ECE 210 LAB-3 REPORT

Name: Misbah Ahmed Nauman
Student ID: 1830574
CCID: misbahah

Name: Muhammad Abdur Rab Siddiqui
Student ID: 1804733
CCID: msiddiq6

Section Number: D25
Lab Number: 3
Lab Date: 11.05.2024

PRE-LAB ASSIGNMENT

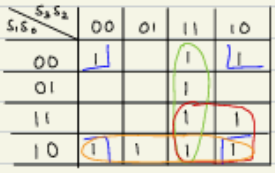
1. Given the truth table below, Use K-Maps, determine the minimum logical expression for each of **the 7 outputs (A through G)**:

Switches Input				Selected 7-Segment Binary Inputs								Display
					out_7seg(6)	out_7seg(5)	out_7seg(4)	out_7seg(3)	out_7seg(2)	out_7seg(1)	out_7seg(0)	
SW3	SW2	SW1	SW0	CC	A	B	C	D	E	F	G	
0	0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	0	0	0	0	1	1	0	1
0	0	1	0	0	1	1	0	1	1	0	1	2
0	0	1	1	0	1	0	0	1	1	1	1	3
0	1	0	0	0	0	0	1	0	1	1	1	4
0	1	0	1	0	1	0	1	1	0	1	1	5
0	1	1	0	0	1	1	1	1	0	1	1	6
0	1	1	1	0	0	0	0	1	1	1	0	7
1	0	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	0	1	0	1	1	1	1	1	9
1	0	1	0	0	0	1	1	1	1	1	1	A
1	0	1	1	0	1	1	1	0	0	1	1	b
1	1	0	0	0	1	1	1	1	0	0	0	C
1	1	0	1	0	1	1	0	0	1	1	1	d
1	1	1	0	0	1	1	1	1	0	0	1	E
1	1	1	1	0	0	1	1	1	0	0	1	F

For A:

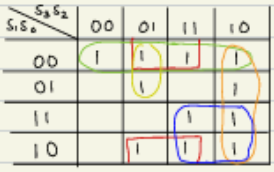
$A = S_1'S_3 + S_1'S_0S_3 + S_0'S_1'S_2' + S_1S_0'S_2 + S_1S_0S_2'$

For B:



$B = S_0'S_1' + S_3S_1 + S_1S_0' + S_1S_3$

For C:

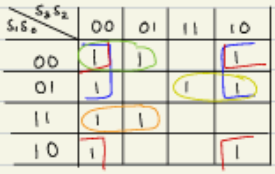


$C = S_1'S_0' + S_0'S_2 + S_3S_2' + S_1S_3 + S_1'S_3'S_2$

For D:

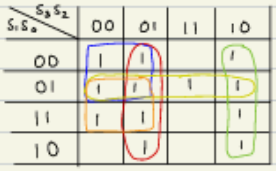
$D = S_0'S_2' + S_0'S_3 + S_1S_3' + S_1S_2 + S_1'S_3S_2' + S_0S_3'S_2$

For E:



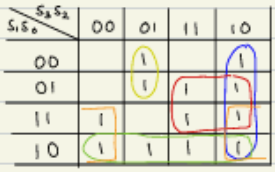
$E = S_1'S_2' + S_0'S_2' + S_1'S_0'S_3' + S_1'S_0S_3 + S_1S_0S_3'$

For F:

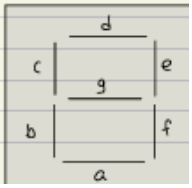


$F = S_1'S_3' + S_2'S_3 + S_3S_2' + S_1'S_0 + S_0S_3'$

For G:



$G1 = S_3S_2' + S_0S_3 + S_1S_0' + S_1S_2' + S_1'S_3'S_2$



$A = S_1'S_3 + S_0'S_3'S_2' + S_1S_0'S_2 + S_1S_0S_2'$

$B = S_0'S_2' + S_3S_2 + S_1S_0' + S_1S_3$

$C = S_1'S_0' + S_0'S_2 + S_3S_2' + S_1S_3 + S_1'S_3'S_2$

$D = S_0'S_2' + S_0'S_3 + S_1S_3' + S_1S_2 + S_1'S_3S_2' + S_0S_3'S_2$

$E = S_1'S_2' + S_0'S_2' + S_1'S_0'S_3' + S_1'S_0S_3 + S_1S_0S_3'$

$F = S_1'S_3' + S_3'S_2 + S_3S_2' + S_1'S_0 + S_0S_3'$

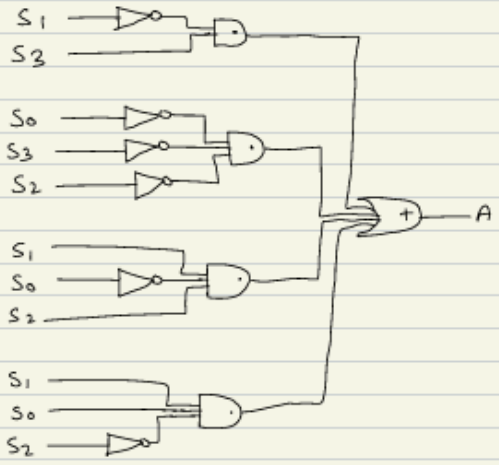
$G = S_3S_2' + S_0S_3 + S_1S_0' + S_1S_2' + S_1'S_3'S_2$

⇨ Ans

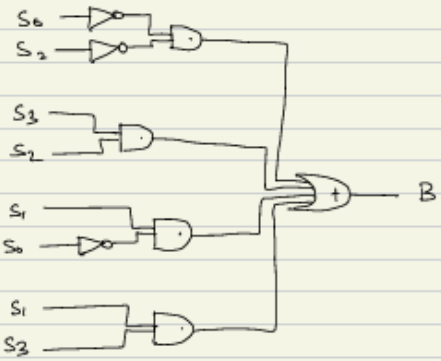
2. Draw circuit schematic diagram using minimized logical expressions, either use all AND-OR-NOT logic or all NAND logic gates.

using this so its easier to implement in VHDL

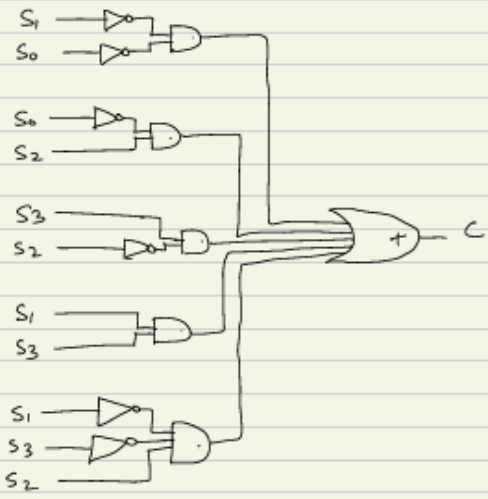
For A:



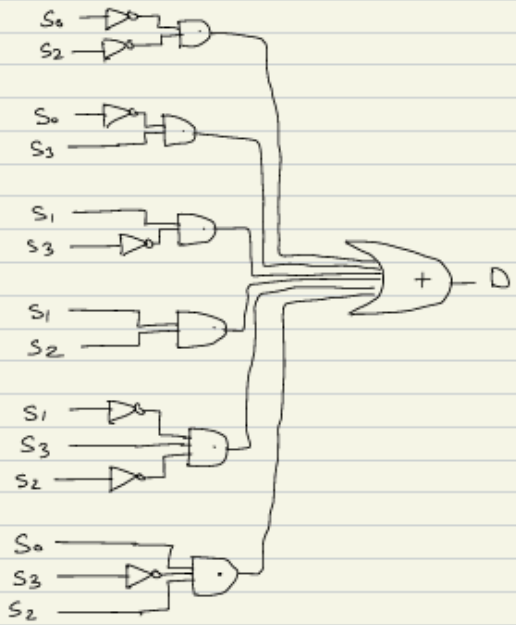
For B:



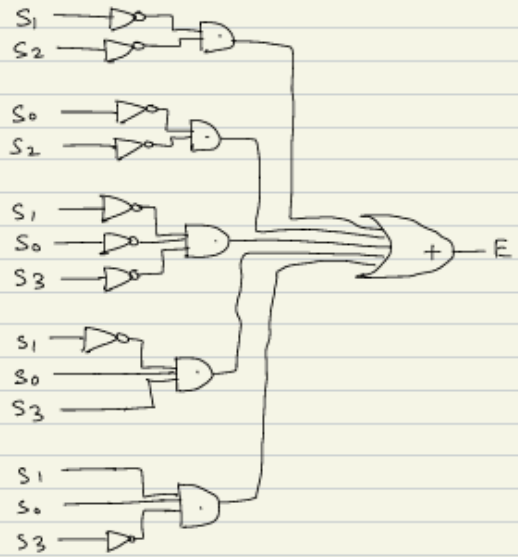
For C:



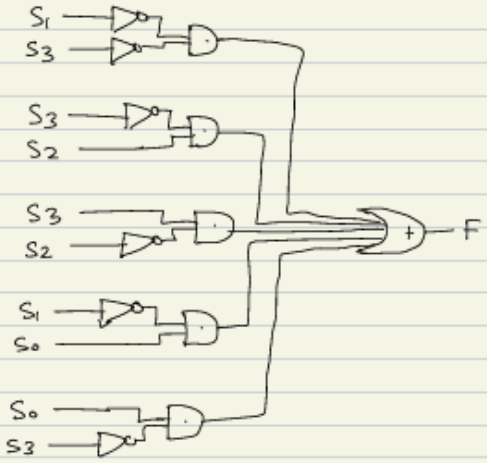
For D:



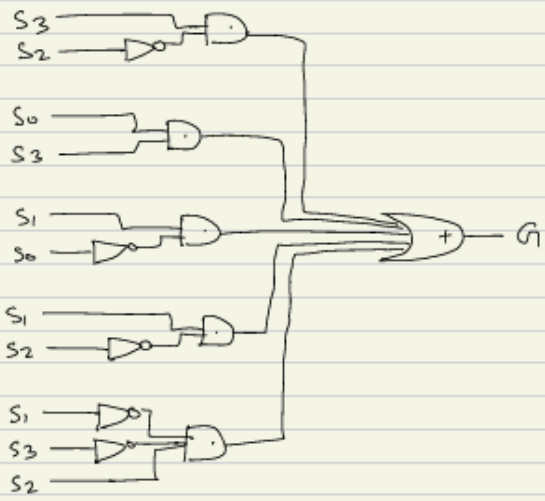
For E:



For F:



For G1:



3. Write and include VHDL solutions in your pre-lab assignment (using VHDL logical operators, such as AND, OR, etc).

Assuming m_input is a 4-bit vector representing the values of the switches.

m_inputs(0) ⟶ switch 3
m_inputs(1) ⟶ switch 2
m_inputs(2) ⟶ switch 1
m_inputs(3) ⟶ switch 0

```
architecture Behavioral of majority is
begin
    output_A <= (not m_input(1) and m_input(3)) or
                (not m_input(0) and not m_input(3) and not m_input(2)) or
                (m_input(1) and not m_input(0) and m_input(2)) or
                (m_input(1) and m_input(0) and not m_input(2));

    output_B <= (not m_input(0) and not m_input(2)) or
                (m_input(3) and m_input(2)) or
                (m_input(1) and not m_input(0)) or
                (m_input(1) and m_input(3));

    output_C <= (not m_input(1) and not m_input(0)) or
                (not m_input(0) and m_input(2)) or
                (m_input(3) and not m_input(2)) or
                (m_input(1) and m_input(3)) or
                (not m_input(1) and not m_input(3) and m_input(2));

    output_D <= (not m_input(0) and not m_input(2)) or
                (not m_input(0) and m_input(3)) or
                (m_input(1) and not m_input(3)) or
                (m_input(1) and m_input(2)) or
                (not m_input(1) and m_input(3) and not m_input(2)) or
                (m_input(0) and not m_input(3) and m_input(2));

    output_E <= (not m_input(1) and not m_input(2)) or
                (not m_input(0) and not m_input(2)) or
                (not m_input(1) and not m_input(0) and not m_input(3)) or
                (not m_input(1) and m_input(0) and m_input(3)) or
                (m_input(1) and m_input(0) and not m_input(3));

    output_F <= (not m_input(1) and not m_input(3)) or
                (not m_input(3) and  m_input(2)) or
                (m_input(3) and not m_input(2)) or
                (not m_input(1) and m_input(0)) or
                (m_input(0) and not m_input(3));

    output_G <= (m_input(3) and not m_input(2)) or
                (m_input(0) and m_input(3)) or
                (m_input(1) and not m_input(0)) or
                (m_input(1) and not m_input(2)) or
                (not m_input(1) and not m_input(3) and m_input(2));

end Behavioral;
```

The screenshots included above provide a comprehensive visual representation of the pre-lab solutions, which were essential for successfully completing the lab. These solutions include the derivation of minimized logical expressions for each of the seven outputs (A through G) of the 7-segment display. The minimized equations were obtained through Karnaugh Maps (K-Maps) and were tailored to the inverted orientation of the 7-segment display, as specified in the lab manual. This approach ensured the output was accurate when displayed on the Zybo Z7 board, considering the unique lab setup.

Additionally, circuit schematic diagrams were created using the minimized logical expressions. These schematics employed AND-OR-NOT. The diagrams served as a crucial

reference for both the in-lab implementation and the VHDL coding process.

The VHDL implementation for the 7-segment decoder was also part of the pre-lab work. Using logical operators such as AND, OR, and NOT, the VHDL solutions were written and verified for correctness. These solutions were cross-checked against the truth table and simulation results to ensure they met the expected functionality. Overall, the pre-lab solutions provided a solid foundation for the subsequent lab tasks.

ABSTRACT

This lab involved designing and implementing a 7-segment LED decoder and a binary up counter on the Zybo Z7 FPGA board. The design translated binary inputs (SW0-SW3) into hexadecimal values displayed on a 7-segment LED. A counter was then implemented to increment values every second up to the last two digits of the student ID. Verification was achieved through simulations and demonstrations of the design.

INTRODUCTION

This experiment aimed to familiarize students with digital circuit design and FPGA implementation using VHDL. The task was divided into three main parts:

1. Designing a 7-segment decoder for hexadecimal display using minimized logic expressions.
2. Implementing the design with VHDL, simulating, and testing on hardware.
3. Developing an up-counter using VHDL that displayed values from 0 to the last two digits of the student ID.

The lab demonstrated practical applications of digital logic, including combinational circuits and synchronous counters, using modern tools and techniques.

DESIGN SECTION

Part 1: 7-Segment Decoder Design

```
34      --Part I: Uncomment one of the code sections below for part I==============
35
36      out_7seg(0) <= (sw(3) and not sw(2)) or (sw(0) and sw(3)) or (sw(1) and not sw(0)) or (sw(1) and not sw(2)) or (not sw(1) and not sw(3) and sw(2)); --G
37      out_7seg(1) <= (not sw(1) and not sw(3)) or (not sw(3) and  sw(2)) or (sw(3) and not sw(2)) or (not sw(1) and sw(0)) or (sw(0) and not sw(3)); --F
38      out_7seg(2) <= (not sw(1) and not sw(2)) or (not sw(0) and not sw(2)) or (not sw(1) and not sw(0) and not sw(3)) or (not sw(1) and sw(0) and sw(3)) or (sw(1) and sw(0) and not sw(3)); --E
39      out_7seg(3) <= (not sw(0) and not sw(2)) or (not sw(0) and sw(3)) or (sw(1) and not sw(3)) or (sw(1) and sw(2)) or (not sw(1) and sw(3) and not sw(2)) or (sw(0) and not sw(3) and sw(2)); --D
40      out_7seg(4) <= (not sw(1) and not sw(0)) or (not sw(0) and sw(2)) or (sw(3) and not sw(2)) or (sw(1) and sw(3)) or (not sw(1) and not sw(3) and sw(2)); --C
41      out_7seg(5) <= (not sw(0) and not sw(2)) or (sw(3) and sw(2)) or (sw(1) and not sw(0)) or (sw(1) and sw(3));--B
42      out_7seg(6) <= (not sw(1) and sw(3)) or (not sw(0) and not sw(3) and not sw(2)) or (sw(1) and not sw(0) and sw(2)) or (sw(1) and sw(0) and not sw(2));--A
43      CC <= '1';
44      --End of part II==========================================================
```

Fig. 3.1: VHDL Code for Part 1.

1. **Minimized Logical Expressions**
   Using Karnaugh maps (K-Maps), logical expressions for each segment (A through

G) were derived from the truth table provided. These minimized expressions were used to implement the decoder circuit.

2. **Circuit Schematic**
   A schematic for the 7-segment decoder was drawn using AND-OR-NOT logic gates (as seen in the pre-lab solution). This schematic provided a clear representation of the minimized logical expressions.
3. **VHDL Implementation**
   The decoder logic was implemented in VHDL using logical operators (AND, OR, NOT). A simulation test bench verified the functionality for inputs 0 to F.

Part 2: VHDL Implementation of the Decoder

```
46      --Part II: Uncomment one of the code sections below for part II===============
47
48      process(sw)
49      begin
50          case sw is
51              -- Output values are based on the truth table from the Lab Manual
52              when "0000" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 0
53              when "0001" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '0'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 1
54              when "0010" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- 2
55              when "0011" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 3
56              when "0100" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '0'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 4
57              when "0101" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 5
58              when "0110" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 6
59              when "0111" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 7
60              when "1000" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 8
61              when "1001" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 9
62              when "1010" => out_7seg(6) <= '0'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- A
63              when "1011" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '0'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- B
64              when "1100" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '0'; -- C
65              when "1101" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '0'; out_7seg(3) <= '0'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- D
66              when "1110" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- E
67              when "1111" => out_7seg(6) <= '0'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- F
68              when others => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '0'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '0'; -- Default off
69              CC <= '0';
70          end case;
71      end process;
72
73
74      --End of part II==========================================================
```

Fig. 3.2: VHDL Code for Part 2.

1. **Conditional Statements**
   The decoder design was re-implemented using VHDL `if-else` or `case` statements to simplify readability and maintenance.
2. **Simulation and Verification**
   - The simulation outputs were verified against expected results for each hexadecimal input (0-F).
   - Results were plotted using waveform software to demonstrate the correctness of the design.

Part 3: Binary Counter Design

```
BEGIN

    CLK_Gen : PROCESS (clk) --Process will generate Clocks used for up counting 1Hz and 7Segmant CC 128Hz
        --SevenSeg_Count every (clk_divider_forOneHz) should increase. If SevenSeg_Count is equal to max_SevenSeg_Count it should go back to 0
    BEGIN
        IF rising_edge(clk) THEN

            IF (count < clk_divider_forOneHz) THEN --dividing internal clk by 2*clk_divider_forOneHz
                count <= count + '1';
            ELSE
                count <= (OTHERS => '0');                   --1 second has passed
                IF (SevenSeg_Count < max_SevenSeg_Count) THEN --up counting from 0 to last 2-digits of your student ID
                    SevenSeg_Count <= SevenSeg_Count + 1; ------------------------------------------------------ Write code to count up SevenSeg_Count
                ELSE
                    SevenSeg_Count <= 0 ;---------------------------------------------------------------------- Reset SevenSeg_count
                END IF;
            END IF;

            IF (count_7seg < clk_divider_for7seg) THEN --dividing internal clk by 2*clk_divider_for7seg
                count_7seg <= count_7seg + '1';
            ELSE
                count_7seg <= (OTHERS => '0'); --5ms has passed
                clk_out_7seg <= NOT clk_out_7seg ;----------------------------------------------------------------write your code here
                -- toggle 'clk_out_7seg' which will be assigned to 'CC' later in code
            END IF;

        END IF;
    END PROCESS;

    decimal_to_binary : PROCESS (SevenSeg_Count) --SevenSeg_show is 8 bits, write a code to show 4 MSBs on the left 7segment and 4 LSBs on right 7segment.
        --SevenSeg_Count is an integer that needs to be converted to a vector format
    BEGIN
        SevenSeg_show(7 DOWNTO 4) <= STD_LOGIC_VECTOR(to_unsigned(SevenSeg_Count / 10, 4));   --first digit of the last 2-digits of student ID is assigned to SevenSeg_show(7 downto 4)
        SevenSeg_show(3 DOWNTO 0) <= STD_LOGIC_VECTOR(to_unsigned(SevenSeg_Count REM 10, 4)); --second digit of the last 2-digits of student ID is assigned to SevenSeg_show(7 downto 4)
    END PROCESS;

    Decoder_8bitsto2SevenSegments : PROCESS (clk_out_7seg, SevenSeg_show)
        --SevenSeg_show is 8 bits, write a code to show 4 MSBs on the left 7segment and 4 LSBs on right 7segment.
        --Hint: While clk_out_7seg is '1' you are displaying 4 digits on one segment and while it is '0' you are displaying on the other one.
    BEGIN --Here students can call a component from part 2 of the lab or reuse their code



        IF (clk_out_7seg = '0') THEN
            -- Write code to display the left digit
            case SevenSeg_show(7 DOWNTO 4) is
            -- Output values are based on the truth table from the Lab Manual
                when "0000" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 0
                when "0001" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '0'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 1
                when "0010" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- 2
                when "0011" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 3
                when "0100" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '0'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 4
                when "0101" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 5
                when "0110" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 6
                when "0111" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 7
                when "1000" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 8
                when "1001" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 9
                when "1010" => out_7seg(6) <= '0'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- A
                when "1011" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '0'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- B
                when "1100" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- C
                when "1101" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- D
                when "1110" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- E
                when "1111" => out_7seg(6) <= '0'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- F
                when others => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '0'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '0'; -- Default off
            end case;
        ELSE
            -- Write code to display the right digit
            case SevenSeg_show(3 DOWNTO 0) is
            -- Output values are based on the truth table from the Lab Manual
                when "0000" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 0
                when "0001" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '0'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 1
                when "0010" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- 2
                when "0011" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 3
                when "0100" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '0'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 4
                when "0101" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 5
                when "0110" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 6
                when "0111" => out_7seg(6) <= '0'; out_7seg(5) <= '0'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '0'; -- 7
                when "1000" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 8
                when "1001" => out_7seg(6) <= '1'; out_7seg(5) <= '0'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- 9
                when "1010" => out_7seg(6) <= '0'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '1'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- A
                when "1011" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '0'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- B
                when "1100" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '0'; -- C
                when "1101" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '0'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '1'; out_7seg(0) <= '1'; -- D
                when "1110" => out_7seg(6) <= '1'; out_7seg(5) <= '1'; out_7seg(4) <= '1'; out_7seg(3) <= '1'; out_7seg(2) <= '0'; out_7seg(1) <= '0'; out_7seg(0) <= '1'; -- E
```

Fig. 3.3: VHDL Code for Part 3.

Code Explanation

The code controls a 7-segment display to show specific characters or digits based on a 4-bit input (SevenSeg_show). It alternates between two digits, the left and right,

depending on the clock signal (`clk_out_7seg`).

When the clock is low, the left digit is displayed by mapping the 4-bit input to specific segment outputs (`out_7seg(6 downto 0)`) using a `case` statement. Similarly, when the clock is high, the right digit is displayed following the same logic. Each 4-bit input corresponds to a character (0-9, A-F), as defined in a truth table.

A default case is included to turn off all segments when no valid input is provided. This design enables efficient control of two digits using a single set of signals.

1. **Counter Logic**
   An up-counter was designed to increment values from 0 to the last two digits of the student ID (e.g., 78). For demonstration, the count toggled between two 7-segment displays every second using the CC signal.
2. **VHDL Implementation**
   The design utilized a clock divider to slow down the FPGA clock to 1 Hz and a process block to implement the counter logic.

EXPERIMENTAL PROCEDURE AND EQUIPMENTS

EQUIPMENTS

- Xilinx Zybo Z7 FPGA development board
- Vivado software

PROCEDURE

1. We set up the circuit on the Zybo Z7 FPGA development board and ensured all connections were accurate according to the provided schematic.
2. Using the Vivado software, we wrote the VHDL code to implement the required functionality for the 7-segment display.
3. The code was compiled, synthesized, and uploaded to the FPGA board.
4. Next, we tested the circuit by providing various 4-bit inputs to verify that the correct characters or digits were displayed on the 7-segment display.
5. The clock signal was used to alternate between displaying the left and right digits, ensuring proper synchronization.
6. Any discrepancies in output were debugged by reviewing the truth table logic and verifying the VHDL code against the requirements.
7. Finally, the setup was observed and documented for analysis.

RESULTS

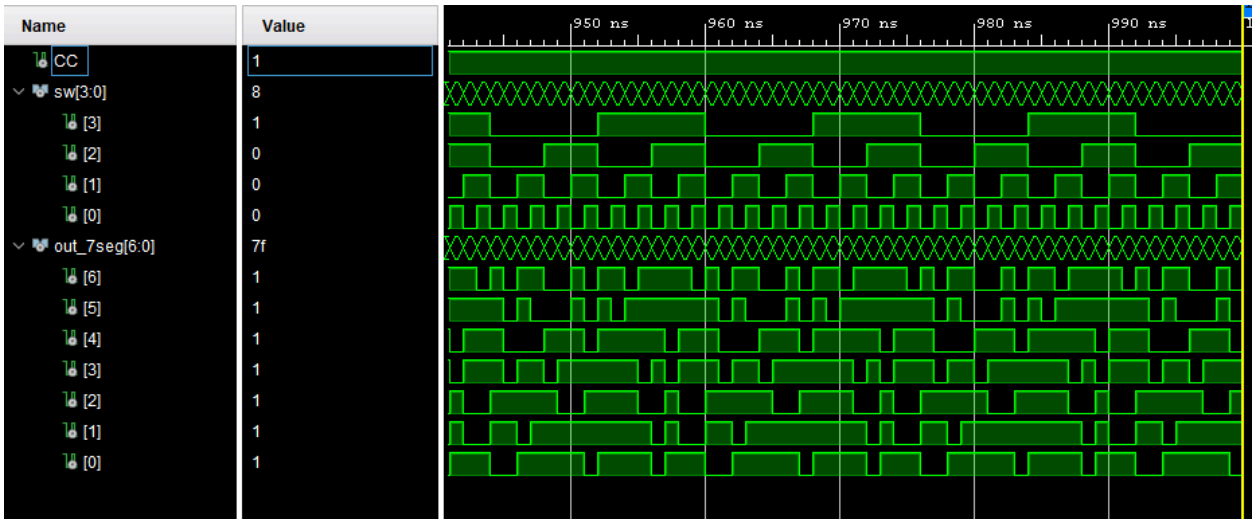


Fig. 3.4: Photo of a logic window taken in the Waveform software for Part 1.

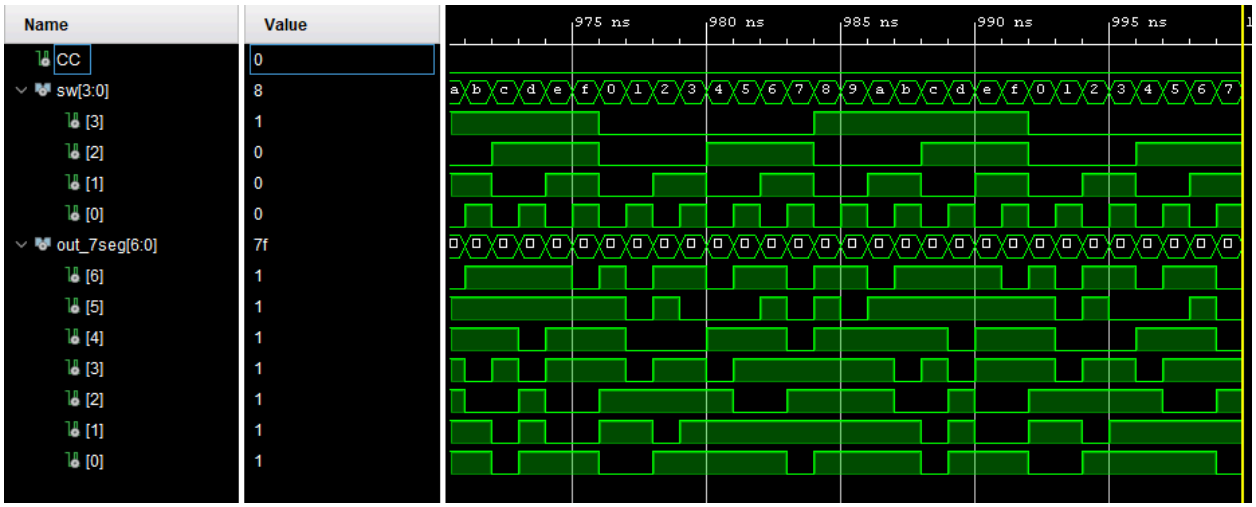


Fig. 3.5: Photo of a logic window taken in the Waveform software for Part 2.

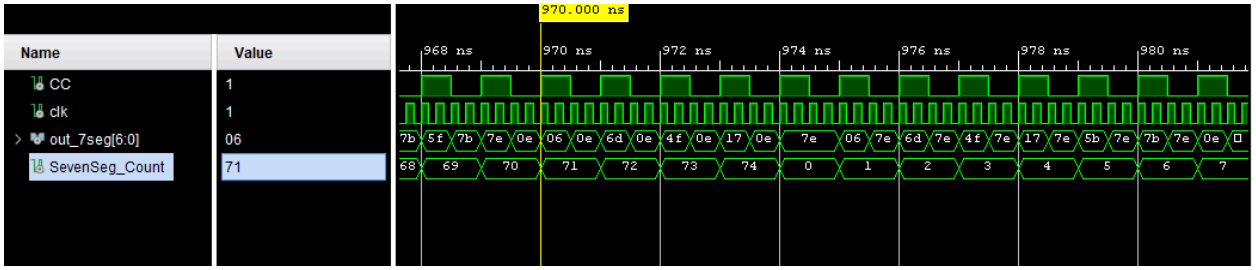


Fig. 3.6: Photo of a logic window taken in the Waveform software for Part 3.

DISCUSSION

Conclusions from Results

The results from all parts of the experiment confirmed the successful implementation of the design. In Part 1, the 7-segment decoder simulation accurately displayed the expected hexadecimal outputs (0-F) for all input combinations, as validated by the waveform analysis. Part 2 demonstrated the effectiveness of using conditional statements in VHDL to simplify the design. The results matched those of Part 1, showing that the conditional implementation was both concise and functional. In Part 3, the binary counter correctly incremented through the required range, displaying the last two digits of the student ID on the 7-segment display, with waveforms confirming proper timing and operation.

Uncertainties and Ambiguities

Minor uncertainties could arise from potential timing mismatches due to clock synchronization or propagation delays, especially in Part 3 where accurate timing was crucial. Additionally, ambiguities may exist in interpreting waveform results if glitches or noise are present, which can complicate debugging and validation.

Comparison to Expected Results

The experimental results closely matched the expected outcomes. In Part 1, the truth table and waveforms aligned perfectly with the theoretical values. Part 2's implementation demonstrated equivalent outputs while offering a more streamlined coding approach. Part 3's counter operation also adhered to the expected behavior, displaying the correct sequence of digits. These outcomes validate both the design and the implementation process.

Overall, the experiment successfully demonstrated the practical application of VHDL in designing and testing digital systems, with all parts performing as intended.

CONCLUSION

This lab successfully demonstrated the principles of combinational and sequential logic design. The implementation of a 7-segment LED decoder using minimized logic and VHDL highlighted the importance of optimization. Developing an up-counter provided hands-on experience with synchronous design and hardware verification. These concepts are fundamental in digital system design and FPGA applications.

REFERENCES

1. *ECE 210 Lab Manual 3*, University of Alberta, 2024. [Online]. Available: eClass. [Accessed: Nov. 18, 2024].