to your module from the first part of lab. It should hook them both together, so the system simultaneously computes the HEX display, and the Sale and Stolen lights. Test and debug with ModelSim, then load onto your board.

ModelSim Tip: When you put your entire design together and simulate in ModelSim, you'll probably need some help organizing all your signals. A couple tips:

1.) ModelSim can have signals from multiple modules displayed at the same time. Simply select in the left pane the module whose signals you want to display, then drag the signals you want from the middle pane to the waveform window.
2.) To order the signals, you can click and drag names in the waveform window.
3.) To organize signals for each module, highlight all of the signal names related to one module in the waveform window, right-click on one of the signal names, and select "Group". Give it a memorable name, then hit okay. You can now move the signals as a group, and hide/expose them easily.
4.) Once you have organized signals the way you like, remember to save the formatting into the <modulename>_wave.do file.

Assigned Task – Don't Cares

Your design has outputs for only 6 of the 8 possible UPC codes. For the other two cases, a line such as "default: LEDs = 7'bx;" tells Quartus II that it can treat these cases as a Don't Care condition (if you didn't do this, go back and correct it to do so). Test your design on the circuit board and record the pattern it shows for these Don't Care conditions.

Lab Demonstration/Turn-In Requirements

You will be graded 100 points on correctness, style, testing, test coverage, etc. Your bonus is using as few logic gates as possible and quality of results. Each gate (an individual Inverter, an individual AND, and an individual OR) appearing in your hand-drawn circuit diagram costs the same, and gates (other than inverters) can have as many inputs as you want. Unlike lab #1, inverters on the inputs DO count towards your total; each and every gate on your schematic will count as 1 gate. Note that you can only use standard gates (AND, OR, NAND, NOR, NOT, XOR, XNOR) – no AND gates with one input inverted, etc. In this lab "quality of results" means how nice/creative your display is. Results that are particularly well done, pretty, amusing, or otherwise makes your TA laugh or applaud will get more points.

A TA needs to "Check You Off" for each of the tasks listed below.

- To turn in as a pdf on canvas:
    - The K-maps or Boolean simplification you did to create your design for the implementation showing if the item is discounted and/or stolen based on the UPC and Mark.
    - Your circuit diagram of the implementation showing if the item is discounted and/or stolen based on the UPC and Mark.
    - The Verilog code you produced for your design.
    - A drawing of what the circuit does when given each of the unused UPC codes (from Assigned Task – Don't Cares).
- Demonstrate the 2x7seg in ModelSim.
- Demonstrate the 2x7seg on the DE1 board.
- Demonstrate your full UPC code circuit working in simulation under ModelSim.
- Demonstrate your full UPC code circuit working on the DE1 board.
- Tell the TA how many hours (estimated) it took to complete this lab, including reading, planning, design, coding, debugging, testing, etc. Everything related to the lab (in total).