CSE 369 Lab 4

High-Level Verilog

Isaac Wu
2360957
October 30, 2024

1. Item Table

Item Name	UPC	Discounted?	Expensive?
Banana	0 0 0	No	Yes
Bottle	0 0 1	No	No
Needle	0 1 1	Yes	No
Pencil	1 0 0	No	Yes
Spring	1 0 1	Yes	Yes
Sponge	1 1 0	Yes	No

2. ModelSim Simulations

(a) Double 7-Segment Display

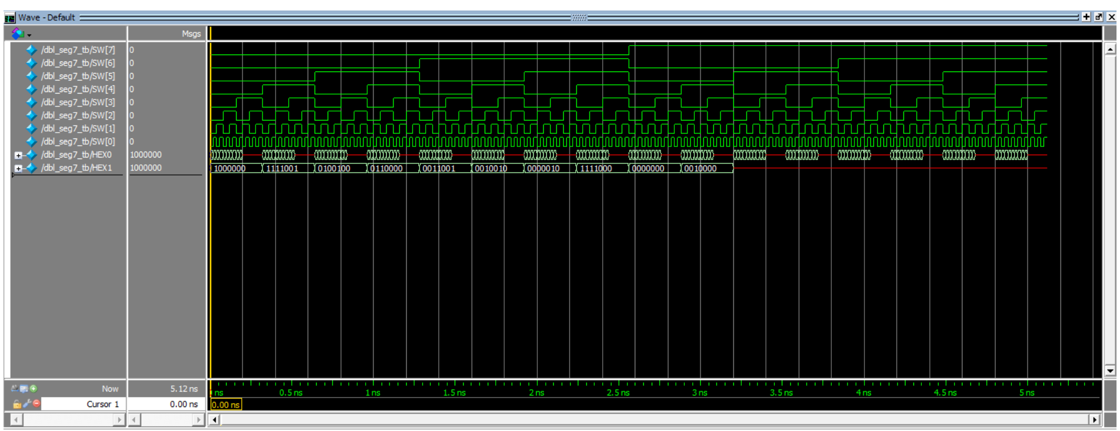
Here is the code for my double 7-segment display. I defined logic variables for the output of the seg7 module calls to explicitly invert them. The HEX outputs on our DE1s are active low, necessitating the inversion here. I called seg7 twice, once for each digit we want to output with their defined range of switches (3-0 for the first, 7-4 for the second). I then assigned the inverses of the outputs to their corresponding HEX displays on the DE1 board.

```
module dbl_seg7 (
   output logic [6:0] HEX0, HEX1,
   input  logic [7:0] SW
   );

   logic[6:0] inv_d1, inv_d2;

   seg7 d1 (.bcd(SW[3:0]), .leds(inv_d1));
   seg7 d2 (.bcd(SW[7:4]), .leds(inv_d2));

   assign HEX0 = ~inv_d1;
   assign HEX1 = ~inv_d2;

endmodule  // dbl_seg7
```

This is the code for the double 7-segment display test benchmark used to ensure correct functionality from my code. This benchmark will run my dbl_seg7 module with all 256 combinations of switch inputs.

```
module dbl_seg7_tb();
  logic [7:0] SW;
  logic [6:0] HEX0, HEX1;

   // Instantiate device under test
   dbl_seg7 dut (.HEX0, .HEX1, .SW);

   // Try all switch combinations
   integer i;
   initial begin
     SW[7:0] = 1'b0;
     for(i = 0; i < 256; i++) begin
       SW = i; #20;
     end
   end
endmodule   // dbl_seg7_tb
```

This is what my wave diagram looks like. The top 8 waves show the 8 switches alternating to test all 256 combinations. The bottom two waves are the HEX outputs, with HEX0 corresponding to switches 3-0 and HEX1 corresponding to switches 7-4. The binary sequences shown on the wave diagram correspond to the pre-defined display of the switch combination number. Because we are only displaying numbers 0-9, we have some switch combinations we don't care about, which are the red lines shown in the HEX outputs.



(b) Fred's Pawn Shop

Here is the code for Lab 4, or Fred's Pawn Shop. I call lab3 to calculate if the item was on sale or stolen using the input switches and outputs to two LEDs. I then call my module display to display the name of the item on the HEX display. I only need the switches corresponding to the UPC number for the display, so I slice the switches array for this input.

```
module lab4 (
    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
    );

    lab3 sale_stolen (.LEDR, .KEY, .SW);
    display item (.SW(SW[3:1]), .*);

endmodule   // lab4

```

Here is a revised version of my code from Lab 3, which calculates whether an item was on sale or stolen using the item's UPC code and a Mark input from switches and outputs to two LEDs. I removed the code that would set the HEX displays to their default value (off) since I would be using them to display the item name and the assignment would conflict. I also swapped the LED outputs of Discounted and Stolen to better match the situation.

```
/* Top-level module that defines the I/Os for the DE1-SoC board
 * and the circuit behavior.
 */
module lab3 (
    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
    );

    logic U, P, C, Mark, Discounted, Stolen;
    assign U = SW[3];
    assign P = SW[2];
    assign C = SW[1];
    assign Mark = SW[0];

    assign Discounted = P | (U & C);
    assign Stolen = (~U & ~C & ~Mark) | (U & ~P & ~Mark);

    assign LEDR[1] = Discounted;
    assign LEDR[0] = Stolen;

endmodule  // lab3

```

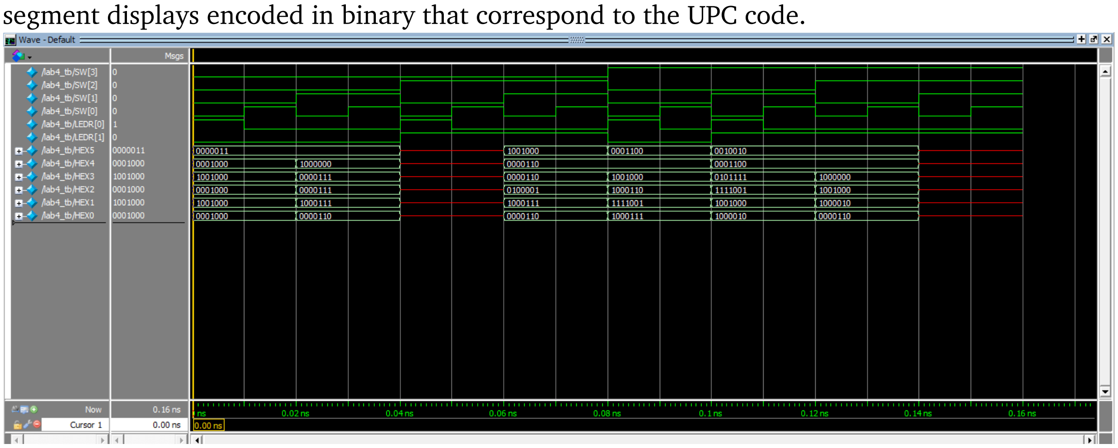
Here is my display code. This simple module takes in the UPC code of the item and uses a switch-case statement to then assign predetermined values to the HEX displays to display the name of the item. With 8 UPC codes and 6 items, there are two codes that we don't care about, and those are handled in the default case.

```
module display(
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
   input  logic [2:0] SW
   );

   always_comb
      case (SW)
         // Banana
         3'b000: begin
           HEX5 = 7'b0000011;
           HEX4 = 7'b0001000;
           HEX3 = 7'b1001000;
           HEX2 = 7'b0001000;
           HEX1 = 7'b1001000;
           HEX0 = 7'b0001000;
           end
         // Bottle
         3'b001: begin
           HEX5 = 7'b0000011;
           HEX4 = 7'b1000000;
           HEX3 = 7'b0000111;
           HEX2 = 7'b0000111;
           HEX1 = 7'b1000111;
           HEX0 = 7'b0000110;
           end
         // Needle
         3'b011: begin
           HEX5 = 7'b1001000;
           HEX4 = 7'b0000110;
           HEX3 = 7'b0000110;
           HEX2 = 7'b0100001;
           HEX1 = 7'b1000111;
           HEX0 = 7'b0000110;
           end
         // Pencil
         3'b100: begin
           HEX5 = 7'b0001100;
           HEX4 = 7'b0000110;
           HEX3 = 7'b1001000;
           HEX2 = 7'b1000110;
           HEX1 = 7'b1111001;
           HEX0 = 7'b1000111;
           end
         // Spring
         3'b101: begin
           HEX5 = 7'b0010010;
           HEX4 = 7'b0001100;
           HEX3 = 7'b0101111;
           HEX2 = 7'b1111001;
           HEX1 = 7'b1001000;
           HEX0 = 7'b1000010;
           end
         // Sponge
         3'b110: begin
           HEX5 = 7'b0010010;
           HEX4 = 7'b0001100;
           HEX3 = 7'b1000000;
           HEX2 = 7'b1001000;
           HEX1 = 7'b1000010;
           HEX0 = 7'b0000110;
           end
         default: begin
           HEX5 = 7'bX;
           HEX4 = 7'bX;
           HEX3 = 7'bX;
           HEX2 = 7'bX;
           HEX1 = 7'bX;
           HEX0 = 7'bX;
        end
      endcase

module   // display
```

This is the code for the test benchmark used to ensure correct functionality from Fred's Pawn Shop. This benchmark will run my lab4 module with all 16 combinations of switch inputs, representing all 8 UPC codes and the Mark input.

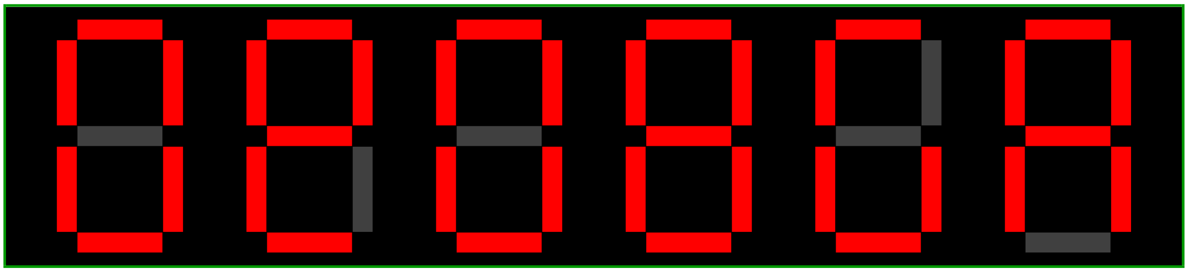
```
module lab4_tb();
  logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
  logic [9:0] LEDR;
  logic [3:0] KEY;
  logic [9:0] SW;

  // instantiate device under test
  lab4 dut (.*);

  // test input sequence - try all combinations of inputs
  integer i;
  initial begin
    SW[9:4] = 1'b0;
    for(i = 0; i < 16; i++) begin
      SW[3:0] = i; #10;
    end
  end
endmodule  // lab4_tb
```

This is what the resulting wave diagram looks like. The top 4 waves show the 4 switches alternating to test all 16 combinations. The middle two waves are the LED output, with LEDR[0] corresponding to if a product is Discounted, and LEDR[1] if a product was stolen. The bottom 6 waves are the HEX displays, that show, in order, the pre-programmed letters codes for the 7-segment displays encoded in binary that correspond to the UPC code.

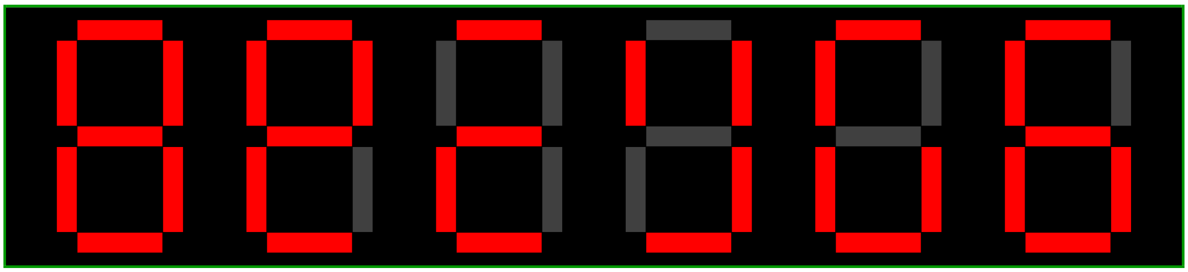


3. Unused UPC 7-Segment Display Outputs

(a) UPC = 010



(b) UPC = 111



4. Misc.

How many hours (estimated) it took to complete this lab in total, including reading, planning, designing, coding, debugging, and testing.

It took around 5 hours to complete this lab.