National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

# Department of Electrical Engineering and Computer Science

Faculty Member: Dr. Rehan Ahmed Dated: 22/02/2023

Semester: 6th Section: BEE 12C

## EE-421: Digital System Design

## Lab 3: Combinational - Circuits Building Blocks

## Group Members

| | | PLO4-CLO3 | | PLO5 - CLO4 | PLO8 - CLO5 | PLO9 - CLO6 |
|---|---|---|---|---|---|---|
| Name | Reg. No | Viva / Quiz / Lab Performance | Analysis of data in Lab Report | Modern Tool Usage | Ethics and Safety | Individual and Teamwork |
| | | 5 Marks | 5 Marks | 5 Marks | 5 Marks | 5 Marks |
| Danial Ahmad | 331388 | | | | | |
| Muhammad Umer | 345834 | | | | | |
| Tariq Umar | 334943 | | | | | |

## 1 Table of Contents

# 2 Combinational - Circuits Building Blocks

## 2.1 Objectives

The purpose of this exercise is to learn how to design combinational circuits that can perform binary-to-decimal number conversion and binary-coded-decimal (BCD) addition. We will use the switches on the DE-series boards as inputs to the circuit. We will use light emitting diodes (LEDs) and 7-segment displays as output devices.

## 2.2 Introduction

The use of combination circuits is ubiquitous in the field of electronics, particularly in digital electronics. These circuits are designed to perform logical operations using input signals, producing an output signal based on the logical conditions. A combinational circuit's output depends solely on its input signals, and it has no memory of previous inputs. Through this lab exercise, we aim to gain a practical understanding of how combination circuits work and how they can be used to perform binary-to-decimal number conversion and binary-coded-decimal (BCD) addition. We will also gain experience with Quartus Prime and learn how to use it to design and test digital circuits.

## 2.3 Software

Quartus Prime is a comprehensive design software developed by Intel Corporation for designing digital circuits using Field-Programmable Gate Arrays (FPGAs). It is a leading software platform in the field of digital design, offering a range of advanced tools and features that enable users to easily create, debug, and verify complex digital circuits. With Quartus Prime, users can benefit from a streamlined design flow that facilitates the creation of digital circuits from concept to implementation. It provides an intuitive graphical user interface that allows users to easily design, test, and debug their circuits. Additionally, Quartus Prime supports a variety of popular programming languages, making it a versatile platform for digital designers of all levels.

## 3 Lab Procedure

### 3.1 Part I

You are to design a circuit that converts a four-bit binary number V = v3v2v1v0 into its two-digit decimal equivalent D = d1d0. Table 1 shows the required output values. A partial design of this circuit is given in Figure 1. It includes a comparator that checks when the value of V is greater than 9 and uses the output of this comparator in the control of the 7-segment displays.

| $v_3v_2v_1v_0$ | $d_1$ | $d_0$ |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 0 | 1 |
| 0010 | 0 | 2 |
| ... | ... | ... |
| 1001 | 0 | 9 |
| 1010 | 1 | 0 |
| 1011 | 1 | 1 |
| 1100 | 1 | 2 |
| 1101 | 1 | 3 |
| 1110 | 1 | 4 |
| 1111 | 1 | 5 |

Table 1: Binary-to-decimal conversion values.

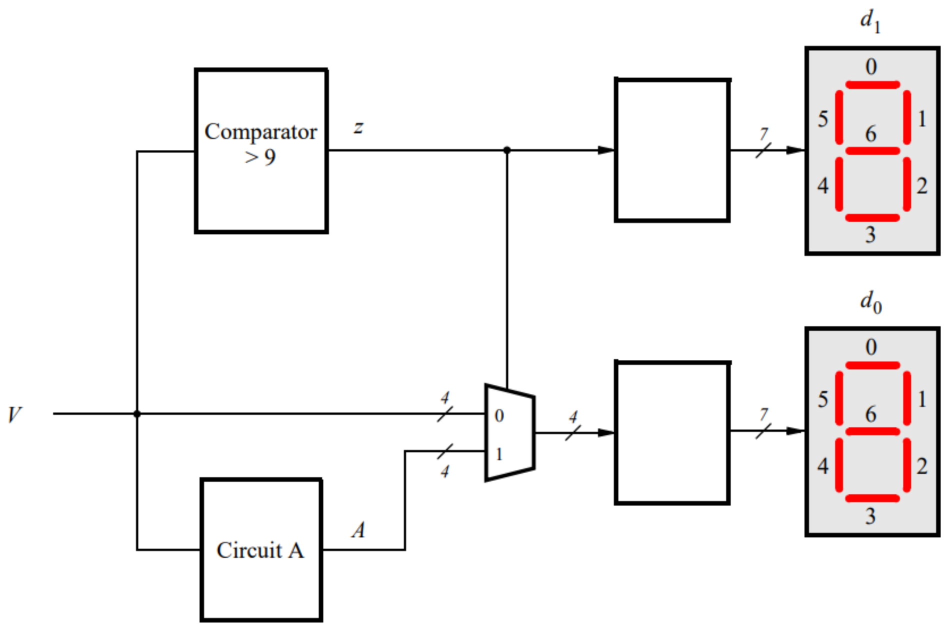


Figure 1: Partial design of the binary-to-decimal conversion circuit.

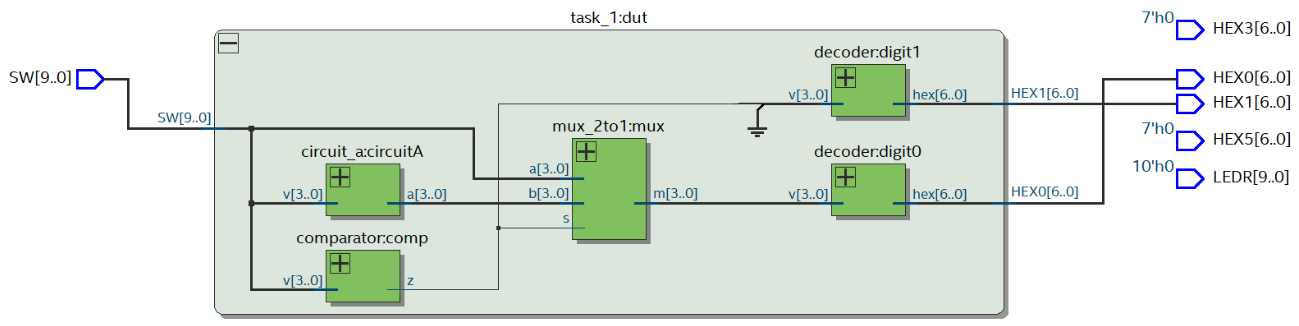
1. Write Verilog code to implement your design. The code should have the 4-bit input SW3−0, which should be used to provide the binary number V, and the two 7-bit outputs HEX1 and HEX0, to show the values of decimal digits d1 and d0. The intent of this exercise is to use simple Verilog

assign statements to specify the required logic functions using Boolean expressions. Your Verilog code should not include any if-else, case, or similar statements.

2. Make a Quartus project for your Verilog module.
3. Compile the circuit and use functional simulation to verify the correct operation of your comparator, multiplexers, and circuit A.
4. Download the circuit into an FPGA board. Test the circuit by trying all possible values of V and observing the output displays.

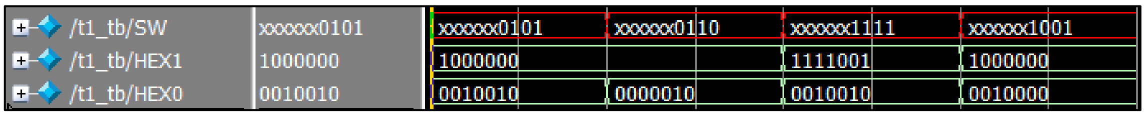
```
module task_1 (
    input  [9:0] SW,
    output [6:0] HEX0,
    output [6:0] HEX1
);

   wire [3:0] v = SW[3:0];
   wire [3:0] a, m;
   wire [3:0] z;
   assign z[3:1] = 3'b000;

   circuit_a circuitA (
       .v(v),
       .a(a)
   );

   comparator comp (
       .v(v),
       .z(z[0])
   );

   mux_2to1 #(
       .n(3)
   ) mux (
       .a(v),
       .b(a),
       .s(z[0]),
       .m(m)
   );

   decoder digit0 (
       .v  (m),
       .hex(HEX0)
   );
   decoder digit1 (
       .v  (z),
       .hex(HEX1)
   );

endmodule

module mux_2to1 #(
    parameter n = 4
) (
    input [n:0] a,
    input [n:0] b,
    input s,
    output reg [n:0] m
);
```

```verilog
   integer i;

   always @(*)
      for (i = 0; i <= n; i = i + 1) begin
         m[i] = (a[i] & ~s) | (b[i] & s);
      end

endmodule

module comparator ( // > 9 comparator
    input [3:0] v,
    output z
);

   assign z = v[3] & (v[2] | v[1]);

endmodule

module circuit_a (
    input  [3:0] v,
    output [3:0] a
);

   assign a[3] = 0;
   assign a[2] = v[3] & v[2] & v[1];
   assign a[1] = v[3] & v[2] & ~v[1];
   assign a[0] = (v[3] & v[0]) & (v[2] | v[1]);

endmodule

module decoder ( // 4-bit to 7-segment decoder
    input  [3:0] v,
    output [6:0] hex
);

   assign hex[0] = (v[2] & ~v[1] & ~v[0]) | (~v[3] & ~v[2] & ~v[1] & v[0]);
   assign hex[1] = (v[2] & ~v[1] & v[0]) | (v[2] & v[1] & ~v[0]);
   assign hex[2] = (~v[2] & v[1] & ~v[0]);
   assign hex[3] = (v[2] & ~v[1] & ~v[0]) | (v[2] & v[1] & v[0]) | (~v[3] & ~v[2] & ~v[1] &
v[0]);
   assign hex[4] = v[0] | (v[2] & ~v[1]);
   assign hex[5] = (~v[2] & v[1]) | (v[1] & v[0]) | (~v[3] & ~v[2] & v[0]);
   assign hex[6] = (~v[3] & ~v[2] & ~v[1]) | (v[2] & v[1] & v[0]);

endmodule
```



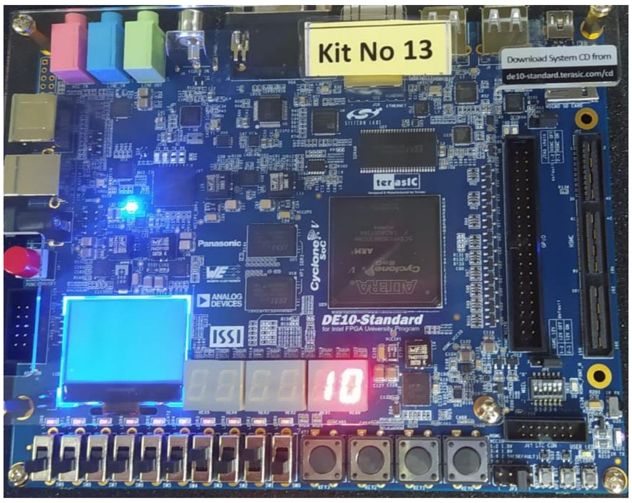
To test the desired functioning of the circuit, we implement the following testbench:

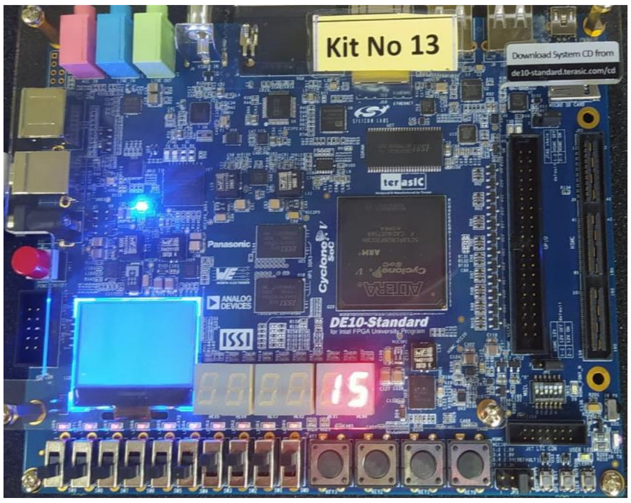
```
module t1_tb (); // testbench for task 1

   reg [9:0] SW;
   wire [6:0] HEX0, HEX1;

   task_1 dut (
       .SW  (SW),
       .HEX0(HEX0),
       .HEX1(HEX1)
   );

   initial begin
      SW = 10'bXXXXXX_0101;
      #10;
      SW = 10'bXXXXXX_0110;
      #10;
      SW = 10'bXXXXXX_1111;
      #10;
      SW = 10'bXXXXXX_1001;
      #10;
   end

endmodule
```



```
/t1_tb/SW     xxxxxx0101  xxxxxx0101  xxxxxx0110  xxxxxx1111  xxxxxx1001
/t1_tb/HEX1   1000000     1000000                 1111001     1000000
/t1_tb/HEX0   0010010     0010010     0000010     0010010     0010000
```

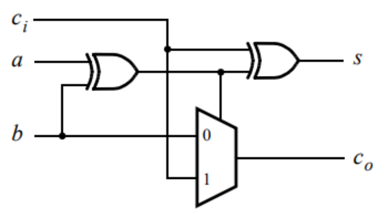
Following are sample hardware demonstrations of Part I on the DE-10 board:



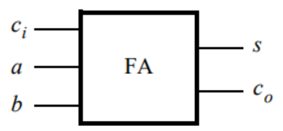


## 3.2 Part II

Figure 2a shows a circuit for a full adder, which has the inputs a, b, and ci, and produces the outputs s and co. Parts b and c of the figure show a circuit symbol and truth table for the full adder, which produces the two-bit binary sum cos = a + b + ci. Figure 2d shows how four instances of this full adder module can be used to design a circuit that adds two four-bit numbers. This type of circuit is usually called a ripple-carry adder, because of the way that the carry signals are passed from one full adder to the next. Write Verilog code that implements this circuit, as described below.



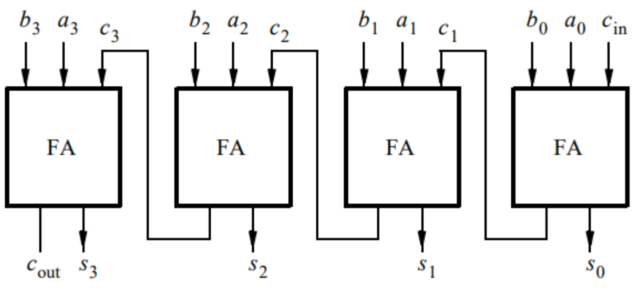
a) Full adder circuit



b) Full adder symbol

| $b$ | $a$ | $c_i$ | $c_o$ | $s$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

c) Full adder truth table

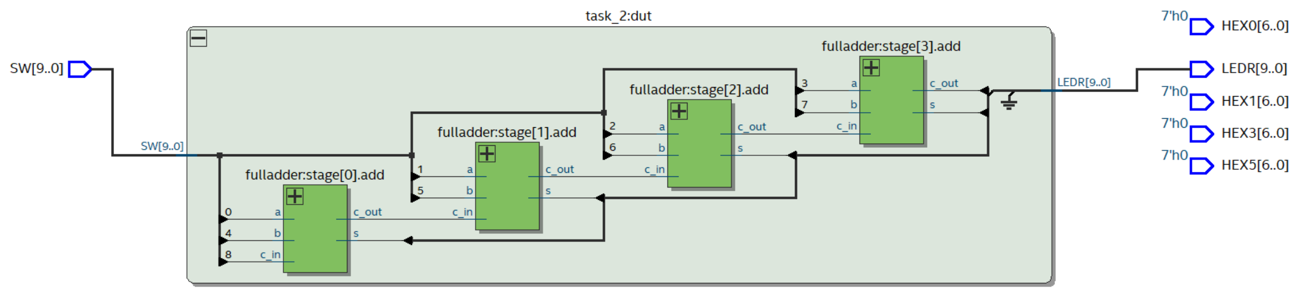


d) Four-bit ripple-carry adder circuit

Figure 2: A ripple-carry adder circuit.

1. Create a new Quartus project for the adder circuit. Write a Verilog module for the full adder subcircuit and write a top-level Verilog module that instantiates four instances of this full adder.
2. Use switches SW7−4 and SW3−0 to represent the inputs A and B, respectively. Use SW8 for the carry-in cin of the adder. Connect the outputs of the adder, cout and S, to the red lights LEDR.
3. Include the necessary pin assignments for your DE-series board, compile the circuit, and download it into the FPGA chip.
4. Test your circuit by trying different values for numbers A, B, and cin.

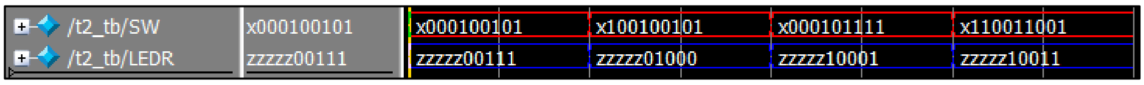
```
module fulladder (
    input  c_in,
    input  a,
    input  b,
    output s,
    output c_out
);

   wire x_s;
   assign x_s = a ^ b;
   assign s   = c_in ^ x_s;

   mux_2to1 #(
       .n(0)
   ) mux (
       .s(x_s),
       .a(b),
       .b(c_in),
       .m(c_out)
   );

endmodule

module task_2 (
    input  [9:0] SW,
    output [9:0] LEDR
);

   wire [3:0] a = SW[3:0];
   wire [3:0] b = SW[7:4];
   wire [3:0] s;
   wire [4:0] c;

   genvar i;
   assign c[0]  = SW[8];
   assign c_out = c[4];

   generate
      for (i = 0; i < 4; i = i + 1) begin : stage
         fulladder add (
             .c_in(c[i]),
             .a(a[i]),
             .b(b[i]),
             .s(s[i]),
             .c_out(c[i+1])
         );
      end
   endgenerate

   assign LEDR[3:0] = s[3:0];
   assign LEDR[4]   = c_out;

endmodule
```



To test the desired functioning of the circuit, we implement the following testbench:

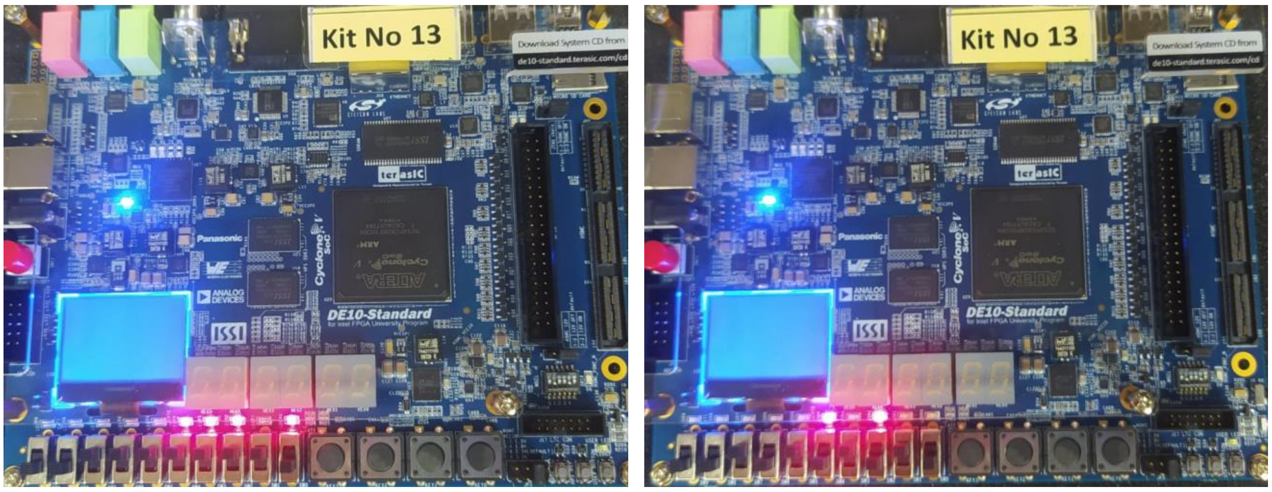
```
module t2_tb ();

   reg  [9:0] SW;
   wire [9:0] LEDR;

   task_2 test (
       .SW  (SW),
       .LEDR(LEDR)
   );

   initial begin
```

```
        SW = 10'bX_0_0010_0101;
        #10;
        SW = 10'bX_1_0010_0101;
        #10;
        SW = 10'bX_0_0010_1111;
        #10;
        SW = 10'bX_1_1001_1001;
        #10;
    end

endmodule
```



```
/t2_tb/SW     x000100101   x000100101   x100100101   x000101111   x110011001
/t2_tb/LEDR   zzzzz00111   zzzzz00111   zzzzz01000   zzzzz10001   zzzzz10011
```

Following are sample hardware demonstrations of Part II on the DE-10 board:



## 3.3 Part III

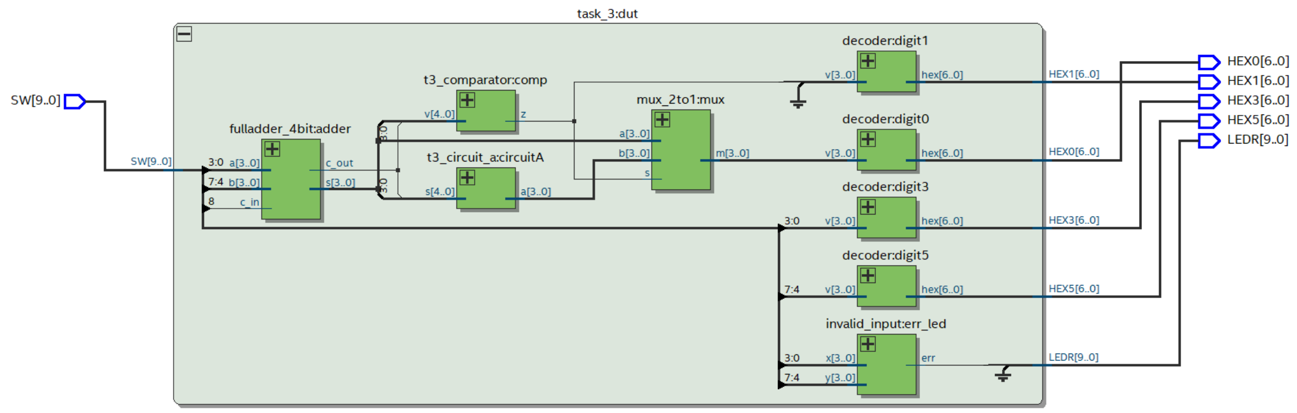
In part II we discussed the conversion of binary numbers into decimal digits. For this part you are to design a circuit that has two decimal digits, X and Y, as inputs. Each decimal digit is represented as a 4-bit number. In technical literature this is referred to as the binary coded decimal (BCD) representation. You are to design a circuit that adds the two BCD digits. The inputs to your circuit are the numbers X and Y, plus a carry-in, cin. When these inputs are added, the result will be a 5-bit binary number. But this result is to be displayed on 7-segment displays as a two-digit BCD sum S1S0. For a sum equal to zero you would display S1S0 = 00, for a sum of one S1S0 = 01, for nine S1S0 = 09, for ten S1S0 = 10, and so on. Note that the inputs X and Y are assumed to be decimal digits, which means that the largest sum that needs to be handled by this circuit is S1S0 = 9 + 9 + 1 = 19.

1. Create a new Quartus project for your BCD adder. Write your Verilog code using simple assign statements to specify the required logic functions–do not use other types of Verilog statements such as if-else or case statements for this part of the exercise.
2. Use switches SW7−4 and SW3−0 for the inputs X and Y, respectively, and use SW8 for the carry-in. Connect the four-bit sum and carry-out produced by the operation X + Y to the red lights

LEDR. Display the BCD values of X and Y on the 7-segment displays HEX5 and HEX3 and display the result S1S0 on HEX1 and HEX0.

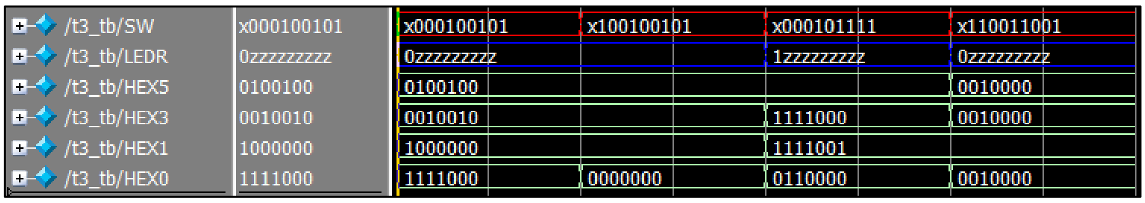
3. Since your circuit handles only BCD digits, check for the cases when the input X or Y is greater than nine. If this occurs, indicate an error by turning on the red light LEDR9.
4. Include the necessary pin assignments for your DE-series board, compile the circuit, and download it into the FPGA chip.
5. Test your circuit by trying different values for numbers X, Y, and cin.

```
module task_3 (
    input  [9:0] SW,
    output [9:0] LEDR,
    output [6:0] HEX0,
    output [6:0] HEX1,
    output [6:0] HEX3,
    output [6:0] HEX5
);

   wire [4:0] s;
   wire [3:0] x = SW[3:0];
   wire [3:0] y = SW[7:4];
   wire [3:0] m, a, z;
   wire c_in = SW[8];
   wire c_out;
   wire err;
   assign z[3:1] = 3'b000;

   fulladder_4bit adder (
       .a(x),
       .b(y),
       .c_in(c_in),
       .s(s),
       .c_out(c_out)
   );

   assign s[4] = c_out;

   t3_circuit_a circuitA (
       .s(s),
       .a(a)
   );

   t3_comparator comp (
       .v(s),
       .z(z[0])
   );

   mux_2to1 #(
       .n(3)
   ) mux (
       .a(s[3:0]),
       .b(a),
       .s(z[0]),
       .m(m)
   );

   decoder digit0 (.v (m), .hex(HEX0));
   decoder digit1 (.v (z), .hex(HEX1));
   decoder digit3 (.v (x), .hex(HEX3));
   decoder digit5 (.v (y), .hex(HEX5));

   invalid_input err_led (
       .x  (x),
       .y  (y),
       .err(err)
   );

   assign LEDR[9] = err;

endmodule

module fulladder_4bit (
    input [3:0] a,
    input [3:0] b,
    input c_in,
    output [3:0] s,
    output c_out
);

   genvar i;
   wire [4:0] c;
   assign c_out = c[4];
   assign c[0]  = c_in;

   generate
      for (i = 0; i < 4; i = i + 1) begin : stage
         fulladder add (
             .c_in(c[i]),
             .a(a[i]),
             .b(b[i]),
             .s(s[i]),
             .c_out(c[i+1])
         );
      end
   endgenerate

endmodule

module t3_comparator (
    input [4:0] v,
    output z
);

   assign z = (v[3] & (v[2] | v[1])) | v[4];

endmodule

module invalid_input (
    input [3:0] x,
    input [3:0] y,
    output err
);

   assign err = (x[3] & (x[2] | x[1])) | (y[3] & (y[2] | y[1]));

endmodule

module t3_circuit_a (
    input  [4:0] s,
    output [3:0] a
);

   assign a[3] = ~s[3] & s[1];
   assign a[2] = (~s[3] & ~s[1]) & (s[2] & s[1]);
   assign a[1] = ~s[1];
   assign a[0] = s[0];

endmodule
```



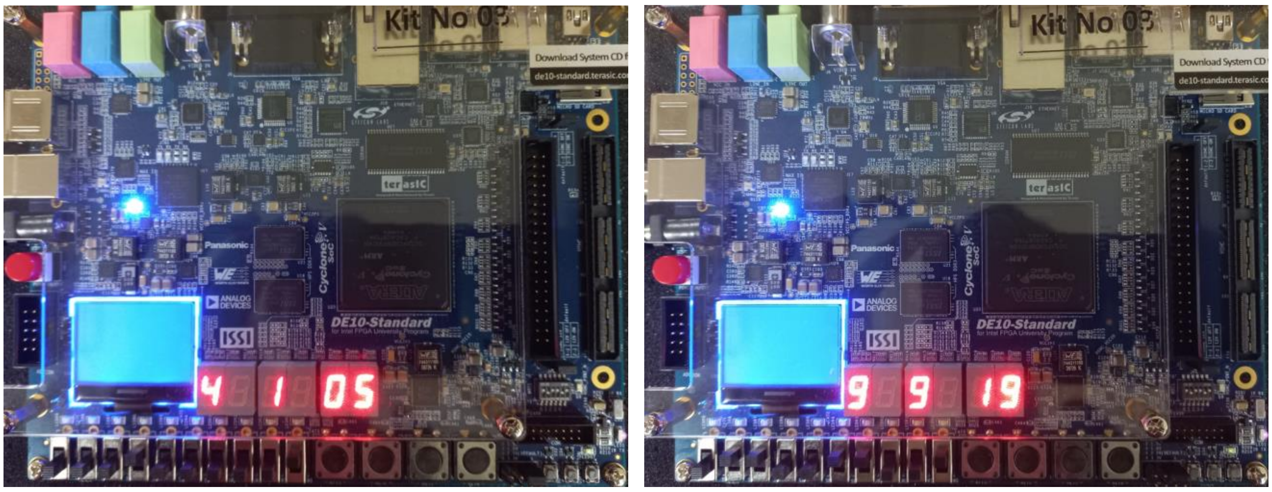
To test the desired functioning of the circuit, we implement the following testbench:

```
module t3_tb ();

   reg  [9:0] SW;
   wire [9:0] LEDR;
   wire [6:0] HEX0, HEX1, HEX3, HEX5;

   task_3 test (
       .SW  (SW),
       .LEDR(LEDR),
       .HEX0(HEX0),
       .HEX1(HEX1),
       .HEX3(HEX3),
       .HEX5(HEX5)
   );

   initial begin
      SW = 10'bX_0_0010_0101;
      #10;
      SW = 10'bX_1_0010_0101;
      #10;
      SW = 10'bX_0_0010_1111;
      #10;
      SW = 10'bX_1_1001_1001;
      #10;
   end

endmodule
```



```
/t3_tb/SW     x000100101   x000100101   x100100101   x000101111   x110011001
/t3_tb/LEDR   0zzzzzzzzz   0zzzzzzzzz                1zzzzzzzzz   0zzzzzzzzz
/t3_tb/HEX5   0100100      0100100                                0010000
/t3_tb/HEX3   0010010      0010010                   1111000      0010000
/t3_tb/HEX1   1000000      1000000                   1111001
/t3_tb/HEX0   1111000      1111000      0000000      0110000      0010000
```

Following are sample hardware demonstrations of Part III on the DE-10 board:



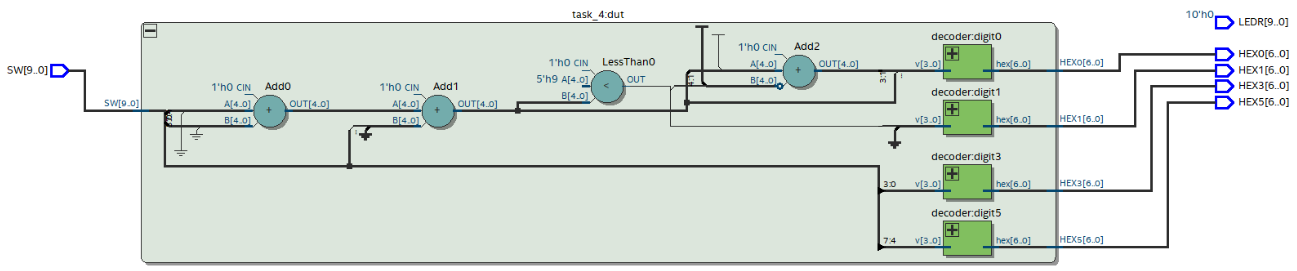
## 3.4 Part IV

In part IV you created Verilog code for a BCD adder. A different approach for describing the adder in Verilog code is to specify an algorithm like the one represented by the following pseudo-code:

```
1   $T_0 = A + B + c_0$
2   if ($T_0 > 9$) then
3       $Z_0 = 10$;
4       $c_1 = 1$;
5   else
6       $Z_0 = 0$;
7       $c_1 = 0$;
8   end if
9   $S_0 = T_0 - Z_0$
10  $S_1 = c_1$
```

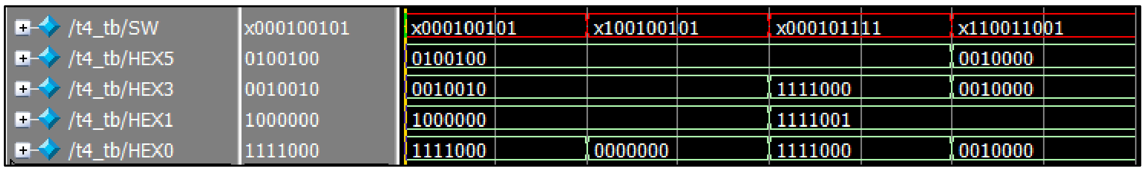
1. Create a new Quartus project for your Verilog code. Use switches SW7−4 and SW3−0 for the inputs A and B, respectively, and use SW8 for the carry-in. The value of A should be displayed on the 7-segment display HEX5, while B should be on HEX3. Display the BCD sum, S1S0, on HEX1 and HEX0.
2. Use the Quartus RTL Viewer tool to examine the circuit produced by compiling your Verilog code. Compare the circuit to the one you designed in Part IV.
3. Download your circuit onto your DE-series board and test it by trying different values for numbers A and B.

```
module task_4 (
    input  [9:0] SW,
    output [6:0] HEX0,
    output [6:0] HEX1,
    output [6:0] HEX3,
    output [6:0] HEX5
);

   wire [4:0] total;
   wire [3:0] x = SW[3:0];
   wire [3:0] y = SW[7:4];
   wire c_in = SW[8];
   reg [3:0] s1, s0, z;

   assign total = x + y + c_in;

   always @(*) begin

      s1[3:1] = 3'b000;

      if (total > 9) begin
         z = 10;
         s1[0] = 1;
      end else begin
         z = 0;
         s1[0] = 0;
      end

      s0 = total - z;

   end

   decoder digit0 (.v (s0), .hex(HEX0));
   decoder digit1 (.v (s1), .hex(HEX1));
   decoder digit3 (.v (x), .hex(HEX3));
   decoder digit5 (.v (y), .hex(HEX5));

endmodule
```



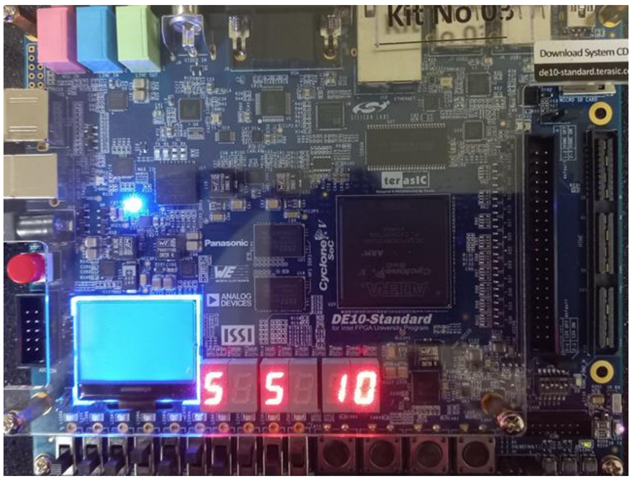
To test the desired functioning of the circuit, we implement the following testbench:

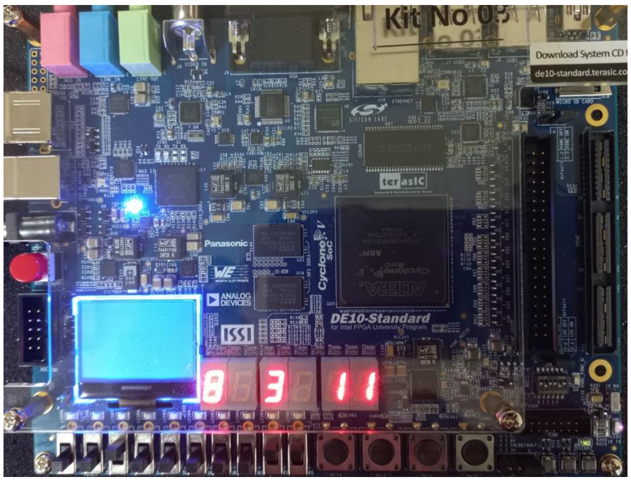
```
module t4_tb ();

   reg [9:0] SW;
   wire [6:0] HEX0, HEX1, HEX3, HEX5;

   task_4 test (
       .SW  (SW),
       .HEX0(HEX0),
        .HEX1(HEX1),
        .HEX3(HEX3),
        .HEX5(HEX5)
    );

    initial begin
      SW = 10'bX_0_0010_0101;
      #10;
      SW = 10'bX_1_0010_0101;
      #10;
      SW = 10'bX_0_0010_1111;
      #10;
      SW = 10'bX_1_1001_1001;
      #10;
    end

endmodule
```



```
/t4_tb/SW    x000100101  x000100101  x100100101  x000101111  x110011001
/t4_tb/HEX5  0100100     0100100                             0010000
/t4_tb/HEX3  0010010     0010010                 1111000     0010000
/t4_tb/HEX1  1000000     1000000                 1111001
/t4_tb/HEX0  1111000     1111000     0000000     1111000     0010000
```

Following are sample hardware demonstrations of Part IV on the DE-10 board:





## 4 Conclusion

In conclusion, this lab exercise has provided us with a practical understanding of combination circuits and their building blocks. We have learned how to design and implement combinational circuits that can perform binary-to-decimal number conversion and binary-coded-decimal (BCD) addition using switches as inputs and LEDs and 7-segment displays as output devices. Overall, this lab exercise has been a valuable learning experience, providing us with hands-on experience with combination circuits and their building blocks. It has also given us a foundation for further exploration and experimentation in the field of digital electronics.