Bert Zhao
EE 271

Abstract: The purpose of this lab was to continue exploring the fundamentals of the Quartus Prime Lite programming software, verilog, and the FPGA development board. This lab focuses more extensively on the programming of the development board. The tasks of the lab build sequentially off each other to show how we can use the same set of inputs to control multiple sets of outputs. More specifically in task two, we will see how the same set of switches can be easily programmed to control both a hex display and led display.

Introduction: The core challenge of this lab is to design a circuit that keeps track of a store's inventory of six items in the form of UPC codes. This helps track if an item is on sale, as well as if an item is expensive. This system is designed to track expensive items using a mark to determine if they were stolen. In the first task we create the foundational system that determines whether an item is on sale or stolen. In the second task, we add on a hex display to display the corresponding decimal digit with each UPC code. In the third task, we take advantage of all the hex displays to show actually show the full name of the items being sold.

Materials:

- Altera Terasic DE1-SoC Board with power and USB cables
- PC with Altera Quartus Prime Lite installed

Procedure:

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

The first part of this lab requires designing the core circuit that controls the system that keeps track of the status of the items. This involves four inputs: U, P, C, and the mark. There are also two outputs which note if an item is discounted or stolen. Items which are expensive must have a mark on them otherwise they are considered stolen. Inexpensive items will never have a mark. Designing the circuit involved a four variable truth table and k-maps for simplification to the least gates design. The design uses switches 9-7 for the UPC, switch 0 for the mark, and leds 1 and 0 for discount and stolen respectively.

Truth Table:

U	P	C	Mark	Discount	Stolen
0	0	0	0	0	1
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	x
0	1	0	0	x	x
0	1	0	1	x	x
0	1	1	0	1	0
0	1	1	1	1	x
1	0	0	0	0	1

1	0	0	1	0	0
1	0	1	0	1	1
1	0	1	1	1	0
1	1	0	0	1	0
1	1	0	1	1	x
1	1	1	0	x	x
1	1	1	1	x	x

K-maps:

Discount = P + *UC*

C mark \ U P	**00**	**01**	**11**	**10**
00	0	x	1	0
01	0	x	1	0
11	0	1	*x*	*1*
10	0	1	*x*	*1*

Stolen = ~P~C~M + U~P~M

C mark \ U P	**00**	**01**	**11**	**10**
00	1	x	0	1
01	0	x	x	0
11	x	x	x	0
10	0	0	x	1

Schematic Diagrams:

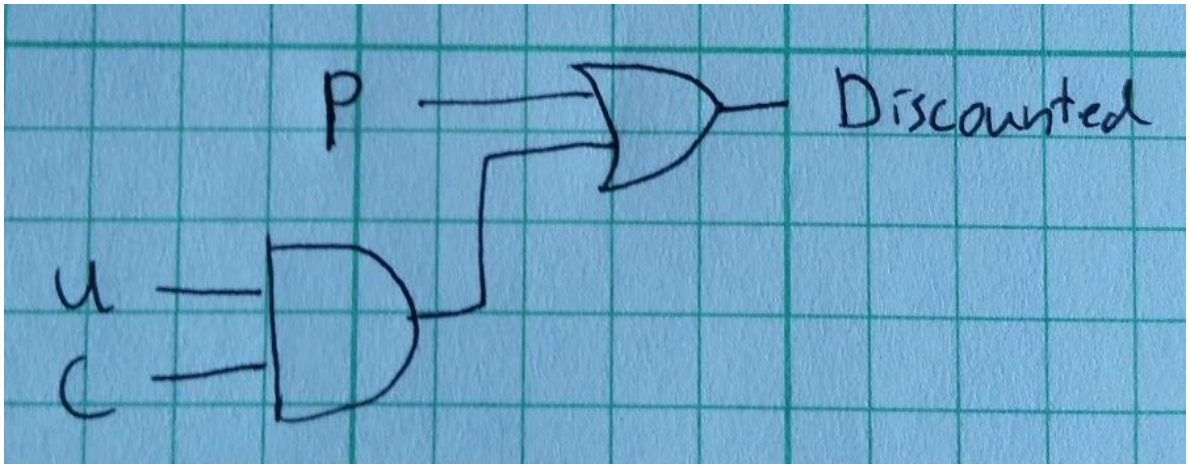

Figure 1: schematic diagram for discount

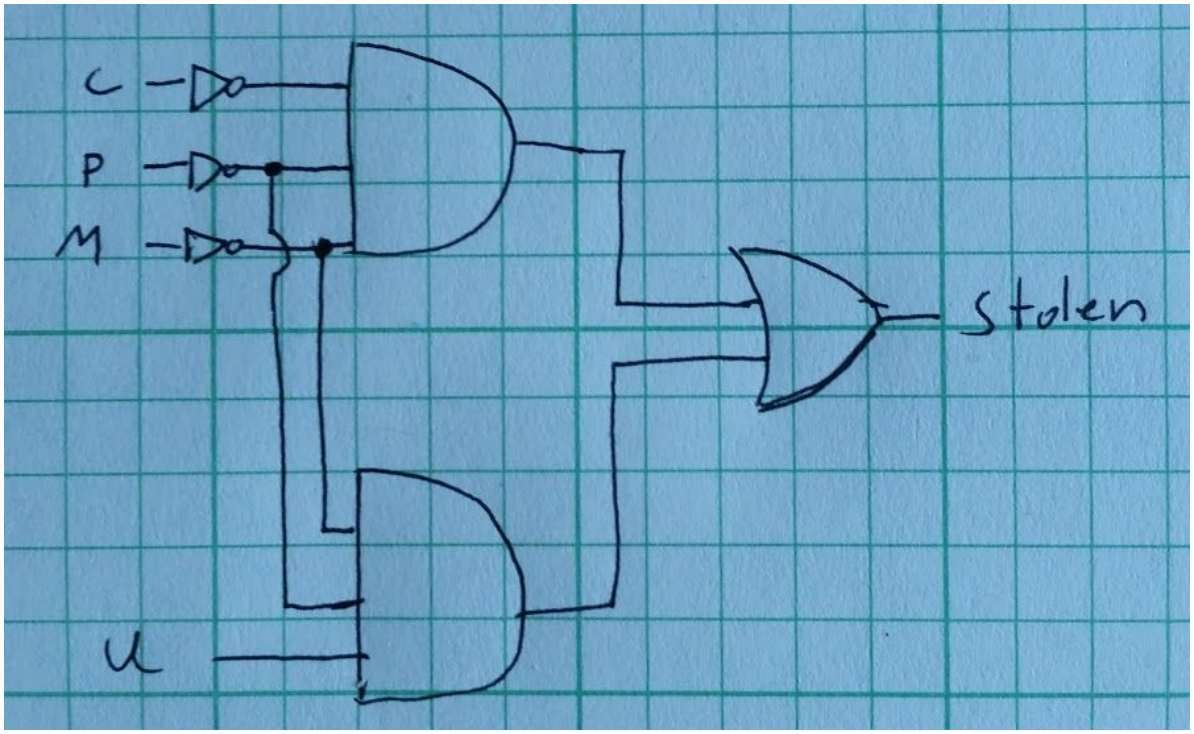

Figure 2: schematic diagram for stolen

Part 2: Seven-segment display

The second part of the lab is a continuation of the first were now we also add a hex display to the outputs. We first create a test module for the seven segment display, testing all the possible single digit outputs on the hex display. Note that the hex displays are active low so we need to adjust the provided seg7 code to properly associate each binary number 0-9 with its proper digit on the hex display. After testing the provided seg7 module, we need to create our own hex display outputs for our UPC code inputs. We can create a similar module as the seg7 with three inputs and the hex displaying the digits 0-7. The only difference between this part and part one is that we are now using switches 8-6 for the UPC codes.

Part 3: Design Problem – UPC code to display

The final part of this lab required just a small modification to part two. Instead of showing the single digit representation of the UPC code on a hex display, we use up to all six hex displays to show the full name of the item. In input/output relationships of the UPC and mark with the discount and stolen, but we can choose the names of any six items we want to associate with the UPC codes.

Results:

The primary purpose of using k-maps was to find the minimum gate design for the marker circuit. The final complete system could be made with five gates and three inverter. All parts of this lab were also tested with ModelSim prior to programming the development board.

ModelSim Screenshots:

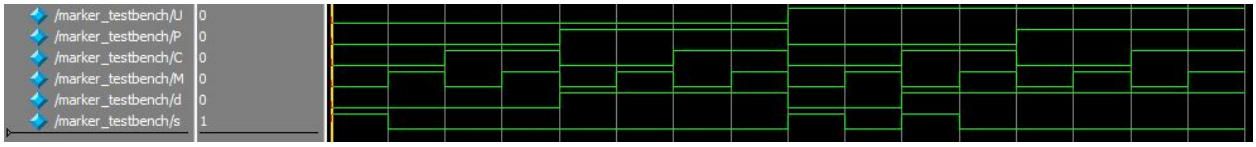

Figure 3: marker circuit ModelSim screenshot (Task 1)

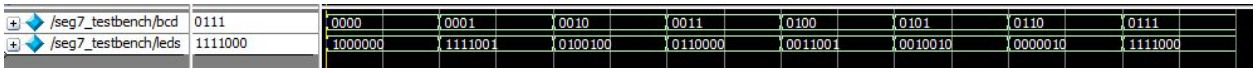

Figure 4: seg7 ModelSim screenshot (Task 2)

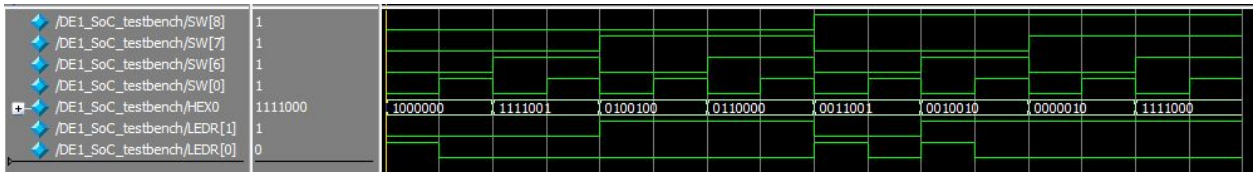

Figure 5: full system with marker and display ModelSim screenshot (Task 2)

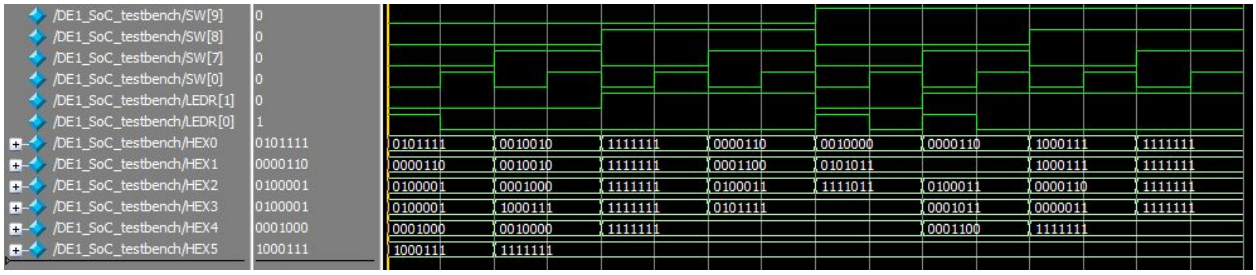

Figure 6: full system ModelSim screenshot (Task 3)

Analysis and Conclusion:

Overall, this lab provided more detail and practice with designing a circuit and then programming it to the development board. The tasks of this lab were also very sequential to show how we could use multiple modules to control multiple outputs with the same set of inputs.

Appendix:

```
module marker(U, P, C, M, d, s);
   input logic U, P, C, M;
   output logic d, s;

   assign d = P | (U & C); // discount
   assign s = (~P & ~C & ~M) | (U & ~P & ~M); // stolen

endmodule

module marker_testbench();

   logic U, P, C, M, d, s;

   marker dut(U, P, C, M, d, s);

   integer i;
   initial begin

      for(i = 0; i < 2**4; i++) begin
         {U, P, C, M} = i; #10;
      end
   end

endmodule
```

Code 1: marker (All Tasks)

```
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'b1000000; // 0
         4'b0001: leds = 7'b1111001; // 1
         4'b0010: leds = 7'b0100100; // 2
         4'b0011: leds = 7'b0110000; // 3
         4'b0100: leds = 7'b0011001; // 4
         4'b0101: leds = 7'b0010010; // 5
         4'b0110: leds = 7'b0000010; // 6
         4'b0111: leds = 7'b1111000; // 7
         4'b1000: leds = 7'b0000000; // 8
         4'b1001: leds = 7'b0010000; // 9
         default: leds = 7'bX;
      endcase
   end
endmodule


module seg7_testbench();
   logic [3:0] bcd;
   logic [6:0] leds;

   seg7 dut(.bcd, .leds);

   integer i;
   initial begin
      for(i = 0; i < 2**3; i++) begin
         bcd[3:0] = i; #10;
      end
   end
endmodule
```

Code 2: seg7 (Task 2)

```
module display(U, P, C, leds);
   input logic U, P, C;

   output logic [6:0] leds;

   always_comb begin
      case ({U, P, C})
            // Light: 6543210
         3'b000: leds = 7'b1000000; // 0
         3'b001: leds = 7'b1111001; // 1
         3'b010: leds = 7'b0100100; // 2
         3'b011: leds = 7'b0110000; // 3
         3'b100: leds = 7'b0011001; // 4
         3'b101: leds = 7'b0010010; // 5
         3'b110: leds = 7'b0000010; // 6
         3'b111: leds = 7'b1111000; // 7
         default: leds = 7'bX; // don't cares
      endcase
   end
endmodule

module display_testbench();
   logic U, P, C;
   logic [6:0] leds;

   display dut(.U, .P, .C, .leds);

   integer i;
   initial begin
      for(i = 0; i < 2**3; i++) begin
         {U, P, C} = i; #10; // test all possibilities of UPC
      end
   end
endmodule
```

Code 3: display (Task 2)

```
module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);

   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;

   input logic [3:0] KEY;
   input logic [9:0] SW;

   // marker module for led outputs for discount and sale
   marker UPC0 (.U(SW[8]), .P(SW[7]), .C(SW[6]), .M(SW[0]), .d(LEDR[1]), .s(LEDR[0]));

   // display module for hex digit representation of UPC
   display UPC1 (.U(SW[8]), .P(SW[7]), .C(SW[6]), .leds(HEX0));

   // don't need other hex displays
   assign HEX1 = 7'b1111111;
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;

endmodule

module DE1_SoC_testbench();

   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic [9:0] LEDR;
   logic [3:0] KEY;
   logic [9:0] SW;

   DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW);

   integer i;
   initial begin

   // don't need switches 1 through 5 or switch 9
   SW[9] = 1'b0;
   SW[5:1] = 1'b0;
      for(i=0; i<2**4; i++) begin
         {SW[8:6], SW[0]} = i; #10; // tests all possibilities of UPS and mark
      end
   end
endmodule
```

Code 4: DE1_SoC (Task 2)

```
module display(U, P, C, led5, led4, led3, led2, led1, led0);
   input logic U, P, C;

   output logic [6:0] led5, led4, led3, led2, led1, led0;

   always_comb begin
      case ({U, P, C}) // cases relating item to UPC
         //          Light: 6543210
         3'b000: begin
                 led5 = 7'b1000111; // L
                 led4 = 7'b0001000; // A
                 led3 = 7'b0100001; // d
                 led2 = 7'b0100001; // d
                 led1 = 7'b0000110; // E
                 led0 = 7'b0101111; // r
                 end

         3'b001: begin
                 led5 = 7'b1111111;
                 led4 = 7'b0010000; // g
                 led3 = 7'b1000111; // L
                 led2 = 7'b0001000; // A
                 led1 = 7'b0010010; // s
                 led0 = 7'b0010010; // s
                 end

         3'b011: begin
                 led5 = 7'b1111111;
                 led4 = 7'b1111111;
                 led3 = 7'b0101111; // r
                 led2 = 7'b0100011; // o
                 led1 = 7'b0001100; // p
                 led0 = 7'b0000110; // E
                 end

         3'b100: begin
                 led5 = 7'b1111111;
                 led4 = 7'b1111111;
                 led3 = 7'b0101111; // r
                 led2 = 7'b1111011; // i
                 led1 = 7'b0101011; // n
                 led0 = 7'b0010000; // g
                 end

         3'b101: begin
                 led5 = 7'b1111111;
                 led4 = 7'b0001100; // p
                 led3 = 7'b0001011; // h
                 led2 = 7'b0100011; // o
                 led1 = 7'b0101011; // n
                 led0 = 7'b0000110; // E
                 end

         3'b110: begin
                 led5 = 7'b1111111;
                 led4 = 7'b1111111;
                 led3 = 7'b0000011; // b
                 led2 = 7'b0000110; // E
                 led1 = 7'b1000111; // L
                 led0 = 7'b1000111; // L
                 end
```

```
            default: begin // don't cares
                        led5 = 7'b1111111;
                        led4 = 7'b1111111;
                        led3 = 7'b1111111;
                        led2 = 7'b1111111;
                        led1 = 7'b1111111;
                        led0 = 7'b1111111;
                        end
         endcase
      end

   endmodule

   module display_testbench();
      logic U, P, C;
      logic [6:0] led5, led4, led3, led2, led1, led0;

      display dut(.U, .P, .C, .led5, .led4, .led3, .led2, .led1, .led0);

      integer i;
      initial begin
         for(i = 0; i < 2**3; i++) begin
            {U, P, C} = i; #10; // tests all possibilities of UPC
         end
      end
   endmodule
```

Code 5: display (Task 3)

```
module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;

   input logic [3:0] KEY;
   input logic [9:0] SW;

   // marker module for led outputs for discount and sale
   marker UPC0 (.U(SW[9]), .P(SW[8]), .C(SW[7]), .M(SW[0]), .d(LEDR[1]), .s(LEDR[0]));

   // display module for hex displays for name of item for sale
   display UPC1 (.U(SW[9]), .P(SW[8]), .C(SW[7]), .led5(HEX5), .led4(HEX4), .led3(HEX3), .led2(HEX2), .led1(HEX1), .led0(HEX0));

endmodule

module DE1_SoC_testbench();

   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic [9:0] LEDR;
   logic [3:0] KEY;
   logic [9:0] SW;

   DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW);

   integer i;
   initial begin
   SW[6:1] = 1'b0; // don't need switches 1 through 6
      for(i=0; i<2**4; i++) begin
         {SW[9:7], SW[0]} = i; #10; // tests all possibilities of UPS and mark
      end
   end
endmodule
```

Code 6: DE1_SoC (Task 3)