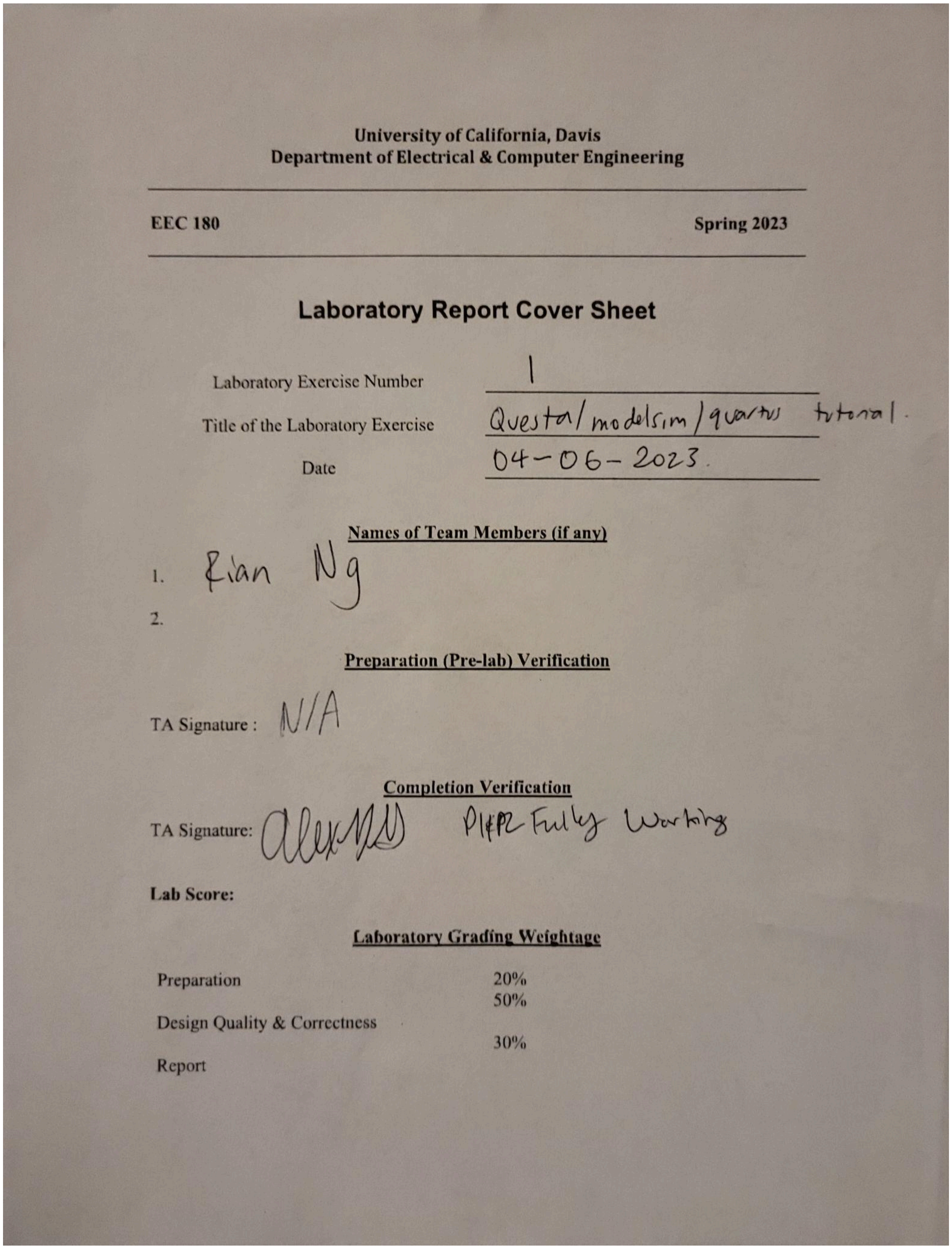


University of California, Davis
Department of Electrical & Computer Engineering

EEC 180 **Spring 2023**

Laboratory Report Cover Sheet

Laboratory Exercise Number	1
Title of the Laboratory Exercise	Questa/modelsim/quartus tutoral.
Date	04-06-2023.

Names of Team Members (if any)

1. Rian Ng
2.

Preparation (Pre-lab) Verification

TA Signature : N/A

Completion Verification

TA Signature: [illegible] P1&P2 Fully Working

Lab Score:

Laboratory Grading Weightage

Preparation	20%
Design Quality & Correctness	50%
Report	30%

Lab 1: Questa/Modelsim/Quartus Tutorial

Objective:

The objective of this lab is to complete a design flow for implementing a high-level Verilog design on the Intel DE10- Lite board. In this lab, we used the tools System Builder, Quartus Prime, and Questa / ModelSim. We practiced implementing simple logic gates on the Intel DE10-Lite board on its 7-Segment displays by inputting 4-bit inputs.

Prelab:

During the prelab, I installed the Quartus Prime Lite 22.1 Software and ModelSim.

Lab:

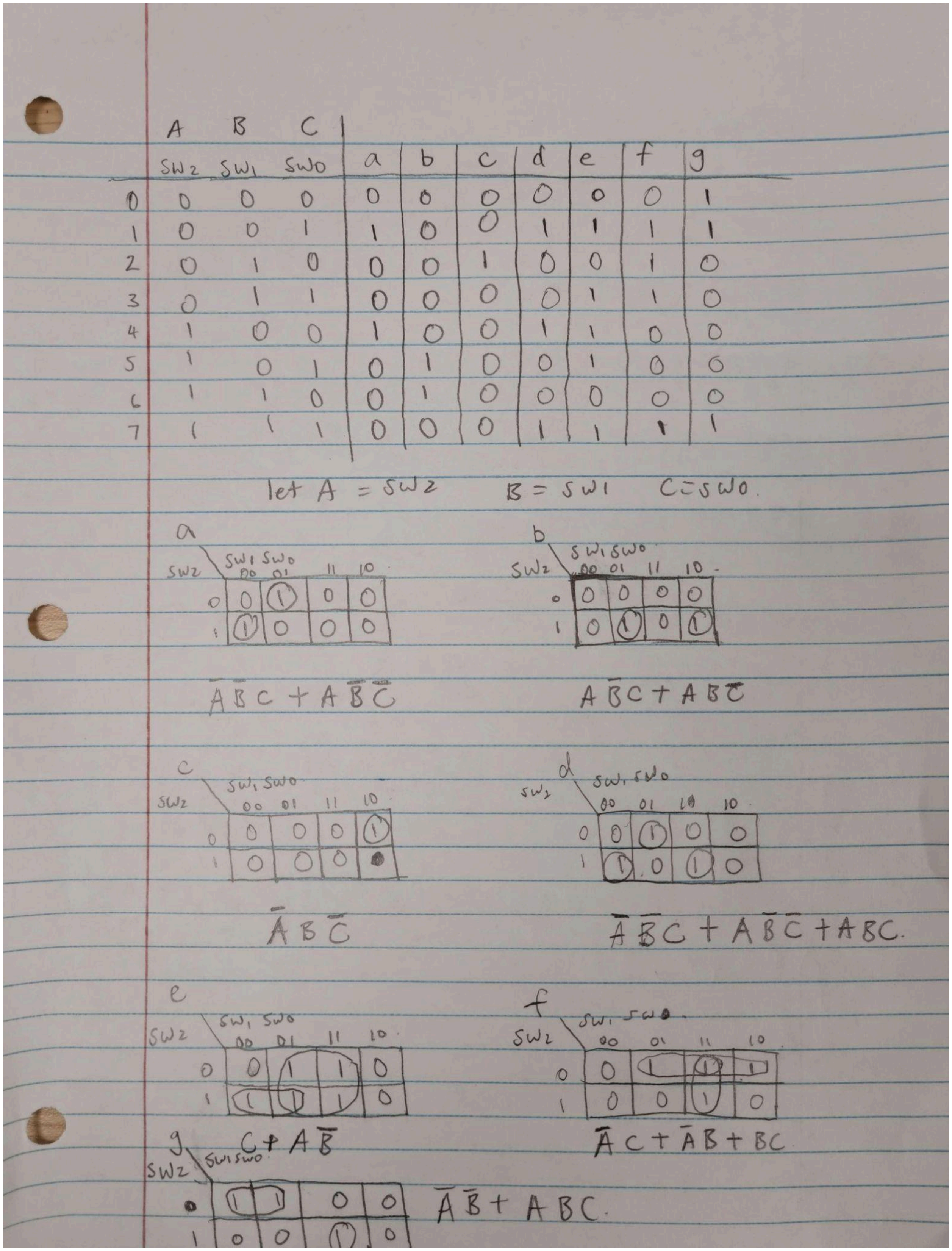
Part 1:

In Part 1 of the lab, I created a new project and implemented logic gates of my choice. I created a 3 bit binary to decimal converter on the 7 segment displays.

The table below shows what inputs and outputs are expected to be displayed on the 7-segment displays after putting in a binary input through the switches on the Intel DE10-Lite board.

Binary Value	Decimal Value (Displayed on 7-Segment Display)
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table 1: 3-Bit Binary to Decimal



	A	B	C							
	SW2	SW1	SW0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1	1	1	1
2	0	1	0	0	0	1	0	0	1	0
3	0	1	1	0	0	0	0	1	1	0
4	1	0	0	1	0	0	1	1	0	0
5	1	0	1	0	1	0	0	1	0	0
6	1	1	0	0	1	0	0	0	0	0
7	1	1	1	0	0	0	1	1	1	1

let A = SW2 B = SW1 C = SW0.

a

SW2 \ SW1 SW0	00	01	11	10
0	0	1	0	0
1	1	0	0	0

$\bar{A}\bar{B}C + A\bar{B}\bar{C}$

b

SW2 \ SW1 SW0	00	01	11	10
0	0	0	0	0
1	0	1	0	1

$A\bar{B}C + AB\bar{C}$

c

SW2 \ SW1 SW0	00	01	11	10
0	0	0	0	1
1	0	0	0	●

$\bar{A}B\bar{C}$

d

SW2 \ SW1 SW0	00	01	11	10
0	0	1	0	0
1	1	0	1	0

$\bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC.$

e

SW2 \ SW1 SW0	00	01	11	10
0	0	1	1	0
1	1	1	1	0

$C + A\bar{B}$

f

SW2 \ SW1 SW0	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$\bar{A}C + \bar{A}B + BC$

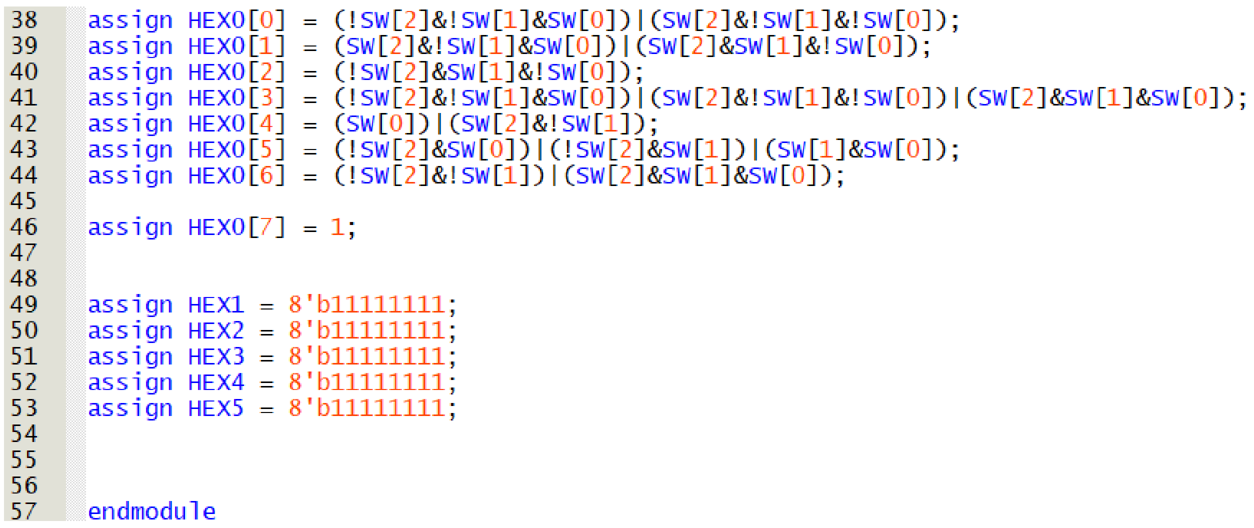
g

SW2 \ SW1 SW0				
0	1	1	0	0
1	0	0	1	0

$\bar{A}\bar{B} + ABC.$

Figure 1.1: Truth Table, K-Maps, Logic Gates for the 7-Segment Display

After constructing the K-Maps, I worked on the Verilog code to code the logic needed to display the output I wanted on the 7-segment displays.



```
assign HEX0[0] = (!SW[2]&!SW[1]&SW[0])|(SW[2]&!SW[1]&!SW[0]);
assign HEX0[1] = (SW[2]&!SW[1]&SW[0])|(SW[2]&SW[1]&!SW[0]);
assign HEX0[2] = (!SW[2]&SW[1]&!SW[0]);
assign HEX0[3] = (!SW[2]&!SW[1]&SW[0])|(SW[2]&!SW[1]&!SW[0])|(SW[2]&SW[1]&SW[0]);
assign HEX0[4] = (SW[0])|(SW[2]&!SW[1]);
assign HEX0[5] = (!SW[2]&SW[0])|(!SW[2]&SW[1])|(SW[1]&SW[0]);
assign HEX0[6] = (!SW[2]&!SW[1])|(SW[2]&SW[1]&SW[0]);

assign HEX0[7] = 1;


assign HEX1 = 8'b11111111;
assign HEX2 = 8'b11111111;
assign HEX3 = 8'b11111111;
assign HEX4 = 8'b11111111;
assign HEX5 = 8'b11111111;



endmodule
```

Figure 1.2: Verilog Code for 3-bit Binary to Decimal

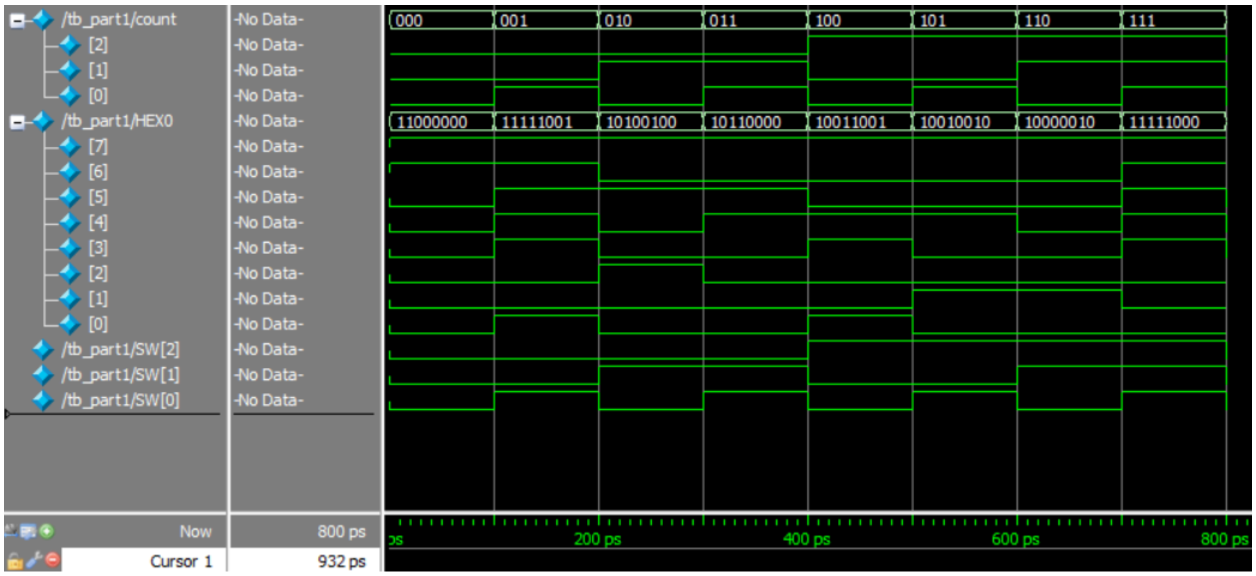
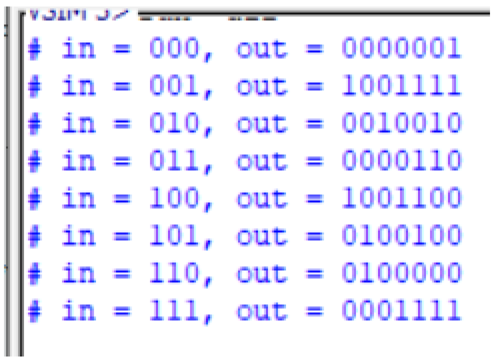


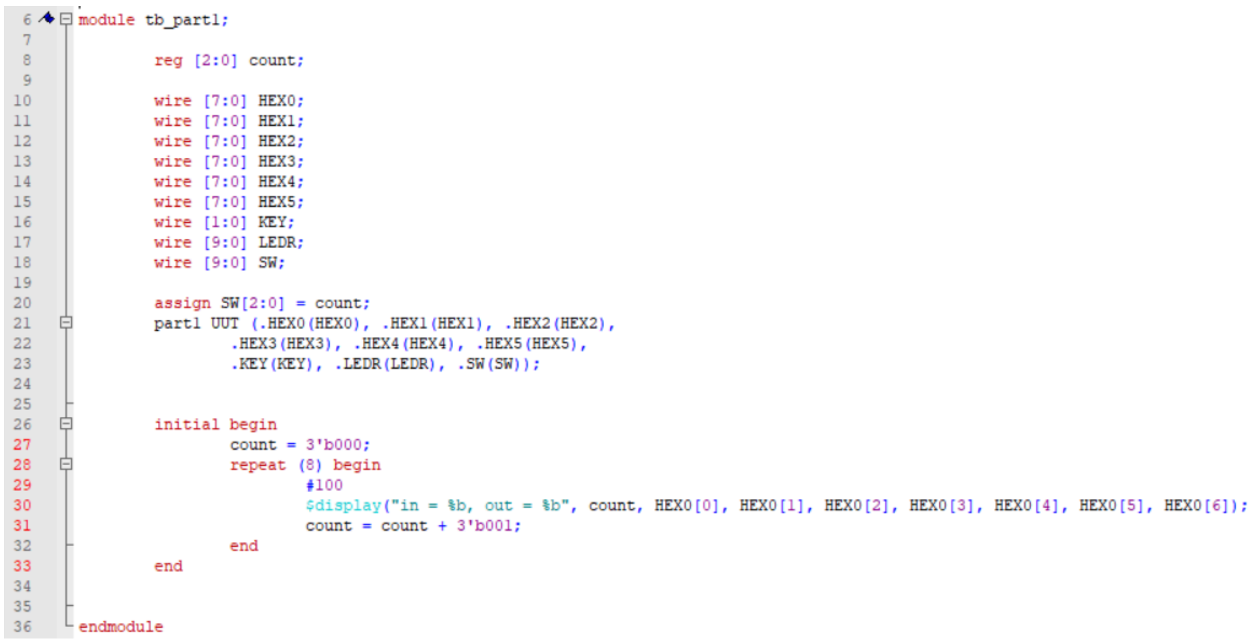
Figure 1.3: Waveform for 3-bit Binary to Decimal



```
# in = 000, out = 0000001
# in = 001, out = 1001111
# in = 010, out = 0010010
# in = 011, out = 0000110
# in = 100, out = 1001100
# in = 101, out = 0100100
# in = 110, out = 0100000
# in = 111, out = 0001111
```

Figure 1.4: Test Bench for 3-bit Binary to Decimal

Figure 1.4 matches the truth table shown in Figure 1.1, meaning that the programmed logic is correct. The Verilog code converts a 3 bit switch input to a decimal displayed on the 7 segment display.



```verilog
module tb_part1;

	reg [2:0] count;

	wire [7:0] HEX0;
	wire [7:0] HEX1;
	wire [7:0] HEX2;
	wire [7:0] HEX3;
	wire [7:0] HEX4;
	wire [7:0] HEX5;
	wire [1:0] KEY;
	wire [9:0] LEDR;
	wire [9:0] SW;

	assign SW[2:0] = count;
	part1 UUT (.HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2),
		.HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5),
		.KEY(KEY), .LEDR(LEDR), .SW(SW));


	initial begin
		count = 3'b000;
		repeat (8) begin
			#100
			$display("in = %b, out = %b", count, HEX0[0], HEX0[1], HEX0[2], HEX0[3], HEX0[4], HEX0[5], HEX0[6]);
			count = count + 3'b001;
		end
	end


endmodule
```

Figure 1.5: Test Bench Verilog Code for 3-bit Binary to Decimal

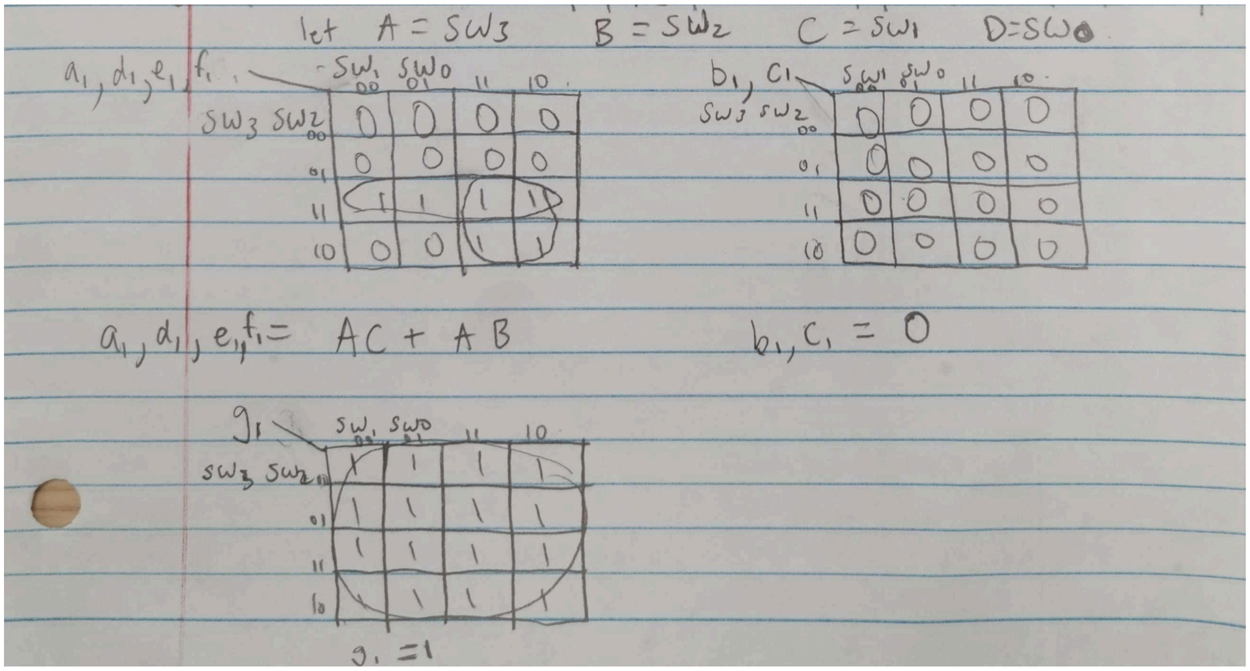
Part 2:

Reference:

i = index, A = SW[3], B = SW[2], C = SW[1], D = SW[0]

i	A	B	C	D	a1	b1	c1	d1	e1	f1	g1	a0	b0	c0	d0	e0	f0	g0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
5	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0
10	1	0	1	0	1	0	0	1	1	1	1	0	0	0	0	0	0	1
11	1	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1
12	1	1	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	1	0
14	1	1	1	0	1	0	0	1	1	1	1	1	0	0	1	1	0	0
15	1	1	1	1	1	0	0	1	1	1	1	0	1	0	0	1	0	0

Table 2: Truth Table for 4-Bit Switch Input to Decimal Display



let A = SW3 B = SW2 C = SW1 D = SW0

a_1, d_1, e_1, f_1

SW3 SW2 \ SW1 SW0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

b_1, c_1

SW3 SW2 \ SW1 SW0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

$a_1, d_1, e_1, f_1 = AC + AB$

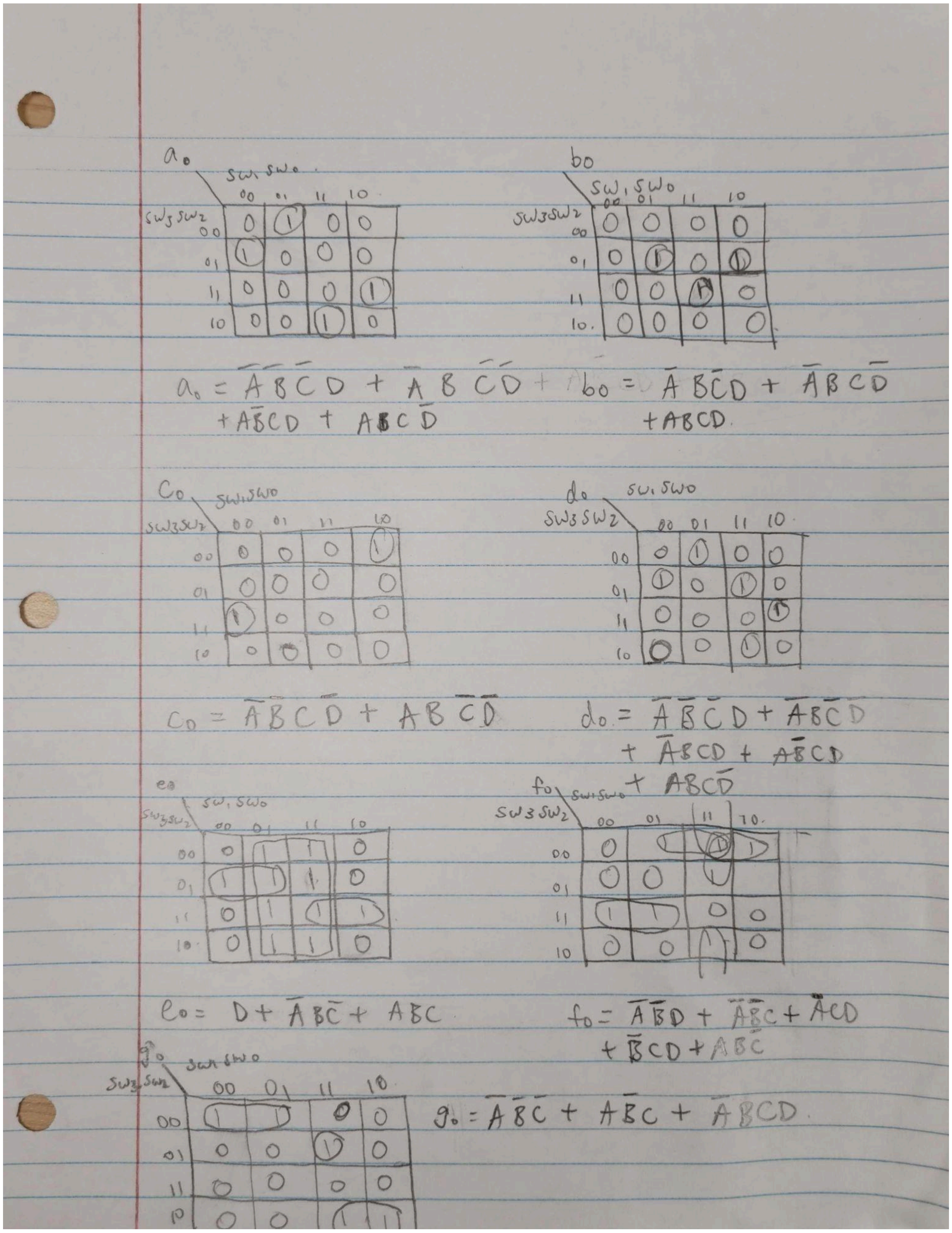
$b_1, c_1 = 0$

g_1

SW3 SW2 \ SW1 SW0	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$g_1 = 1$

Figure 2.1: Truth Table, K-Maps, Logic Gates for the 7-Segment Display



a_0

SW3SW2 \ SW1SW0	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	0	0	0	1
10	0	0	1	0

b_0

SW3SW2 \ SW1SW0	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	0	0	1	0
10.	0	0	0	0

$a_0 = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D} + A\bar{B}CD + ABC\bar{D}$

$b_0 = \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + ABCD.$

c_0

SW3SW2 \ SW1SW0	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	1	0	0	0
10	0	0	0	0

d_0

SW3SW2 \ SW1SW0	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	0	0	0	1
10	0	0	1	0

$c_0 = \bar{A}\bar{B}C\bar{D} + AB\bar{C}\bar{D}$

$d_0 = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD + A\bar{B}CD + ABC\bar{D}$

e_0

SW3SW2 \ SW1SW0	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	0	1	1	1
10	0	1	1	0

f_0

SW3SW2 \ SW1SW0	00	01	11	10.
00	0	1	1	1
01	0	0	1	
11	1	1	0	0
10	0	0	1	0

$e_0 = D + \bar{A}B\bar{C} + A\bar{B}C$

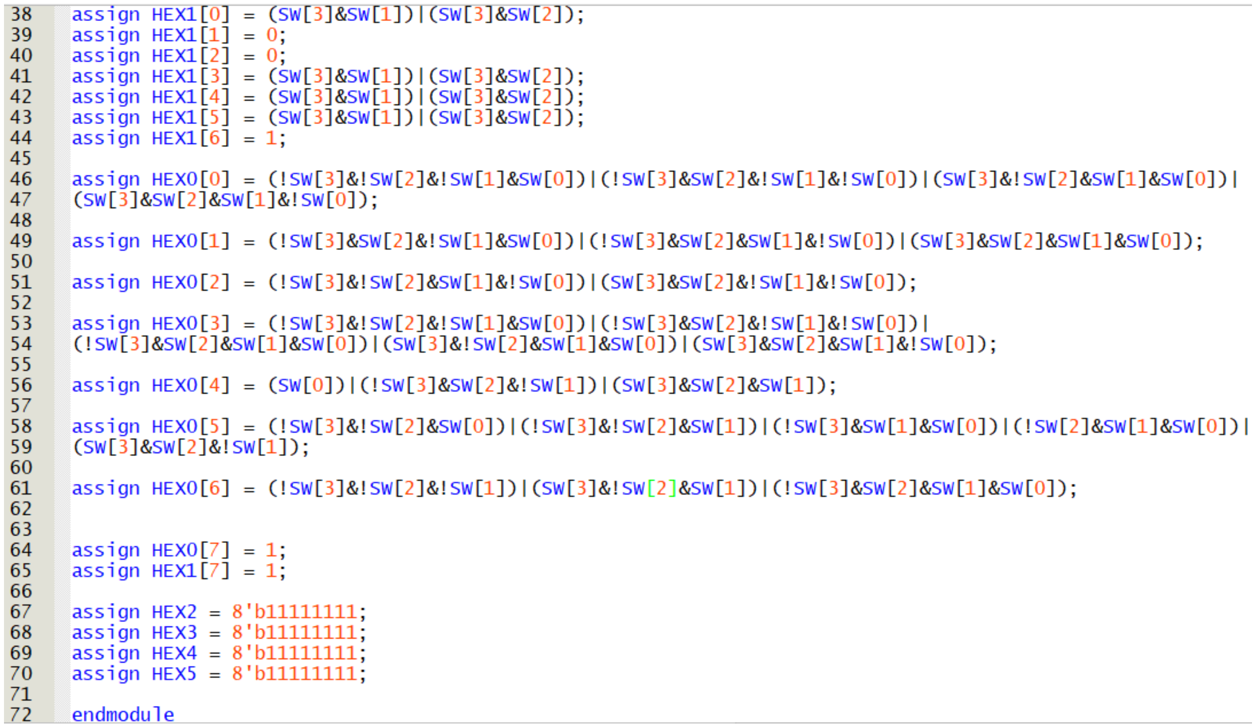
$f_0 = \bar{A}\bar{B}D + \bar{A}\bar{B}C + \bar{A}CD + \bar{B}CD + AB\bar{C}$

g_0

SW3SW2 \ SW1SW0	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	0	0	0	0
10	0	0	1	1

$g_0 = \bar{A}\bar{B}\bar{C} + A\bar{B}C + \bar{A}BCD.$

Figure 2.2: Truth Table, K-Maps, Logic Gates for the 7-Segment Display



```
assign HEX1[0] = (SW[3]&SW[1])|(SW[3]&SW[2]);
assign HEX1[1] = 0;
assign HEX1[2] = 0;
assign HEX1[3] = (SW[3]&SW[1])|(SW[3]&SW[2]);
assign HEX1[4] = (SW[3]&SW[1])|(SW[3]&SW[2]);
assign HEX1[5] = (SW[3]&SW[1])|(SW[3]&SW[2]);
assign HEX1[6] = 1;

assign HEX0[0] = (!SW[3]&!SW[2]&!SW[1]&SW[0])|(!SW[3]&SW[2]&!SW[1]&!SW[0])|(SW[3]&!SW[2]&SW[1]&SW[0])|
(SW[3]&SW[2]&SW[1]&!SW[0]);

assign HEX0[1] = (!SW[3]&SW[2]&!SW[1]&SW[0])|(!SW[3]&SW[2]&SW[1]&!SW[0])|(SW[3]&SW[2]&SW[1]&SW[0]);

assign HEX0[2] = (!SW[3]&!SW[2]&SW[1]&!SW[0])|(SW[3]&SW[2]&!SW[1]&!SW[0]);

assign HEX0[3] = (!SW[3]&!SW[2]&!SW[1]&SW[0])|(!SW[3]&SW[2]&!SW[1]&!SW[0])|
(!SW[3]&SW[2]&SW[1]&SW[0])|(SW[3]&!SW[2]&SW[1]&SW[0])|(SW[3]&SW[2]&SW[1]&!SW[0]);

assign HEX0[4] = (SW[0])|(!SW[3]&SW[2]&!SW[1])|(SW[3]&SW[2]&SW[1]);

assign HEX0[5] = (!SW[3]&!SW[2]&SW[0])|(!SW[3]&!SW[2]&SW[1])|(!SW[3]&SW[1]&SW[0])|(!SW[2]&SW[1]&SW[0])|
(SW[3]&SW[2]&!SW[1]);

assign HEX0[6] = (!SW[3]&!SW[2]&!SW[1])|(SW[3]&!SW[2]&SW[1])|(!SW[3]&SW[2]&SW[1]&SW[0]);


assign HEX0[7] = 1;
assign HEX1[7] = 1;

assign HEX2 = 8'b11111111;
assign HEX3 = 8'b11111111;
assign HEX4 = 8'b11111111;
assign HEX5 = 8'b11111111;

endmodule
```

Figure 2.3: Verilog Code for 4-bit Binary to Decimal

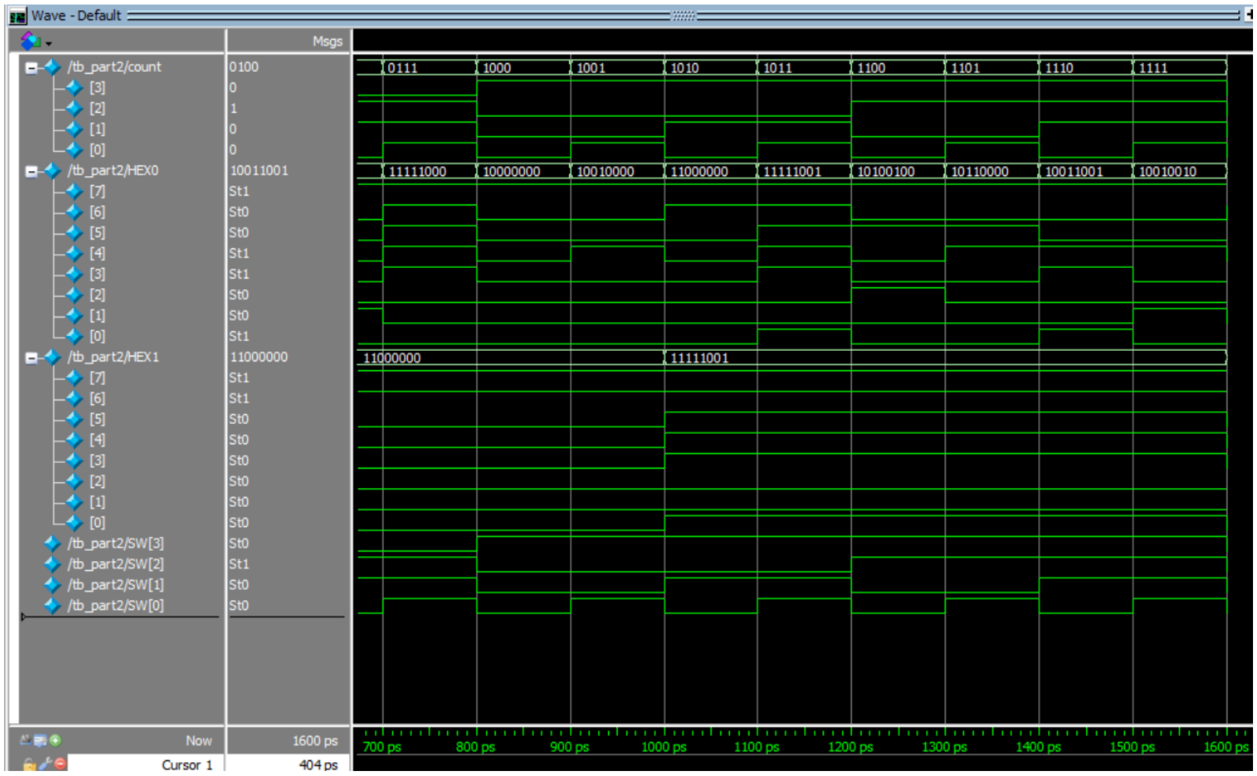
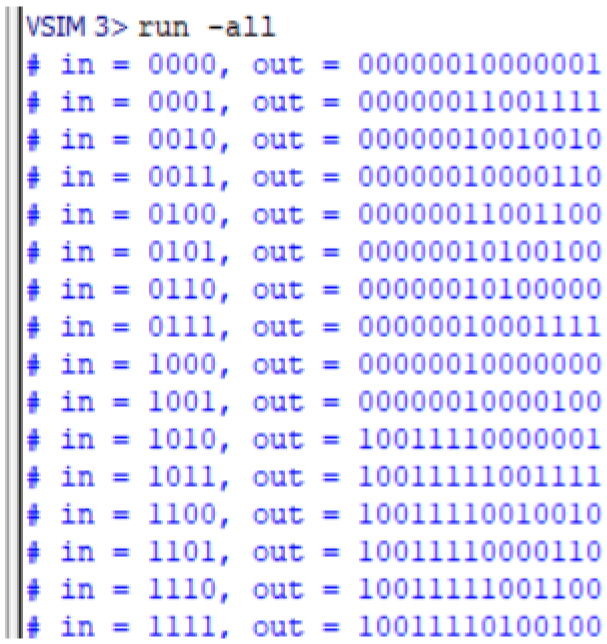


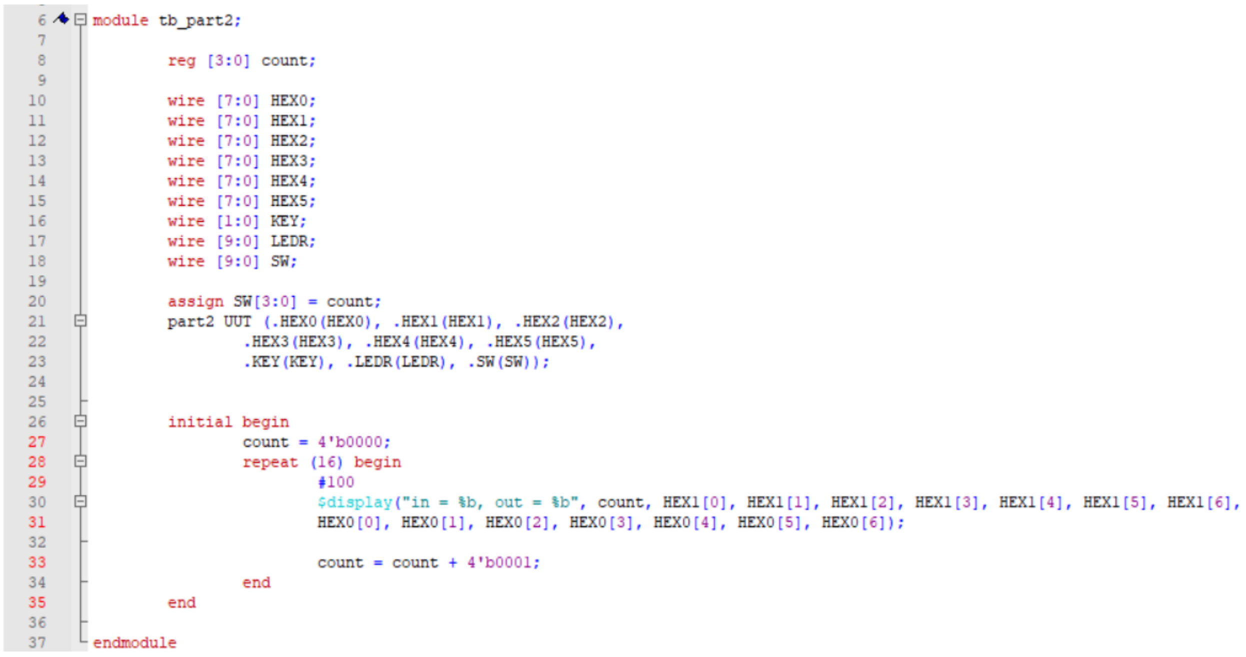
Figure 2.4: Waveform for 4-bit Binary to Decimal



```
VSIM 3> run -all
# in = 0000, out = 00000010000001
# in = 0001, out = 00000011001111
# in = 0010, out = 00000010010010
# in = 0011, out = 00000010000110
# in = 0100, out = 00000011001100
# in = 0101, out = 00000010100100
# in = 0110, out = 00000010100000
# in = 0111, out = 00000010001111
# in = 1000, out = 00000010000000
# in = 1001, out = 00000010000100
# in = 1010, out = 10011110000001
# in = 1011, out = 10011111001111
# in = 1100, out = 10011110010010
# in = 1101, out = 10011110000110
# in = 1110, out = 10011111001100
# in = 1111, out = 10011110100100
```

Figure 2.5: Test Bench Result for 4-bit Binary to Decimal

With the test bench result providing the truth table seen in Table 2, this means that the Verilog code was programmed correctly to convert a 4-bit switch input to a decimal value displayed on the 7 segment displays.



```
module tb_part2;

        reg [3:0] count;

        wire [7:0] HEX0;
        wire [7:0] HEX1;
        wire [7:0] HEX2;
        wire [7:0] HEX3;
        wire [7:0] HEX4;
        wire [7:0] HEX5;
        wire [1:0] KEY;
        wire [9:0] LEDR;
        wire [9:0] SW;

        assign SW[3:0] = count;
        part2 UUT (.HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2),
                .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5),
                .KEY(KEY), .LEDR(LEDR), .SW(SW));


        initial begin
                count = 4'b0000;
                repeat (16) begin
                        #100
                        $display("in = %b, out = %b", count, HEX1[0], HEX1[1], HEX1[2], HEX1[3], HEX1[4], HEX1[5], HEX1[6],
                        HEX0[0], HEX0[1], HEX0[2], HEX0[3], HEX0[4], HEX0[5], HEX0[6]);

                        count = count + 4'b0001;
                end
        end

endmodule
```

Figure 2.6: Test Bench Verilog Code for 4-bit Binary to Decimal

<u>Results:</u>

Part 1:

After creating a truth table, K-Maps, and programming the logic for the 3-bit switch input to 7-segment decimal output, the result produced exactly what I was expecting. Flipping through the binary values of 000 to 111 yielded me the decimal values from 0 to 7. Running the test bench and waveforms produced the same results as the truth table.

Part 2:

After constructing the Verilog code for Part 1, this gave me a better understanding of how to program the 4-bit switch input to 7-segment decimal output. I was able to produce the 4-bit switch input to 7-segment decimal output after creating several K-Maps to get the logic needed for each segment. Flipping through the binary values from 0000 to 1111 produced the decimal outputs 0 to 15. The waveform and test bench results produced exactly what I was expecting as it was exactly the same as the truth table that I constructed.

<u>Conclusion:</u>

In conclusion, this lab went really well. I was able to get a better understanding of how to program in Verilog. I got a better understanding of the Verilog syntax as well as how to produce results with test benches and waveforms. I had some challenges during this lab as I messed up some of my logic inputs and had to figure out how to debug each segment on the display to produce the output I wanted. Overall, I feel that this lab was successful as I was able to produce the results that I expected in both Part I and Part II of the lab.