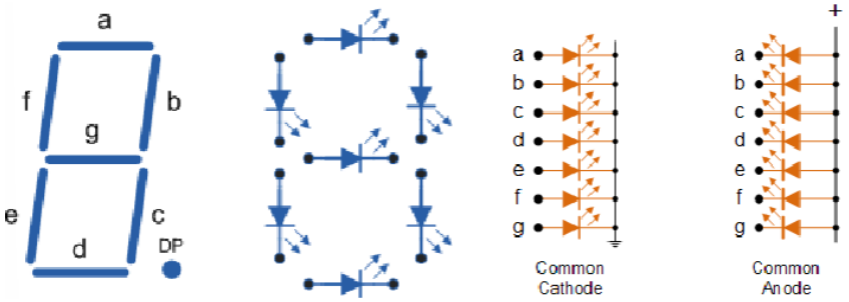
LABORATORY_2

Combinatorial Logic Using EDA Playground

Hannah Hwang | SID : 862419233 | CS120A Section 025
Adithya Chander | SID : 862429692 | CS120A Section 025
Calvin Hong | SID : 862416275 | CS120A Section 025

Overview

In lab 2 we explored the logic behind 7-segment LED displays (a display system with 7 independent LEDs used to represent numbers from 0-9). For the lab, it was stated that the LEDs are common Anode, so a=0 represents ON and a=1 represents OFF and we were provided with the following diagram:



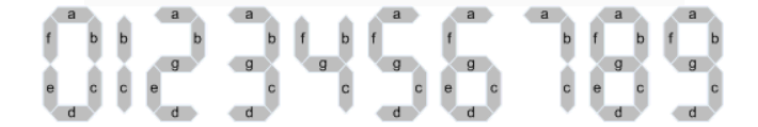
We then implemented the behavioral model of a BCD-to-7seg decoder as well as a test bench in EDA playground with the provided template in the lab manual. Within the design file, we added 9 additional cases within the case bundle for when the leds would represent the numbers 1-9 using the values provided in the truth table [FIGURE 1]. For the testbench, we also added 9 additional test cases. After running our finished code, we got the waveform shown in [FIGURE 2].

Analysis

Truth Table |

Individual Segments							Display
a	b	c	d	e	f	g	
×	×	×	×	×	×		0
	×	×					1
×	×		×	×		×	2
×	×	×	×			×	3
	×	×			×	×	4
×		×	×		×	×	5
×		×	×	×	×	×	6
×	×	×					7

Individual Segments							Display
a	b	c	d	e	f	g	
×	×	×	×	×	×	×	8
×	×	×	×		×	×	9
×	×	×		×	×	×	A
		×	×	×	×	×	b
×			×	×	×		C
	×	×	×	×		×	d
×			×	×	×	×	E
×				×	×	×	F



[FIGURE 1]
screenshot of truth table that was used to implement decoder output design, provided by the provided LED display information page

Records

Link to EDA Playground (our code)

design.sv

```
// Code your design here
`timescale 1ns / 1ps
module bcd_to_7led_bh (
input wire sw0 , // Switches
input wire sw1 ,
input wire sw2 ,
input wire sw3 ,
output reg a , // LED segments
output reg b ,
output reg c ,
output reg d ,
output reg e ,
output reg f ,
output reg g
);

// Internal wire
wire [3:0] bundle ;
assign bundle = {sw3,sw2,sw1,sw0 } ;

always @(*) begin

        // Setting the segments signals (Initialize all to off/1)
  a = 1'b1 ;
  b = 1'b1 ;
  c = 1'b1 ;
  d = 1'b1 ;
  e = 1'b1 ;
  f = 1'b1 ;
  g = 1'b1 ;

case ( bundle )

  4'b0000 : begin // 0
    a = 1'b0 ;
    b = 1'b0 ;
    c = 1'b0 ;
```

```
    d = 1'b0 ;
    e = 1'b0 ;
    f = 1'b0 ;
    g = 1'b1 ; //(Don’t need to explicitly state that g is off here since
                                // it is initialized to off already, but it doesn’t hurt)
  end
// Your code goes here for the other 8 numbers (1-9)
  4'b0001 : begin // 1
    a = 1'b1 ;
    b = 1'b0 ;
    c = 1'b0 ;
    d = 1'b1 ;
    e = 1'b1 ;
    f = 1'b1 ;
    g = 1'b1 ;
  end

  4'b0010 : begin // 2
    a = 1'b0 ;
    b = 1'b0 ;
    c = 1'b1 ;
    d = 1'b0 ;
    e = 1'b0 ;
    f = 1'b1 ;
    g = 1'b0 ;
  end

  4'b0011 : begin // 3
    a = 1'b0 ;
    b = 1'b0 ;
    c = 1'b0 ;
    d = 1'b0 ;
    e = 1'b1 ;
    f = 1'b1 ;
    g = 1'b1 ;
  end

  4'b0100 : begin // 4
    a = 1'b1 ;
    b = 1'b0 ;
    c = 1'b0 ;
    d = 1'b1 ;
    e = 1'b1 ;
    f = 1'b0 ;
```

```
  g = 1'b1 ;
end

4'b0101 : begin // 5
  a = 1'b0 ;
  b = 1'b1 ;
  c = 1'b0 ;
  d = 1'b0 ;
  e = 1'b1 ;
  f = 1'b0 ;
  g = 1'b1 ;
end

4'b0110 : begin // 6
  a = 1'b0 ;
  b = 1'b1 ;
  c = 1'b0 ;
  d = 1'b0 ;
  e = 1'b0 ;
  f = 1'b0 ;
  g = 1'b1 ;
end

4'b0111 : begin // 7
  a = 1'b0 ;
  b = 1'b0 ;
  c = 1'b0 ;
  d = 1'b1 ;
  e = 1'b1 ;
  f = 1'b1 ;
  g = 1'b1 ;
end

4'b1000 : begin // 8
  a = 1'b0 ;
  b = 1'b0 ;
  c = 1'b0 ;
  d = 1'b0 ;
  e = 1'b0 ;
  f = 1'b0 ;
  g = 1'b0 ;
end

4'b1000 : begin // 9
```

```
   a = 1'b0 ;
   b = 1'b0 ;
   c = 1'b0 ;
   d = 1'b0 ;
   e = 1'b1 ;
   f = 1'b0 ;
   g = 1'b0 ;
  end


          endcase
end

endmodule
```

---

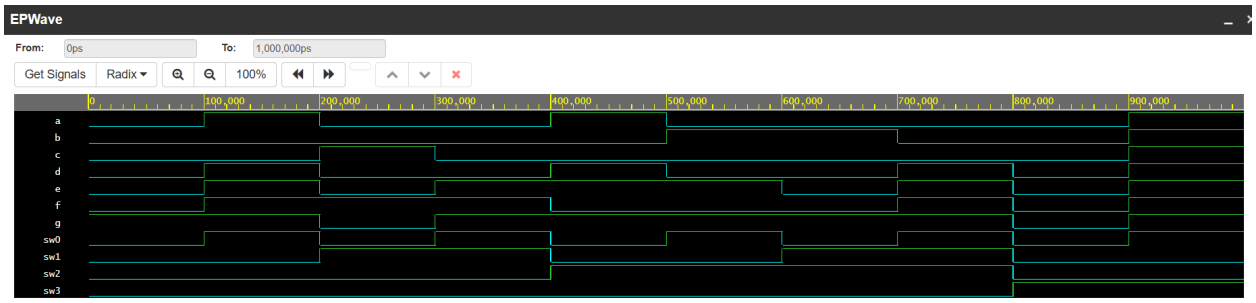
testbench.sv

```
// Code your testbench here
// or browse Examples
`timescale 1ns / 1ps
// http://www.electronics-tutorials.ws/combination/comb_6.html
module bcdtoled_tb;

  // Inputs
  reg sw0;
  reg sw1;
  reg sw2;
  reg sw3;

  // Outputs
  wire a;
  wire b;
  wire c;
  wire d;
  wire e;
  wire f;
  wire g;

  // Instantiate the Unit Under Test (UUT)
  // bcdto7led_st uut (
  bcd_to_7led_bh uut (
    .sw0(sw0),
```

```
  .sw1(sw1),
  .sw2(sw2),
  .sw3(sw3),
  .a(a),
  .b(b),
  .c(c),
  .d(d),
  .e(e),
  .f(f),
        .g(g)
);

initial begin

$dumpfile("dump.vcd"); $dumpvars;

  // Initialize Inputs
  sw3 = 0; sw2 = 0; sw1 = 0; sw0 = 0;
  #100;
  $display("TC10 ");
        if ( {a,b,c,d,e,f,g} != 7'b0000001 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

  sw3 = 0; sw2 = 0; sw1 = 0; sw0 = 1;
  #100;
  $display("TC11 ");
  if ( {a,b,c,d,e,f,g} != 7'b1001111 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

  sw3 = 0; sw2 = 0; sw1 = 1; sw0 = 0;
  #100;
  $display("TC12 ");
  if ( {a,b,c,d,e,f,g} != 7'b0010010 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

  sw3 = 0; sw2 = 0; sw1 = 1; sw0 = 1;
  #100;
  $display("TC13 ");
  if ( {a,b,c,d,e,f,g} != 7'b0000110 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

  sw3 = 0; sw2 = 1; sw1 = 0; sw0 = 0;
  #100;
  $display("TC14 ");
  if ( {a,b,c,d,e,f,g} != 7'b1001100 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

  sw3 = 0; sw2 = 1; sw1 = 0; sw0 = 1;
  #100;
```

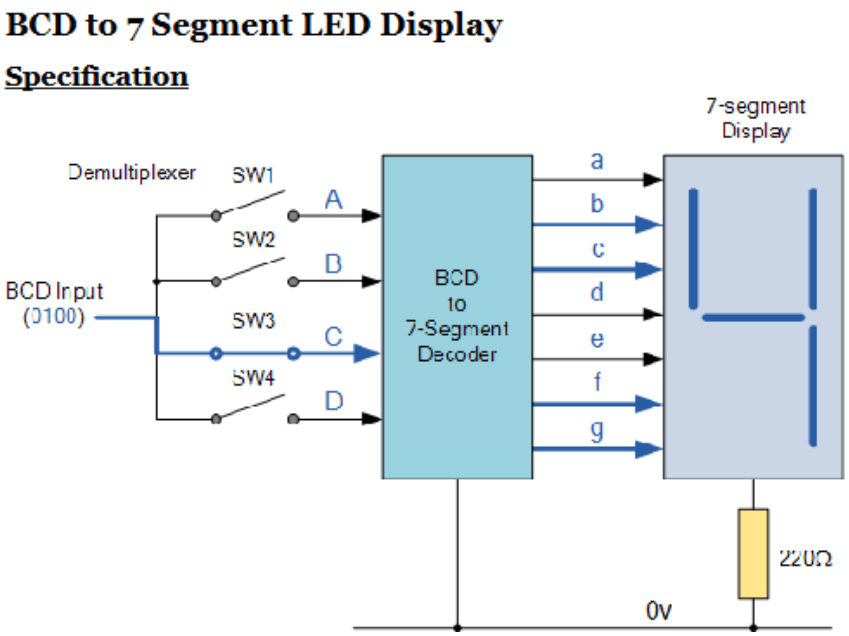
```
    $display("TC15 ");
    if ( {a,b,c,d,e,f,g} != 7'b0100100 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

    sw3 = 0; sw2 = 1; sw1 = 1; sw0 = 0;
    #100;
    $display("TC16 ");
    if ( {a,b,c,d,e,f,g} != 7'b0100000 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

    sw3 = 0; sw2 = 1; sw1 = 1; sw0 = 1;
    #100;
    $display("TC17 ");
    if ( {a,b,c,d,e,f,g} != 7'b0001111 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

    sw3 = 1; sw2 = 0; sw1 = 0; sw0 = 0;
    #100;
    $display("TC18 ");
    if ( {a,b,c,d,e,f,g} != 7'b0000000 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

    sw3 = 1; sw2 = 0; sw1 = 0; sw0 = 1;
    #100;
    $display("TC19 ");
    if ( {a,b,c,d,e,f,g} != 7'b0000100 ) $display ("Result is wrong %b", {a,b,c,d,e,f,g});

end
endmodule
```

Waveform |



[FIGURE 2] - screenshot of waveform generated by EDA Playground for code

Specification:



[FIGURE 3] - Decoder design that was provided in lab manual, using a 4 bit binary input,

Discussion

The provided specifications for our system was that we needed to implement a BCD-to-7 Segment decoder, given a binary decimal input split into four switches, which we labeled sw1, sw2, sw3, and sw4. Those switches were then used as inputs for a decoder, which gave a binary output to 7 additional wires, which were labeled letters a - g. Each of these wires corresponded to a segment on the given 7-segment display.

The test cases that we wrote were intended to make sure that the code would output the correct binary values that would make our hypothetical display output the correct number. They worked as intended, so our system did indeed work according to the expectations set by the spec diagram.

The only technical issue that we encountered was an error in our code that we weren't able to debug at first due to our own unfamiliarity with the language (and Verilog's very ambiguous error output messages), but this was quickly resolved after sending messaging our TA and receiving a pointer on how we could solve the issue, which worked. We were missing an "endmodule" statement at the end of our "design.sv" file, but the Verilog errors that we encountered referenced the location of our timescale directives and a syntax error on a line that didn't exist. The error also pointed us to the wrong file, telling us that these errors were occurring in our "testbench.sv" file as opposed to the other file, where the error actually was. However, this didn't cause us to make any system modifications or redesigns.

I don't think there are many ways that this system could be improved. The first place my brain went was to see if we could decrease the size of the input, but since there are 9 unique inputs, we need all four bits that the input uses. This obviously isn't included in the specifications document, but there might be some room to optimize if the output values were stored in memory somewhere, possibly using the segments that overlap from number to number (i.e. segments "b" and "c" being used in the outputs for both 1 and 9). I'm not too sure how that would be implemented, or if it would be necessary to do at all, but it might be possible.

Conclusion

The purpose of this lab was to understand how we could use switch inputs and assign them outputs that would provide a different result, essentially implementing a decoder with Verilog. Additionally, we learned how to write and use test cases that involved these new ideas, and were able to accurately identify whether or not our code did what we intended it to.

Questions

N/A - No questions were asked in the lab manual.