Laboratory Report #5

Name: Josh Ratificar **Date Completed:** 11-10-2023

Laboratory Exercise Title: Behavioral Modeling of Combinational Circuits

TJ Albiana **(1984) ← ME**

Rafinan **← ECE**

Block Diagrams:

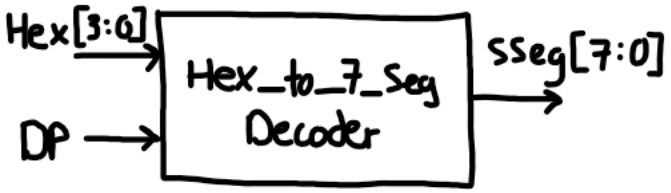


Figure 1.0 – *Hex_to_7_Seg_Decoder Block Diagram*

Figure 2.0 – *n_bit_ALU Block Diagram*

Exercise 5A:

Figure 1.1 – *Hex_to_7seg_Decoder.v Script*

```
/***********************************
*   FILE:               Hex_to_7seg_Decoder.v
*   AUTHOR:             Mohan Francis, Josh Ratificar
*   Class:              Gr.3 CpE 3101L Introduction to HDL
*   Group/Schedule      Friday, 7:30 AM to 10:30 AM
*   Description:        Hex_to_7seg_Decoder.v file
***********************************/
module Hex_to_7seg_Decoder (
    input [3:0] Hex,
    input DP,
    output reg [7:0] SSeg
);
    always @(*)
    begin
        case ({DP, Hex})
            // DP = Off
            5'b00000 : SSeg = 8'b00000001; // SSeg output = 0
            5'b00001 : SSeg = 8'b10011111; // SSeg output = 1
            5'b00010 : SSeg = 8'b00100101; // SSeg output = 2
            5'b00011 : SSeg = 8'b00001101; // SSeg output = 3
            5'b00100 : SSeg = 8'b10011001; // SSeg output = 4
            5'b00101 : SSeg = 8'b01001001; // SSeg output = 5
            5'b00110 : SSeg = 8'b01000001; // SSeg output = 6
            5'b00111 : SSeg = 8'b00011111; // SSeg output = 7
            5'b01000 : SSeg = 8'b00000001; // SSeg output = 8
            5'b01001 : SSeg = 8'b00001001; // SSeg output = 9
            5'b01010 : SSeg = 8'b00010001; // SSeg output = A
            5'b01011 : SSeg = 8'b11000001; // SSeg output = B
            5'b01100 : SSeg = 8'b01100011; // SSeg output = C
            5'b01101 : SSeg = 8'b10000101; // SSeg output = D
            5'b01110 : SSeg = 8'b01100001; // SSeg output = E
            5'b01111 : SSeg = 8'b01110001; // SSeg output = F


            // DP = On
            5'b10000 : SSeg = 8'b00000000; // SSeg output = 0
            5'b10001 : SSeg = 8'b10011110; // SSeg output = 1
            5'b10010 : SSeg = 8'b00100100; // SSeg output = 2
            5'b10011 : SSeg = 8'b00001100; // SSeg output = 3
            5'b10100 : SSeg = 8'b10011000; // SSeg output = 4
            5'b10101 : SSeg = 8'b01001000; // SSeg output = 5
```

```
            5'b10110 : SSeg = 8'b01000000; // SSeg output = 6
            5'b10111 : SSeg = 8'b00011110; // SSeg output = 7
            5'b11000 : SSeg = 8'b00000000; // SSeg output = 8
            5'b11001 : SSeg = 8'b00001000; // SSeg output = 9
            5'b11010 : SSeg = 8'b00010000; // SSeg output = A
            5'b11011 : SSeg = 8'b11000000; // SSeg output = B
            5'b11100 : SSeg = 8'b01100010; // SSeg output = C
            5'b11101 : SSeg = 8'b10000100; // SSeg output = D
            5'b11110 : SSeg = 8'b01100000; // SSeg output = E
            5'b11111 : SSeg = 8'b01110000; // SSeg output = F
        endcase
    end
endmodule
```

Figure 1.2 – *Hex_to_7seg_Decoder.v Analysis and Elaboration Test Results*

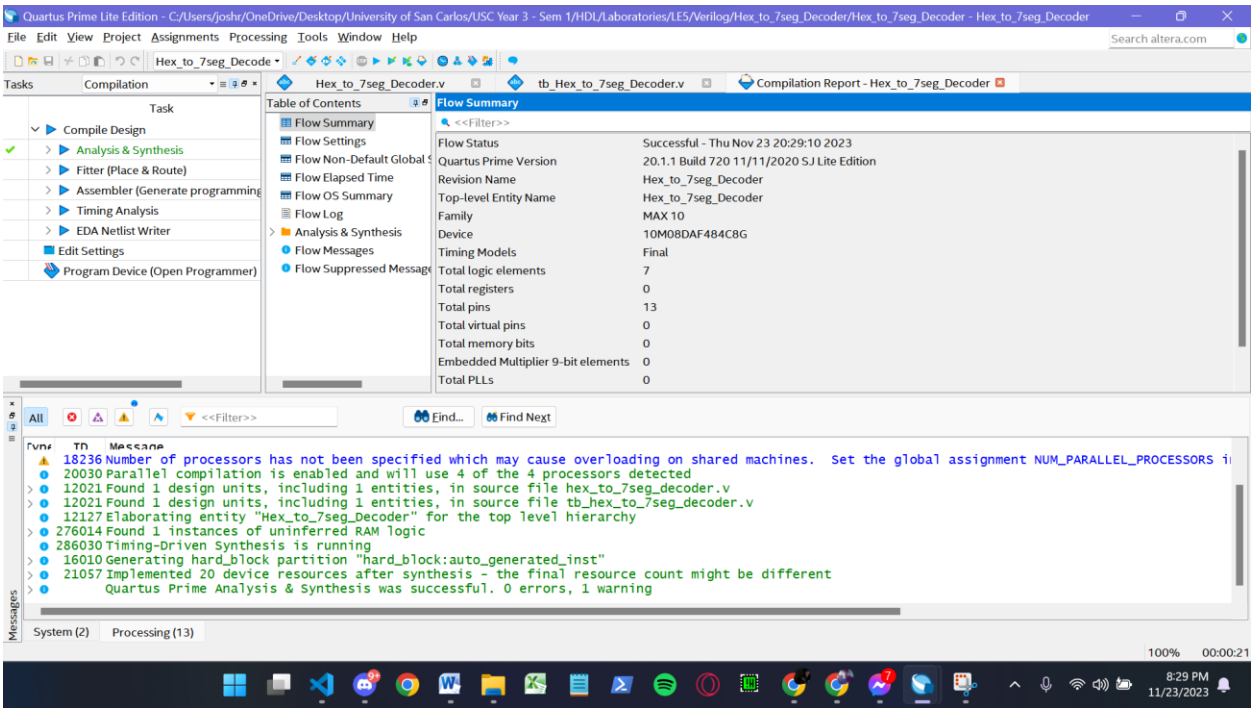


Figure 1.3 – *Hex_to_7seg_Decoder RTL View Output*

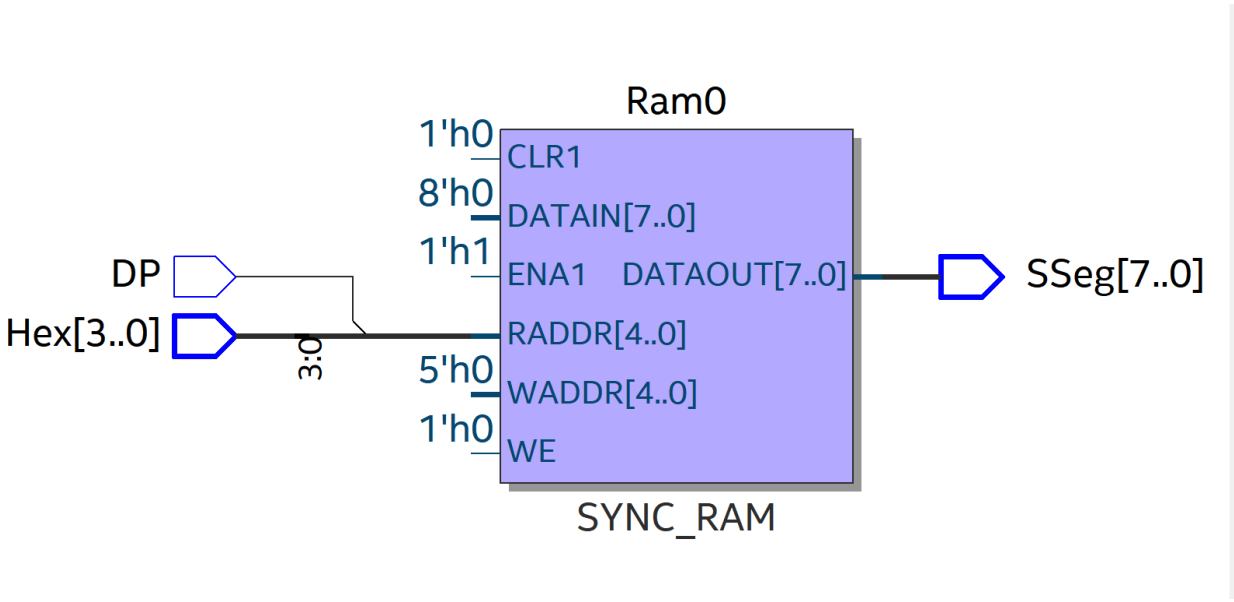


Figure 1.4 – *tb_ Hex_to_7seg_Decoder.v Script for Testing Module*

```
/***********************************
*   FILE:               Hex_to_7seg_Decoder.v
*   AUTHOR:             Mohan Francis, Josh Ratificar
*   Class:              Gr.3 CpE 3101L Introduction to HDL
*   Group/Schedule      Friday, 7:30 AM to 10:30 AM
*   Description:        test bench Hex_to_7seg_Decoder.v file
***********************************/
module tb_Hex_to_7seg_Decoder();
    reg  [3:0] Hex;
    reg  DP;
    wire [7:0] SSeg;

    Hex_to_7seg_Decoder UUT(
        .Hex(Hex),
        .DP(DP),
        .SSeg(SSeg)
    );
    initial begin
        $display("Test Bench | Hex_to_7seg_Decoder...");
        $display("DP is OFF...");
        DP = 1'b0;       Hex = 4'b0000;     #10;
        DP = 1'b0;      Hex = 4'b0001;      #10;
        DP = 1'b0;      Hex = 4'b0010;      #10;
        DP = 1'b0;      Hex = 4'b0011;      #10;
        DP = 1'b0;      Hex = 4'b0100;      #10;
```

```
            DP = 1'b0;      Hex = 4'b0101;      #10;
            DP = 1'b0;      Hex = 4'b0110;      #10;
            DP = 1'b0;      Hex = 4'b0111;      #10;
            DP = 1'b0;      Hex = 4'b1000;      #10;
            DP = 1'b0;      Hex = 4'b1001;      #10;
            DP = 1'b0;      Hex = 4'b1010;      #10;
            DP = 1'b0;      Hex = 4'b1011;      #10;
            DP = 1'b0;      Hex = 4'b1100;      #10;
            DP = 1'b0;      Hex = 4'b1101;      #10;
            DP = 1'b0;      Hex = 4'b1110;      #10;
            DP = 1'b0;      Hex = 4'b1111;      #10;

        $display("DP is ON...");
            DP = 1'b1;      Hex = 4'b0000;      #10;
            DP = 1'b1;      Hex = 4'b0001;      #10;
            DP = 1'b1;      Hex = 4'b0010;      #10;
            DP = 1'b1;      Hex = 4'b0011;      #10;
            DP = 1'b1;      Hex = 4'b0100;      #10;
            DP = 1'b1;      Hex = 4'b0101;      #10;
            DP = 1'b1;      Hex = 4'b0110;      #10;
            DP = 1'b1;      Hex = 4'b0111;      #10;
            DP = 1'b1;      Hex = 4'b1000;      #10;
            DP = 1'b1;      Hex = 4'b1001;      #10;
            DP = 1'b1;      Hex = 4'b1010;      #10;
            DP = 1'b1;      Hex = 4'b1011;      #10;
            DP = 1'b1;      Hex = 4'b1100;      #10;
            DP = 1'b1;      Hex = 4'b1101;      #10;
            DP = 1'b1;      Hex = 4'b1110;      #10;
            DP = 1'b1;      Hex = 4'b1111;      #10;
            $stop;
        end
        initial begin
            $monitor("Time = %2d ns | DP = %b | HEX = %b | SSeg = %b", $time, DP, Hex,
SSeg);
        end
    endmodule
```

Figure 1.5 – *Hex_to_7seg_Decoder RTL Simulation Output*

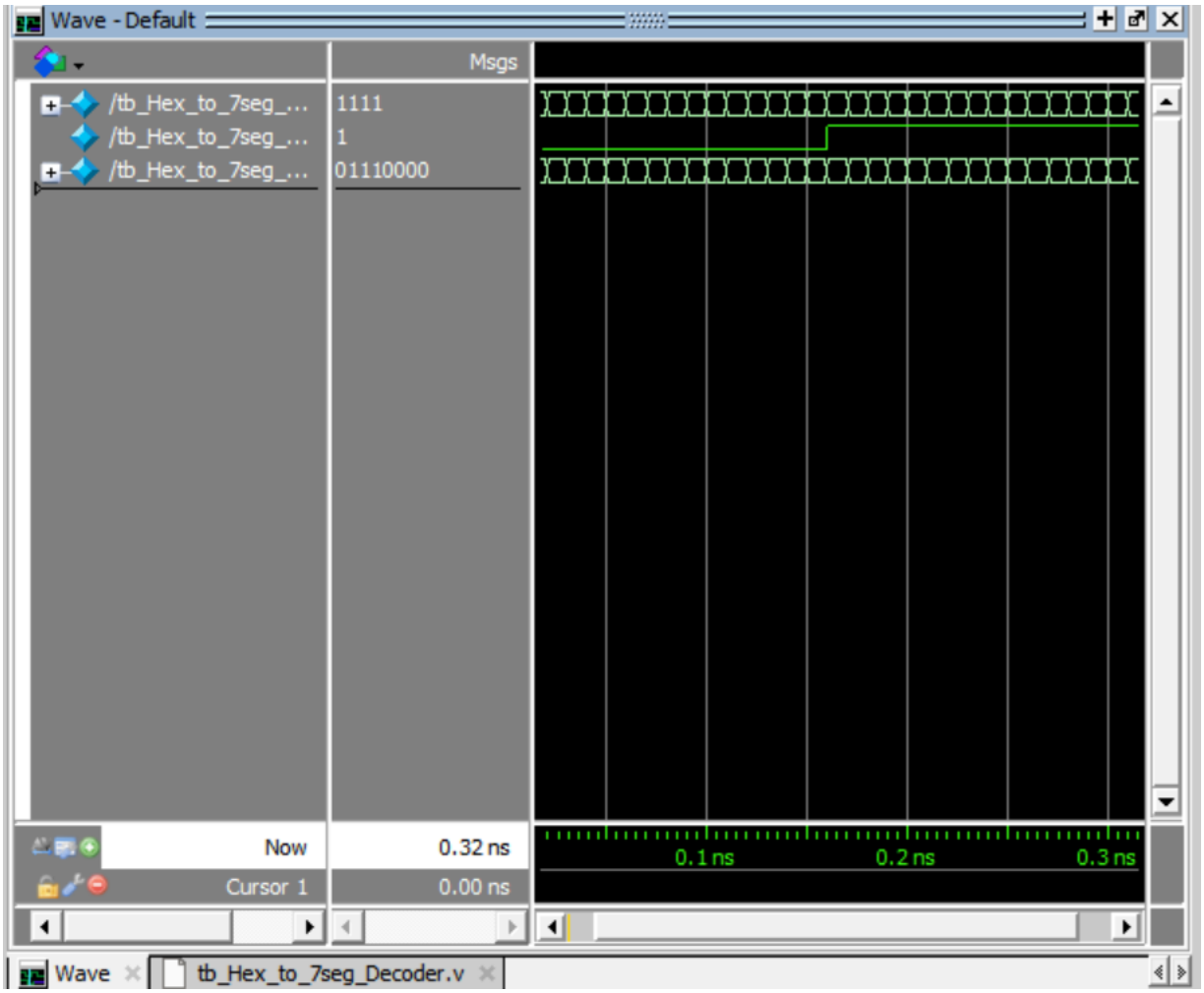
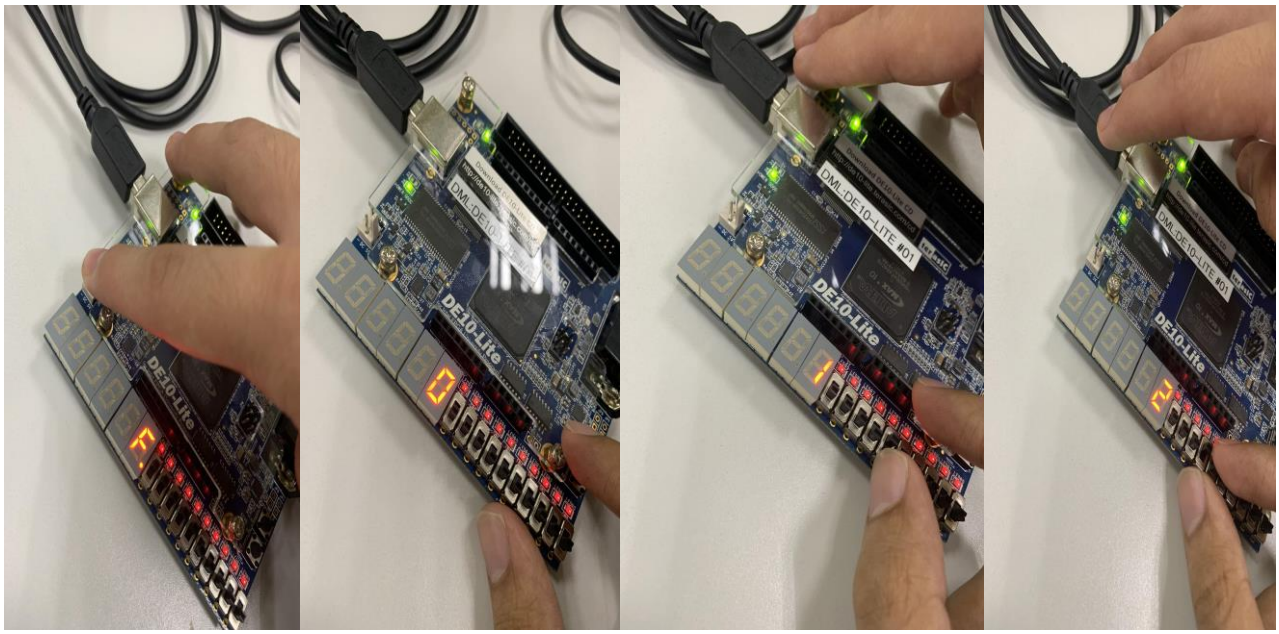
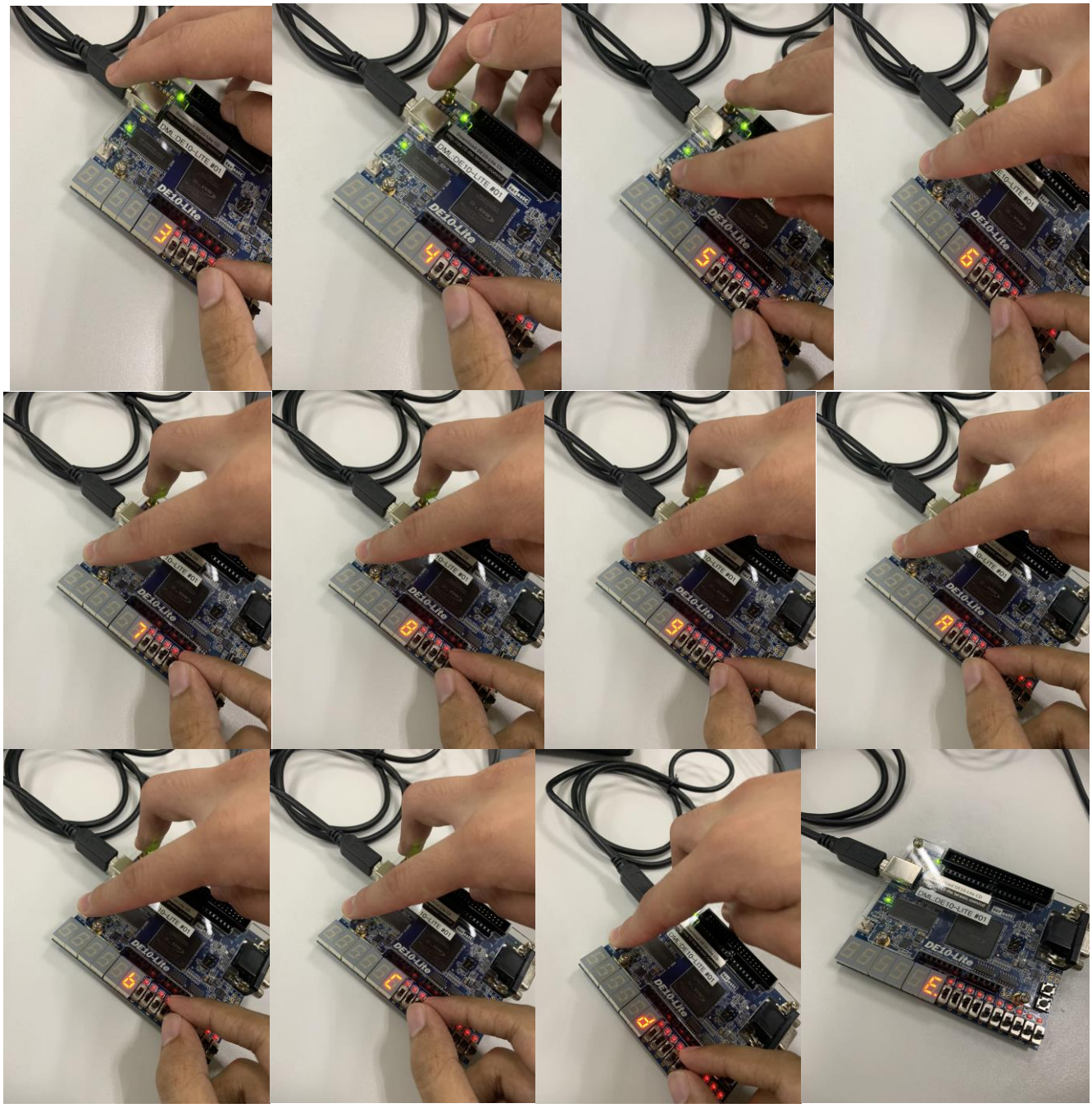


Figure 1.6 – *Hex_to_7seg_Decoder Test Bench Monitor Output (Annotations to **Figure 1.5**)*

```
Transcript
# Test Bench | Hex_to_7seg_Decoder...
# DP is OFF...
# Time =   0 ns | DP = 0 | HEX = 0000 | SSeg = 11111111
# Time =  10 ns | DP = 0 | HEX = 0001 | SSeg = 10011111
# Time =  20 ns | DP = 0 | HEX = 0010 | SSeg = 00100101
# Time =  30 ns | DP = 0 | HEX = 0011 | SSeg = 00001101
# Time =  40 ns | DP = 0 | HEX = 0100 | SSeg = 10011001
# Time =  50 ns | DP = 0 | HEX = 0101 | SSeg = 01001001
# Time =  60 ns | DP = 0 | HEX = 0110 | SSeg = 01000001
# Time =  70 ns | DP = 0 | HEX = 0111 | SSeg = 00011111
# Time =  80 ns | DP = 0 | HEX = 1000 | SSeg = 00000001
# Time =  90 ns | DP = 0 | HEX = 1001 | SSeg = 00001001
# Time = 100 ns | DP = 0 | HEX = 1010 | SSeg = 00010001
# Time = 110 ns | DP = 0 | HEX = 1011 | SSeg = 11000001
# Time = 120 ns | DP = 0 | HEX = 1100 | SSeg = 01100001
# Time = 130 ns | DP = 0 | HEX = 1101 | SSeg = 10000101
# Time = 140 ns | DP = 0 | HEX = 1110 | SSeg = 01100001
# Time = 150 ns | DP = 0 | HEX = 1111 | SSeg = 01110001
# DP is ON...
# Time = 160 ns | DP = 1 | HEX = 0000 | SSeg = 11111110
# Time = 170 ns | DP = 1 | HEX = 0001 | SSeg = 10011110
# Time = 180 ns | DP = 1 | HEX = 0010 | SSeg = 00100100
# Time = 190 ns | DP = 1 | HEX = 0011 | SSeg = 00001100
# Time = 200 ns | DP = 1 | HEX = 0100 | SSeg = 10011000
# Time = 210 ns | DP = 1 | HEX = 0101 | SSeg = 01001000
# Time = 220 ns | DP = 1 | HEX = 0110 | SSeg = 01000000
# Time = 230 ns | DP = 1 | HEX = 0111 | SSeg = 00011110
# Time = 240 ns | DP = 1 | HEX = 1000 | SSeg = 00000000
# Time = 250 ns | DP = 1 | HEX = 1001 | SSeg = 00001000
# Time = 260 ns | DP = 1 | HEX = 1010 | SSeg = 00010000
# Time = 270 ns | DP = 1 | HEX = 1011 | SSeg = 11000000
# Time = 280 ns | DP = 1 | HEX = 1100 | SSeg = 01100000
# Time = 290 ns | DP = 1 | HEX = 1101 | SSeg = 10000100
# Time = 300 ns | DP = 1 | HEX = 1110 | SSeg = 01100000
# Time = 310 ns | DP = 1 | HEX = 1111 | SSeg = 01110000
# ** Note: $stop    : C:/Users/joshr/OneDrive/Desktop/University of San Carlos/USC Year 3 - Sem 1/HDL/Laboratories/LE5/Verilog/Hex_to_7seg_Decoder/tb_Hex_to_7seg_Decoder.v(55)
#    Time: 320 ps  Iteration: 0  Instance: /tb_Hex_to_7seg_Decoder
```

Figure 1.7 – *FPGA Outputs*





Discussion of Results (Exercise 5A)

The outcomes depicted in **Figure 1.5** reveal that the Verilog script adheres to our anticipated behavior as per our programming directives. Corroborating this assertion, **Figure 1.6** substantiates the characteristics of the Hex_to_7_seg behavior. Moreover, subsequent to validation through the test bench, we proficiently programmed the Field-Programmable Gate Array (FPGA) to produce the designated results, as illustrated in **Figure 1.7**.

Exercise 5B:

Figure 2.1 – *ALU_n_bit.v Script*

```
/***********************************
*   FILE:                   ALU_n_bit.v
*   AUTHOR:             Josh Ratificar
*   Class:              Gr.3 CpE 3101L Introduction to HDL
*   Group/Schedule      Friday, 7:30 AM to 10:30 AM
*   Description:        ALU_n_bit.v file
***********************************/
/*****************************
*
*   Mode        Operation
*   000         Addition (A + B) with carry     0
*   001         Subtract (A - B) with borrow        1
*   010         Bitwise AND of A and B              2
*   011         Bitwise OR of A and B               3
*   100         Bitwise XOR of A and B              4
*   101         Complement of A                     5
*   110         Increment A                         6
*   111         Decrement A                         7
*
*****************************/
module ALU_n_bit
#(parameter n = 4)
(
    input [(n-1):0] A,
    input [(n-1):0] B,
    input CB_in,
    input [2:0] Mode,
    output reg [(n-1):0]Result,
    output reg CB_out
);
    always @(*)
    begin
    case (Mode)
        3'b000  :   begin // 0: Addition (A + B) with carry
                        // 3 + 1 = 4 1000
                        // 11  01     0     | 1000 > 111
                        Result = A + B + CB_in;
                        CB_out  = (A + B + CB_in) > ((2**n)-1);
                    end
```

```
            3'b001  :   begin // 1: Subtract (A - B) with borrow
                            // A = 4'b1111;    B = 4'b0001;
                            // 1111 - 0001
                            // edge cases satisfied...
                            /*
                            *   edge cases:
                            *       5 - 9  -> 1 0100B
                            *       15 - 4 -> 0 1010B
                            *       7 - 12 -> 1 1011B
                            *       15 - 12 -> 1 1011B
                            */
                            Result = ((A + CB_in) < B)? (B - (A+CB_in)) : ((A+CB_in) -
B);
                            CB_out = ((A + CB_in) < B)? 1'b1 : 1'b0;
                        end
            3'b010  :   begin // Bitwise AND of A and B
                            Result = A & B;
                            CB_out = 0;
                        end
            3'b011  :   begin // Bitwise OR of A and B
                            Result = A | B;
                            CB_out = 0;
                        end
            3'b100  :   begin // Bitwise XOR of A and B
                            Result = (A ^ B);
                            CB_out = 0;
                        end
            3'b101  :   begin // Complement of A
                            Result = ~A;
                            CB_out = 0;
                        end
            3'b110  :   begin // Increment A
                            Result = A + 1; // 111 + 1 -> 1000
                            CB_out = (A == ((2**n) - 1)); // When A = Max Term, CB_out
is high
                        end
            default :   begin // Decrement A
                            Result = (A < 1)? (1 - A) : (A - 1);
                            CB_out = (A < 1)? 1'b1 : 1'b0;
                        end
        endcase
        end
    endmodule
```

Figure 2.2 – *ALU_n_bit.v Analysis and Elaboration Test Results*

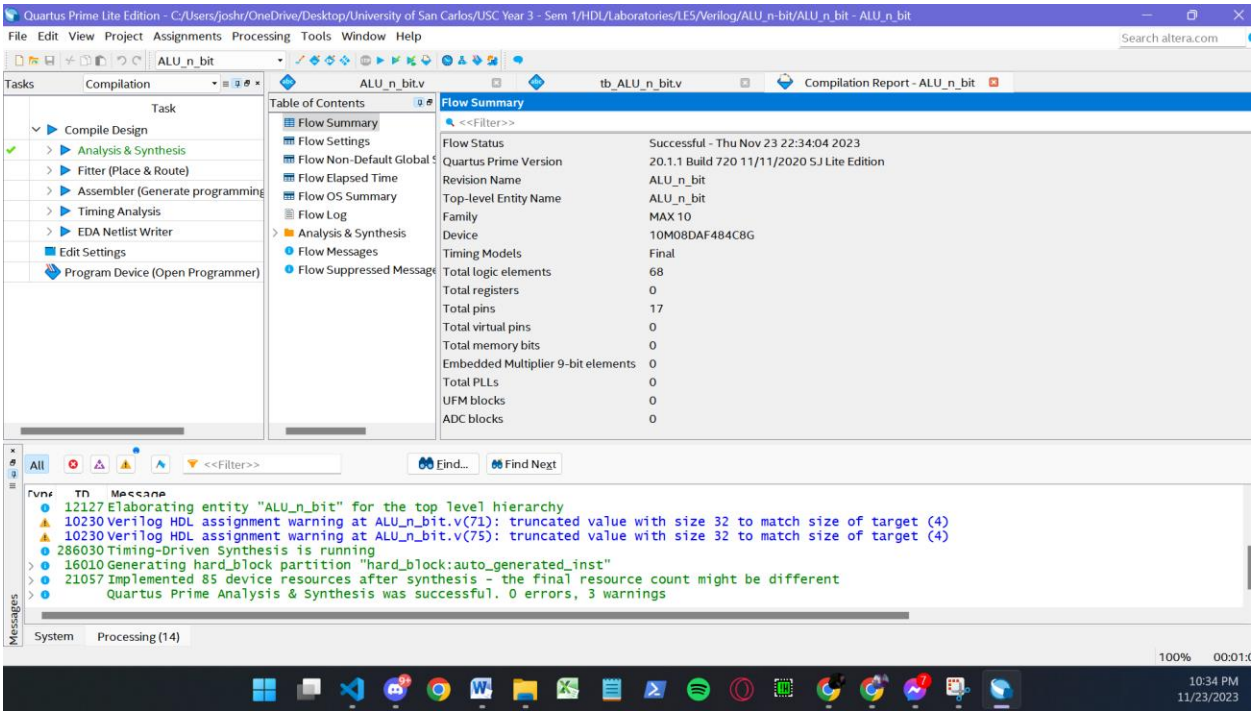


Figure 2.3 – *ALU_n_bit RTL View Output*

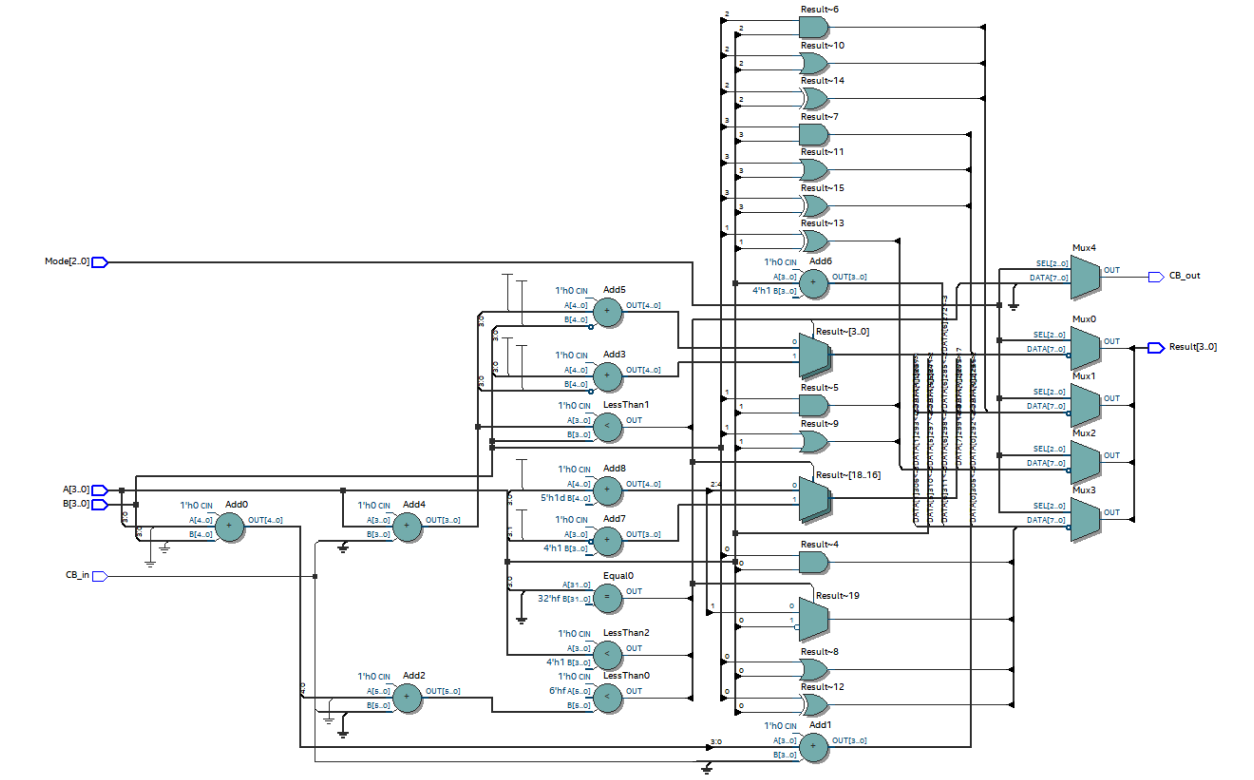


Figure 2.4 – *tb_ALU_4_bit.v Script for Testing Module*

```
    /***********************************
    *   FILE:                   tb_ALU_n_bit.v
    *   AUTHOR:             Josh Ratificar
    *   Class:              Gr.3 CpE 3101L Introduction to HDL
    *   Group/Schedule      Friday, 7:30 AM to 10:30 AM
    *   Description:        Test Bench ALU_n_bit.v file
    ***********************************/
    `timescale 1 ns / 1 ps
    module tb_ALU_n_bit();
        reg [3:0] A, B;
        reg CB_in;
        reg [2:0] Mode;
        wire [3:0] Result;
        wire CB_out;

        ALU_n_bit #(.n(4)) UUT(
            .A(A),
            .B(B),
            .CB_in(CB_in),
            .Mode(Mode),
            .Result(Result),
            .CB_out(CB_out)
        );
        initial begin
            $display("Test Bench | 4-bit ALU_n_bit...");
            $display("Addition with Carry...");
            Mode = 3'b000;    A = 4'b0000;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 0 + 1 = 1          |  00001
            Mode = 3'b000;    A = 4'b1111;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 15 + 1 = 16        |  10000
            Mode = 3'b000;    A = 4'b0010;    B = 4'b0100;    CB_in =
1'b0;    #10;    // 2 + 4 = 6          |  00110

            $display("Subtraction with Borrow...");
            Mode = 3'b001;    A = 4'b0000;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 0 - 1 = 1          |  1 0001
            Mode = 3'b001;    A = 4'b1111;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 15 - 1 = 14        |  0 1110
            Mode = 3'b001;    A = 4'b0010;    B = 4'b0100;    CB_in =
1'b0;    #10;    // 2 - 4 = 6          |  1 0110
```

```
            $display("Bitwise AND of A and B...");
            Mode = 3'b010;    A = 4'b0000;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 0 AND 1 = 0       |  00000
            Mode = 3'b010;    A = 4'b1111;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 15 AND 1 = 1      |  00001
            Mode = 3'b010;    A = 4'b0010;    B = 4'b0100;    CB_in =
1'b0;    #10;    // 2 AND 4 = 0       |  00000

            $display("Bitwise OR of A and B...");
            Mode = 3'b011;    A = 4'b0000;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 0 OR 1 = 1        |  00000
            Mode = 3'b011;    A = 4'b1111;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 15 OR 1 = 15      |  01111
            Mode = 3'b011;    A = 4'b0010;    B = 4'b0100;    CB_in =
1'b0;    #10;    // 2 OR 4 = 6        |  00110

            $display("Bitwise XOR of A and B...");
            Mode = 3'b100;    A = 4'b0000;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 0 XOR 1 = 1     |  00000
            Mode = 3'b100;    A = 4'b1111;    B = 4'b0001;    CB_in =
1'b0;    #10;    // 15 XOR 1 = 14 |  01110
            Mode = 3'b100;    A = 4'b0010;    B = 4'b0100;    CB_in =
1'b0;    #10;    // 2 XOR 4 = 6     |  00110

            $display("Complement of A...");
            Mode = 3'b101;    A = 4'b0000;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 0 = 15            |  01111
            Mode = 3'b101;    A = 4'b1111;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 15 = 0               |  00000
            Mode = 3'b101;    A = 4'b0010;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 2 = 13            |   01101

            $display("Increment A...");
            Mode = 3'b110;    A = 4'b0000;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 0 = 1             |  00001
            Mode = 3'b110;    A = 4'b1111;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 15 = 16           |  10000
            Mode = 3'b110;    A = 4'b0010;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 2 = 3             |  00011

            $display("Decrement A...");
            Mode = 3'b111;    A = 4'b0000;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 0 = 1   |  00001
```

```
            Mode = 3'b111;    A = 4'b1111;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 15 = 14 |  01110
            Mode = 3'b111;    A = 4'b0010;    B = 4'b0000;    CB_in =
1'b0;    #10;    // 2 = 1   |  00001
            $stop;
        end
        initial begin
            $monitor("Time = %2d ns | Mode = %b | A = %b | B = %b | CB_in = %b | Result
= %b | CB_out = %b", $time, Mode, A, B, CB_in, Result, CB_out);
        end
    endmodule
```

Figure 2.5 – *tb_ALU_n_bit RTL Simulation Output*

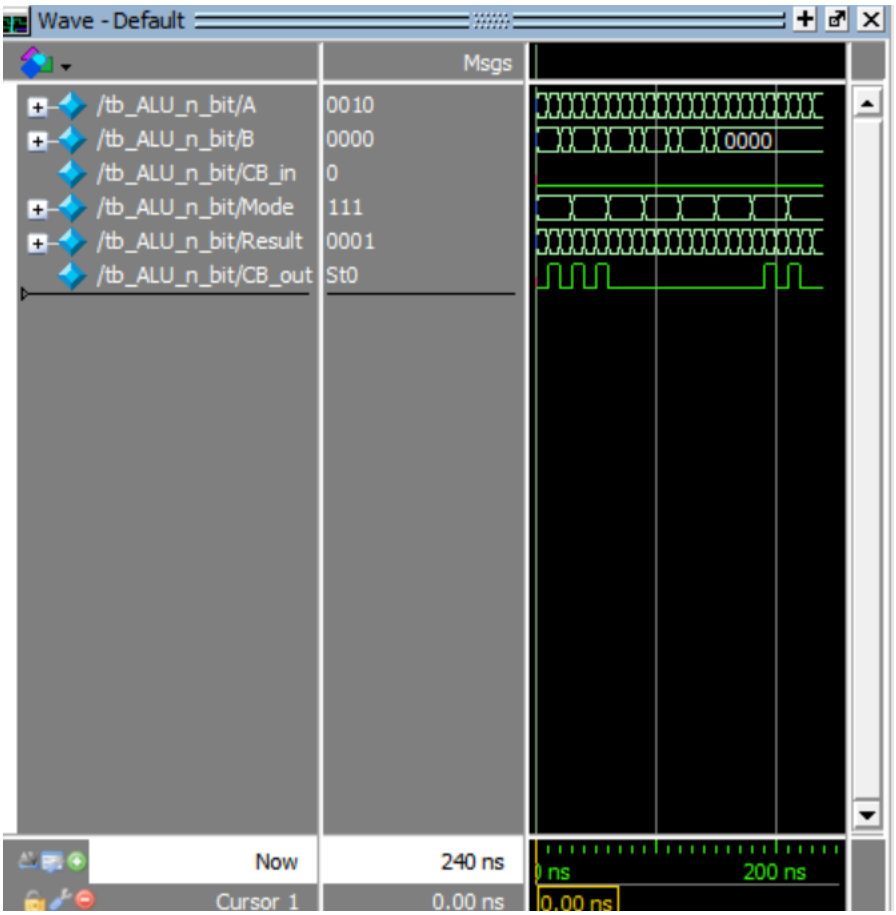


Figure 2.6 – *tb_ALU_n_bit Test Bench Monitor Output (Annotations to **Figure 2.5**)*

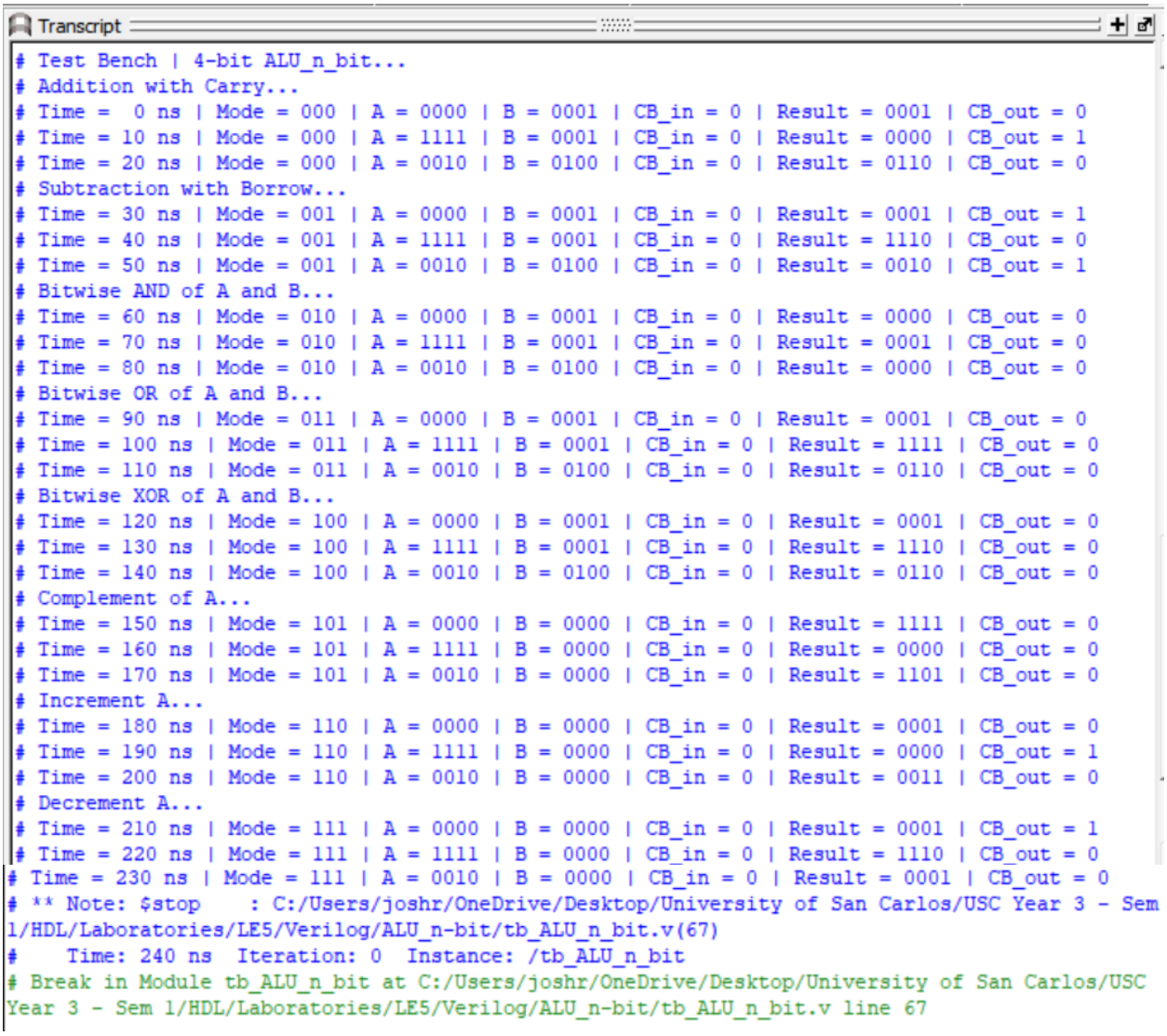
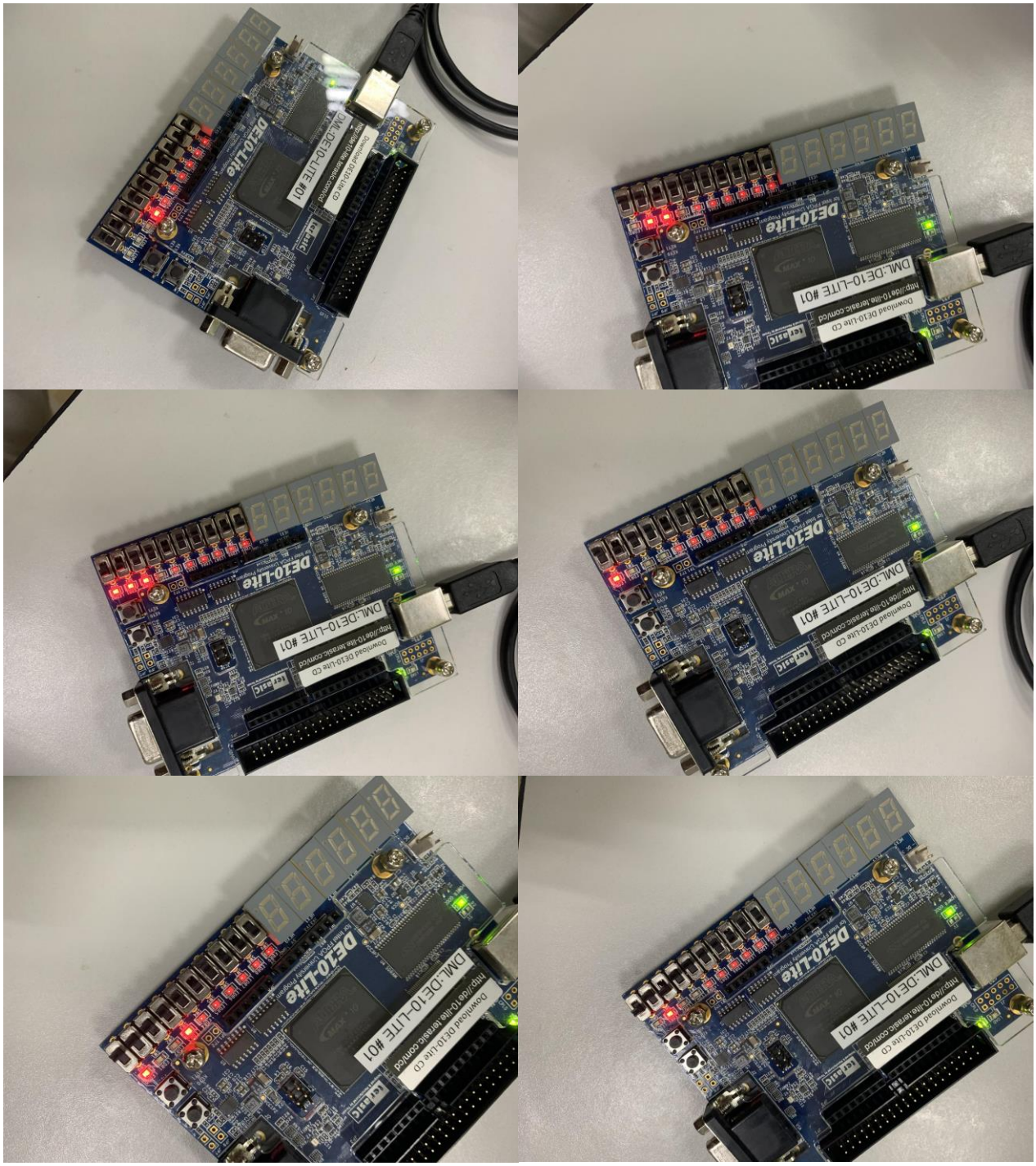


Figure 2.7 – *FPGA Outputs*



Discussion of Results (Exercise 5B)

In the context of this experiment, a behavioural structured Verilog script was effectively employed to attain the intended functionality of an n-bit Arithmetic Logic Unit (ALU). This accomplishment is elucidated in **Figure 2.5**, specifically crafted for testing 4-bit inputs. Furthermore, validation is substantiated by the utilization of the "$monitor" command in the Register-Transfer Level (RTL) simulation, as depicted in **Figure 2.6**.