Intro to Digital Logic, Lab 2
Digital Design using FPGAs

Lab Objectives

1. Developing more complex designs on the DE1_SoC board.
2. Using RTL Verilog, where you identify what you want the output to look like and the Boolean algebra is automatically done for you.

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

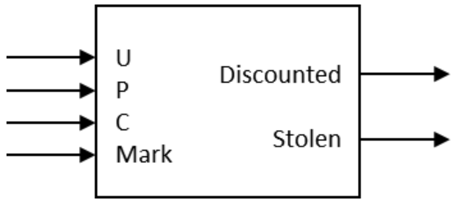
In order to speed up the processing of returns at Nordstrom's, the customer service department wants an electronic detector device. Its goal is to both determine those items that have been discounted so that the proper rebate can be calculated, as well as help find shoplifters returning their ill-gotten 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. **Name the folder of your Quartus project lab2a**. 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 should be a circuit schematic that has inputs U, P, and C, logic gates, and the two outputs "Discounted?" and "Expensive?"). This will be used for evaluating the quality of your design.

Note that for all design problems, you will be graded on correctness, style, testing, etc. For this lab, the bonus goal is to use as FEW logic gates as possible. 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. Note that you can only use standard gates (AND, OR, NAND, NOR, NOT, XOR, XNOR) – no AND gates with one input inverted, etc.

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

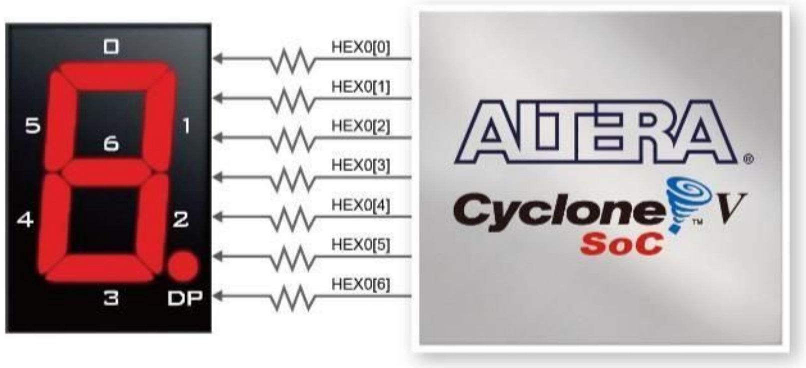
Task 2: Seven-segment display

In the class notes, we presented a seven-segment display driver. RTL code for that seven-segment display is given below. **Create a copy of your lab2a project and call it lab2b**, then enter the code below into a new System Verilog file. Then, create a new module that instantiates the seg7 code taking input from SW9 – SW6 and displaying on Hex0 the corresponding digit. Simulate in ModelSim.

```
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
```

As mentioned in lab1, 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. (The figure below gives you a correct mapping of the HEX digits to its corresponding positions on the display.)



Finally, create a new module that hooks task1 with task2 together, so the system simultaneously computes the HEX0 display and the Sale and Stolen lights. Since we only have six items, we only need 3 switches for UPC. Accordingly set SW9 to 0 and let SW8-SW6 represent the UPC. You still need SW0 for the secret mark to compute the 'stolen' and 'sale' lights but it doesn't have anything to do with the HEX0 display. Test and debug with ModelSim, then load onto your board. (Because you have set SW9 to 0 in this task, the observable HEX patterns should only range from 0 to 7).

(Extra Credit) Task 3: Design Problem – UPC code to display

In task1 we built a system that took in a UPC code and output whether the item was on sale and whether it was stolen. A neighboring store, Fred's House of Useful Stuff, wants a similar system but 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 Fred pays his employees embarrassingly little, they tend to be not the brightest folks, so the Hex display should be as simple as possible.

Since Fred doesn't want to pay for a redesign of the circuit for task1, we will use the same UPC codes, On Sale values, and Expensive markings as before BUT, Fred is willing to sell whatever products you will like him to (Hint: pick products that make good Hex displays!), though the expensive ones must use UPC codes designated "expensive", and the inexpensive ones must use UPC codes designated "inexpensive". BE CREATIVE! And you MUST sell 6 items.

Determine your items and Hex displays. 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 Shoes" as an item and have the hex display show "ShOE". Note that some letters are not representable in HEX. For example, i or t, so you can choose to avoid those.

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). Simulate it in ModelSim, then hook it to the switches and lights of your board to make sure it works. Create a module that combines task1 with this task such that 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 of 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.

Lab Demonstration and Submission Requirements

- Demonstrate all tasks to the TA.
- Submit a short lab report (about 3 pages) that should include 3 main sections, detailed below.

Procedure

- Describe how you approached the problem and include any truth tables or schematics.
- Include screenshots of the code for tasks 1 and 2 (task 3 if applicable).

Results

- Include screenshots of ModelSim results
- Describe what you tested in the simulation, and what the results in the screenshot show
- Give a brief overview of the finished project, compared to what was asked

Problems Faced & Feedback

- Describe any issues you had while completing the lab, and how/if you were able to overcome them.
- Include tips and tricks that you found out about that may be useful to you in future labs.

    - Give any other positive or negative feedback you have on the lab
- Submit the SystemVerilog files (files with extension .sv) of task 3. Make sure to follow the commenting guide provided
- Submit your report and programs to Canvas. No hard copies.