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: 1/03/2023

Semester: 6th

Section: BEE 12C

EE-421: Digital System Design

Lab 4: Latches, Flip-flops, and Registers

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 Latches, Flip-flops, and Registers 3
2.1 Objectives 3
2.2 Introduction 3
2.3 Software 3
3 Lab Procedure 4
3.1 Part I 4
3.2 Part II 5
3.3 Part III 7
3.4 Part IV 8
3.5 Part V 10
4 Conclusion 13

2 Latches, Flip-flops, and Registers

2.1 Objectives

The objective of this laboratory exercise is to investigate the behavior and functionality of latches, flip-flops, and registers. The experiment aims to provide a deeper understanding of the fundamental principles behind these digital circuits and their applications in digital systems.

2.2 Introduction

Digital circuits play a crucial role in modern electronics, powering everything from computers to smartphones to automobiles. These circuits are constructed from basic building blocks, such as latches, flip-flops, and registers, which are essential components for the storage and manipulation of digital data.

Latches are simple storage devices that can hold a single bit of data, while flip-flops are more complex devices that can store a single bit of data and provide a clocked mechanism for input and output. Registers, on the other hand, are arrays of flip-flops that can store larger amounts of data. These digital circuits are widely used in a variety of applications, including memory, counters, and state machines. By understanding how these components work and how they can be combined, we can design more complex digital systems that can perform sophisticated operations.

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

1. Create a Quartus project for the RS latch circuit as follows:
2. Create a new Quartus project for your DE-series board.
3. Generate a Verilog file for the RS latch. Use the code in either Figure 2 or Figure 3 (both versions of the 2 codes should produce the same circuit) and include it in the project.

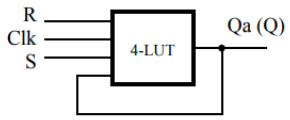
```
// A gated RS latch
module part1 (Clk, R, S, Q);
    input Clk, R, S;
    output Q;

    wire R_g, S_g, Qa, Qb /* synthesis keep */ ;

    and (R_g, R, Clk);
    and (S_g, S, Clk);
    nor (Qa, R_g, Qb);
    nor (Qb, S_g, Qa);

    assign Q = Qa;

endmodule
```

Figure 2: Specifying an RS latch by instantiating logic gates.

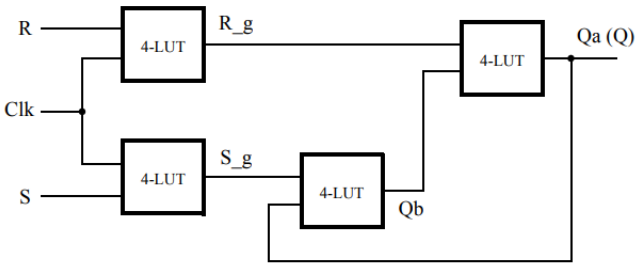
```
// A gated RS latch
module part1 (Clk, R, S, Q);
    input Clk, R, S;
    output Q;

    wire R_g, S_g, Qa, Qb /* synthesis keep */ ;

    assign R_g = R & Clk;
    assign S_g = S & Clk;
    assign Qa = ~(R_g | Qb);
    assign Qb = ~(S_g | Qa);

    assign Q = Qa;

endmodule
```

Figure 3: Specifying an RS latch by using Boolean expressions.

4. Compile the code. Use the Quartus RTL Viewer tool to examine the gate-level circuit produced from the code and use the Technology Map Viewer tool to verify that the latch is implemented as shown in Figure 4b.
5. Simulate the behavior of your Verilog code by using the simulation feature provided in the Quartus software.



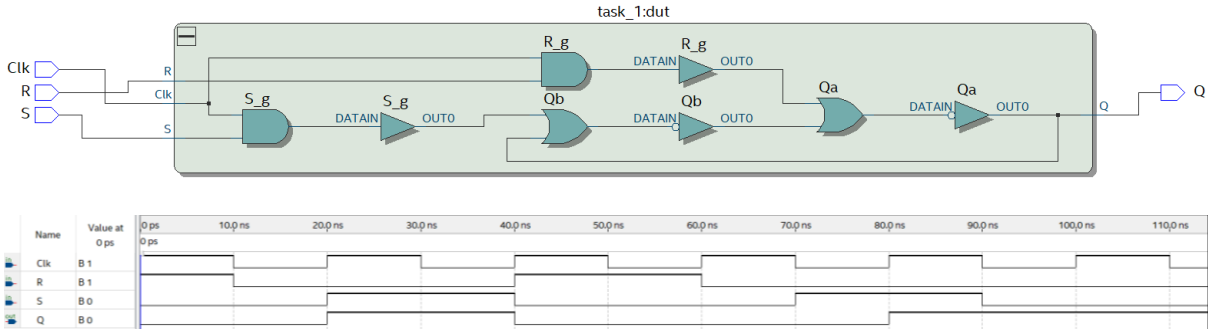
(a) Using one 4-input lookup table for the RS latch.



(b) Using four 4-input lookup tables for the RS latch.

Figure 4: Implementation of the RS latch from Figure 1.

```
module task_1 (
    input  clk,
    input  R,
    input  S,
    output Q
);

   wire R_g, S_g, Qa, Qb  /* synthesis keep */;

   assign R_g = R & clk;
   assign S_g = S & clk;
   assign Qa  = ~(R_g | Qb);
   assign Qb  = ~(S_g | Qa);
   assign Q   = Qa;

endmodule
```



3.2 Part II

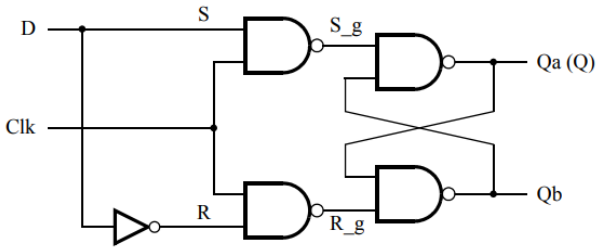


Figure 6: Circuit for a gated D latch.

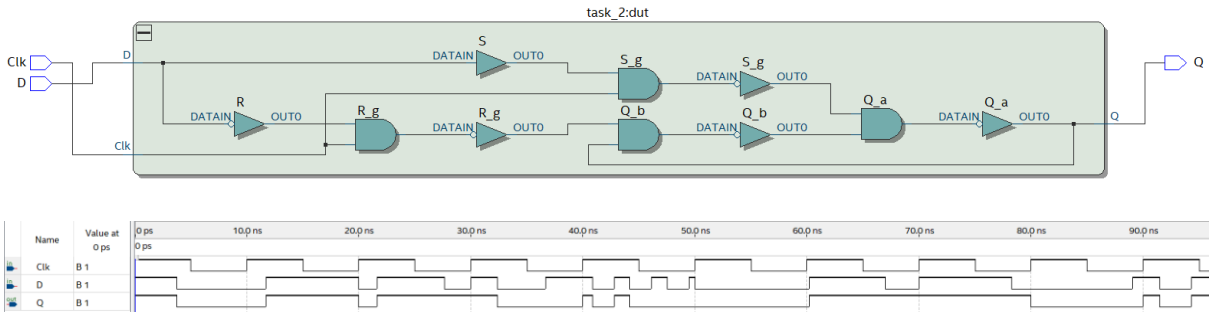
Perform the following steps:

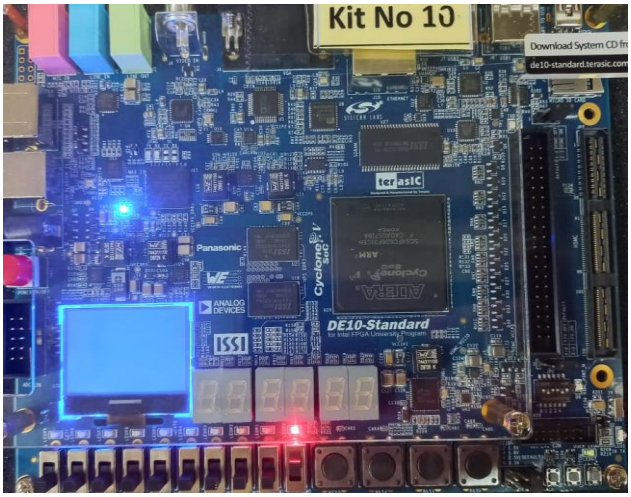
1. Create a new Quartus project. Generate a Verilog file using the style of code in Figure 3 for the gated D latch. Use the /* synthesis keep */ directive to ensure that separate logic elements are used to implement the signals R, S_g, R_g, Qa, and Qb.
2. Compile your project and then use the Technology Map Viewer tool to examine the implemented circuit.
3. Verify that the latch works properly for all input conditions by using functional simulation. Examine the timing characteristics of the circuit by using timing simulation.
4. Create a new Quartus project which will be used for implementation of the gated D latch on your DE-series board. This project should consist of a top-level module that contains the appropriate input and output ports (pins) for your board. Instantiate your latch in this top-level module. Use

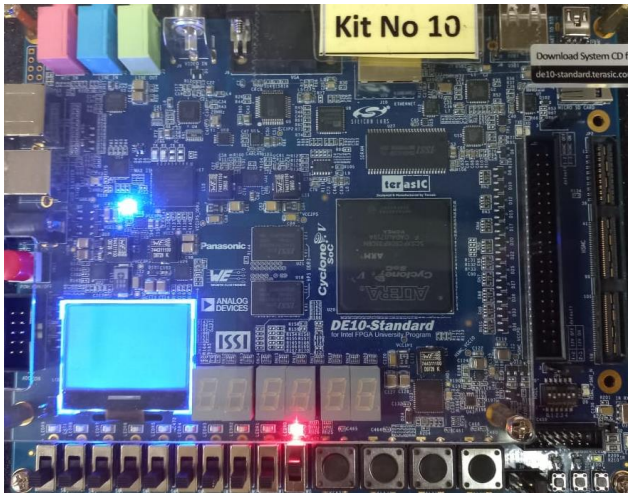
switch SW0 to drive the D input of the latch and use SW1 as the Clk input. Connect the Q output to LEDR0.

5. Include the required pin assignments and then compile your project and download the compiled circuit onto your DE-series board. 6. Test the functionality of your circuit by toggling the D and Clk switches and observing the Q output.

```
module task_2 (
    input  Clk,
    input  D,
    output Q
);

   wire R, S, R_g, S_g, Q_a, Q_b  /* synthesis keep */;

   assign R   = ~D;
   assign S   = D;
   assign R_g = ~(R & Clk);
   assign S_g = ~(S & Clk);
   assign Q_a = ~(S_g & Q_b);
   assign Q_b = ~(R_g & Q_a);
   assign Q   = Q_a;

endmodule
```





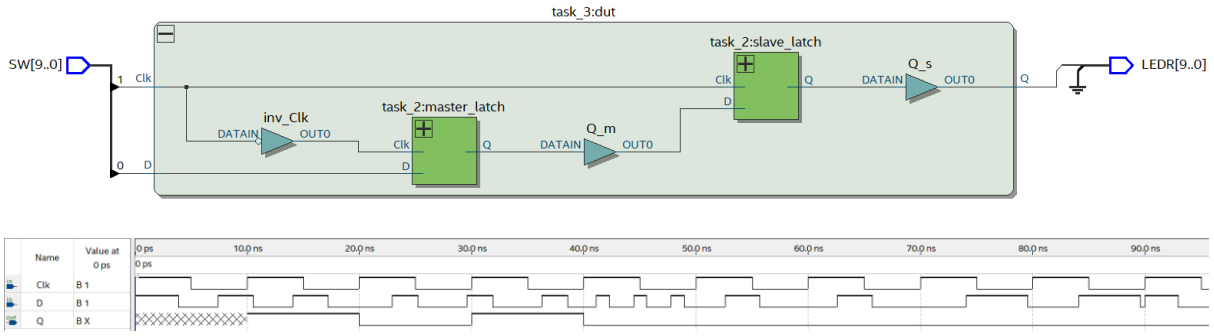


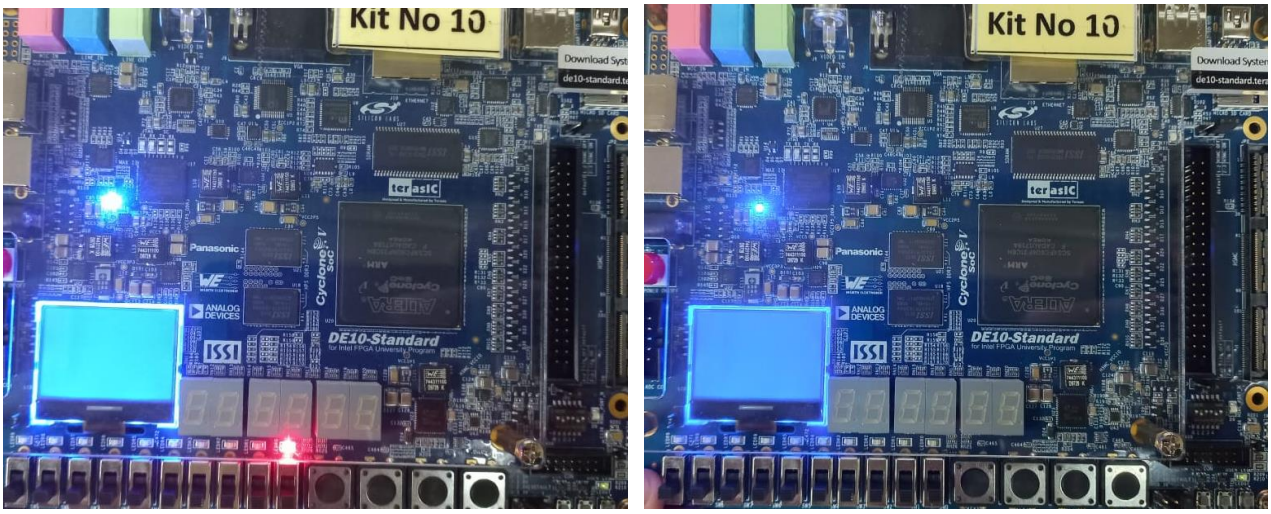
3.3 Part III

Perform the following:

1. Create a new Quartus project. Generate a Verilog file that instantiates two copies of your gated D latch module from Part II to implement the master-slave flip-flop.
2. Include in your project the appropriate input and output ports for your DE-series board. Use switch SW0 to drive the D input of the flip-flop and use SW1 as the Clock input. Connect the Q output to LEDR0.
3. Include the required pin assignments and then compile your project.
4. Use the Technology Map Viewer to examine the D flip-flop circuit and use simulation to verify its correct operation.
5. Download the circuit onto your DE-series board and test its functionality by toggling the D and Clock switches and observing the Q output.

```
module task_3 (
    input  clk,
    input  D,
    output Q
);

   wire Q_m, Q_s, inv_clk  /* synthesis keep */;
   assign inv_clk = ~clk;

   task_2 master_latch (
       .clk(inv_clk),
       .D  (D),
       .Q  (Q_m)
   );

   task_2 slave_latch (
       .clk(clk),
       .D  (Q_m),
       .Q  (Q_s)
   );

   assign Q = Q_s;

endmodule
```





3.4 Part IV

Implement and simulate this circuit using the Quartus software as follows:

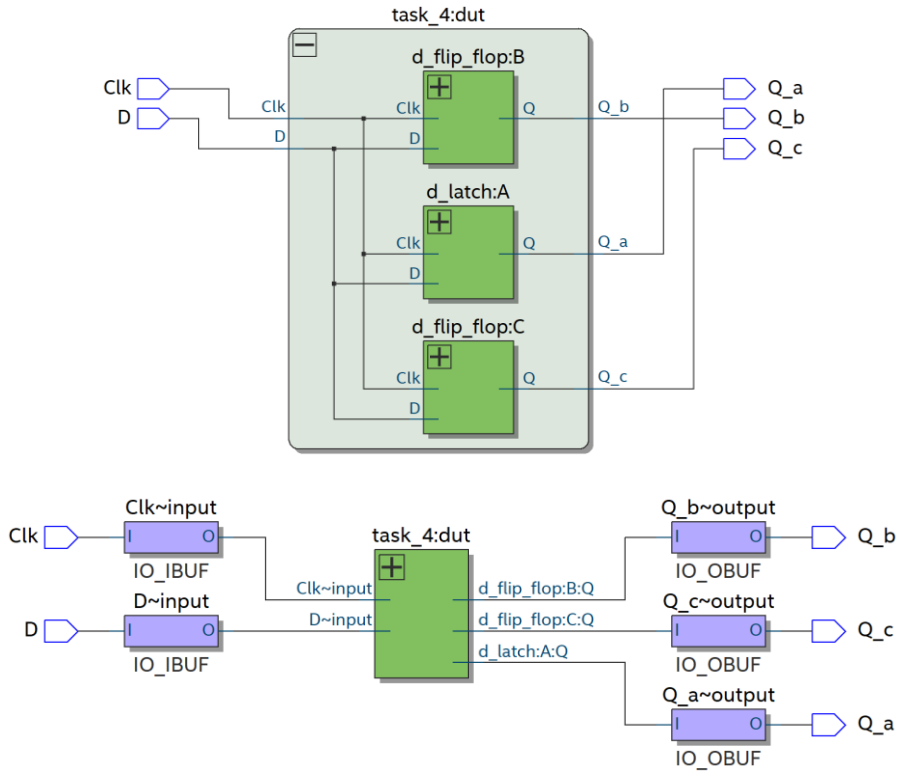
1. Create a new Quartus project.
2. Write a Verilog file that instantiates the three storage elements. For this part you should no longer use the /* synthesis keep */ directive from Parts I to III. Figure 9 gives a behavioral style of Verilog code that specifies the gated D latch in Figure 6.
3. Compile your code and use the Technology Map Viewer to examine the implemented circuit. Verify that the latch uses one lookup table and that the flip-flops are implemented using the flip-flops provided in the target FPGA.
4. Create a Vector Waveform File (.vwf) that specifies the inputs and outputs of the circuit. Draw the inputs D and Clock as indicated in Figure 8. Use functional simulation to obtain the three output signals. Observe the different behavior of the three storage elements.

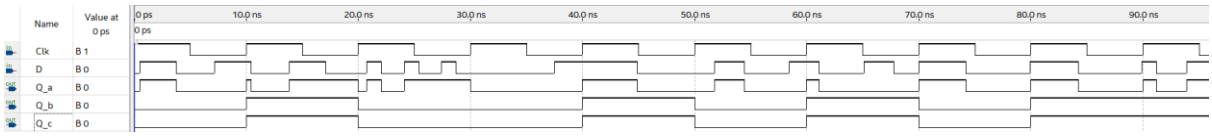
```
module D_latch  (D, Clk, Q);
   input D, Clk;
   output reg Q;

   always @ (D, Clk)
      if (Clk)
         Q = D;
endmodule
```

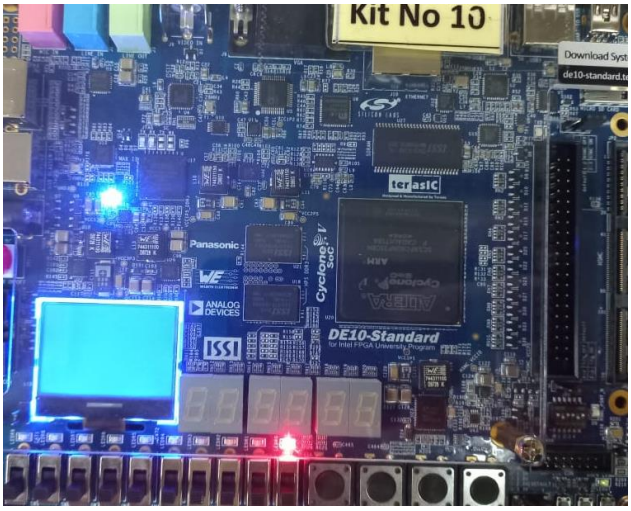
Figure 9: A behavioral style of Verilog code that specifies a gated D latch.

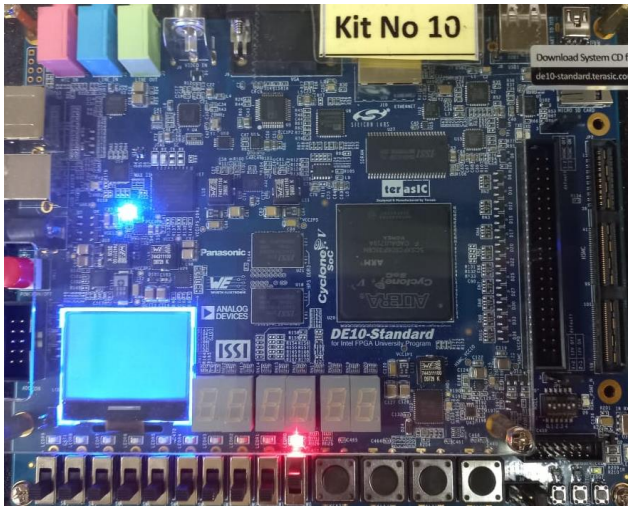
```
module task_4 (
    input  clk,
    input  D,
    output Q_a,
    Q_b,
    Q_c
);

   d_latch A (
       .D  (D),
```

```
        .clk(clk),
        .Q  (Q_a)
    );
    d_flip_flop B (
        .D  (D),
        .clk(clk),
        .Q  (Q_b)
    );
    d_flip_flop C (
        .D  (D),
        .clk(~clk),
        .Q  (Q_c)
    );

endmodule

module d_latch (
    input D,
    input clk,
    output reg Q
);

    always @(D, clk) if (clk) Q = D;

endmodule

module d_flip_flop (
    input D,
    input clk,
    output reg Q
);

    always @(posedge clk) Q <= D;

endmodule
```









3.5 Part V

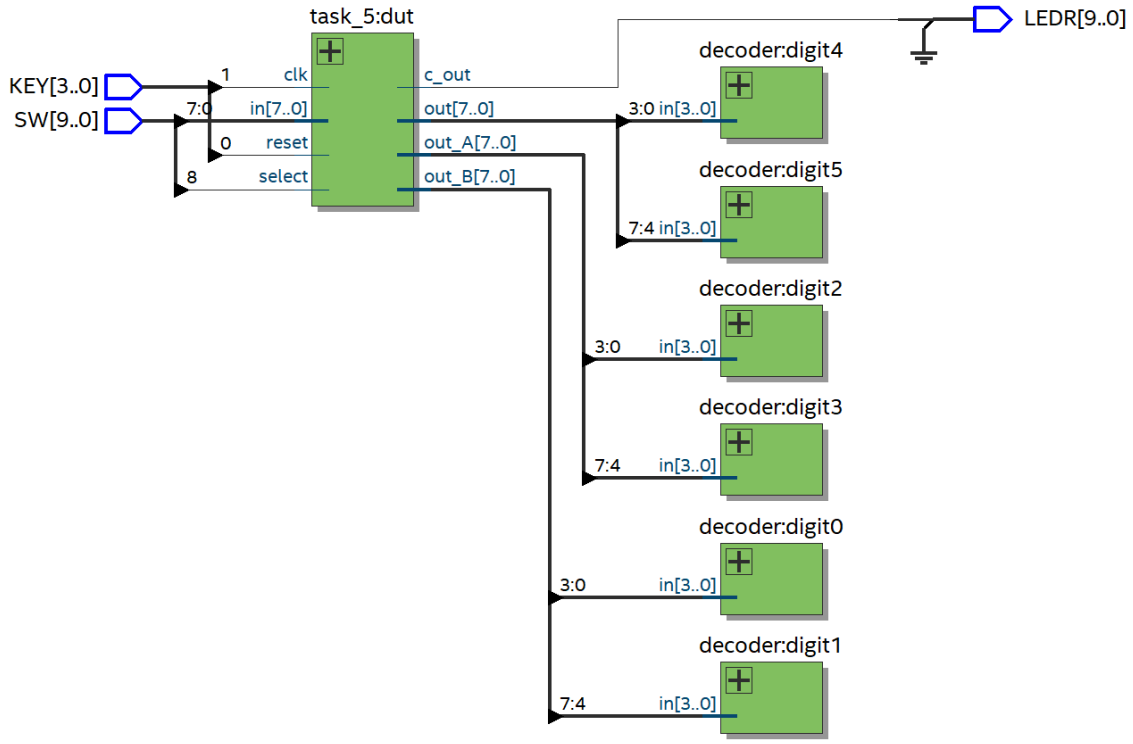
We wish to display the hexadecimal value of an 8-bit number A on the two 7-segment displays HEX3 − 2. We also wish to display the hex value of an 8-bit number B on the two 7-segment displays HEX1 − 0. The values of A and B are inputs to the circuit which are provided by means of switches SW7−0. To input the values of A and B, first set the switches to the desired value of A, store these switch values in a register, and then change the switches to the desired value of B. Finally, use an adder to generate the arithmetic sum S = A + B, and display this sum on the 7-segment displays HEX5 − 4. Show the carry-out produced by the adder on LEDR[0].

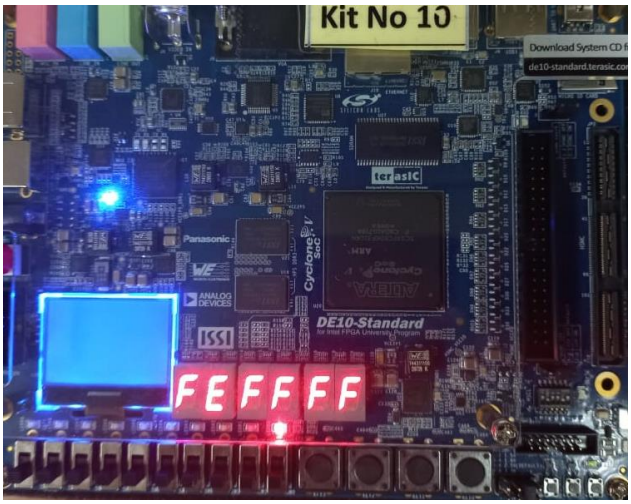
1. Create a new Quartus project which will be used to implement the desired circuit on your DE-series board.
2. Write a Verilog file that provides the necessary functionality. Use KEY0 as an active-low asynchronous reset and use KEY1 as a clock input.
3. Include the necessary pin assignments for the pushbutton switches and 7-segment displays, and then compile the circuit.
4. Download the circuit onto your DE-series board and test its functionality by toggling the switches and observing the output displays.

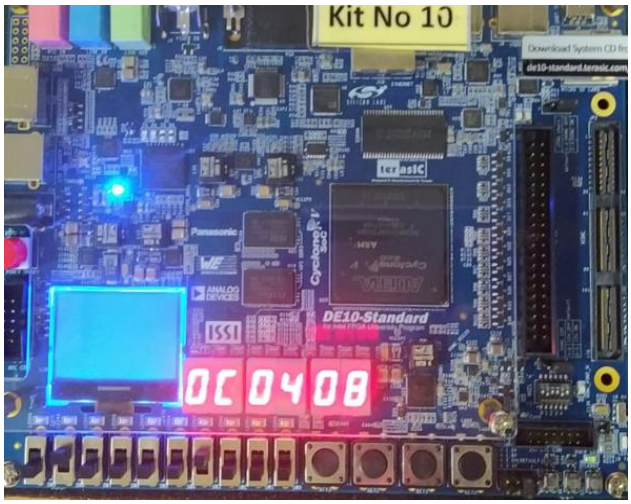
```
module task_5 (
    input clk,
    input select,
    input reset,
    input [7:0] in,
    output [7:0] out_A,
    output [7:0] out_B,
    output [7:0] out,
    output c_out
);
```

```
    wire [7:0] in_A, in_B;

    register_8b regA (
        .in(in),
        .en(~select),
        .clk(clk),
        .reset(reset),
        .out(in_A)
    );

    register_8b regB (
        .in(in),
        .en(select),
        .clk(clk),
        .reset(reset),
        .out(in_B)
    );

    fulladder_8b adder (
        .a(in_A),
        .b(in_B),
        .s(out),
        .c_out(c_out)
    );

    assign out_A = in_A;
    assign out_B = in_B;

endmodule


module fulladder_8b (
    input [7:0] a,
    input [7:0] b,
    output reg [7:0] s,
    output reg c_out
);

    always @(*) begin
       {c_out, s} = a + b;
    end

endmodule

module register_8b (
    input [7:0] in,
    input en,
    input clk,
    input reset,
    output reg [7:0] out
);

    always @(posedge clk, negedge reset) begin
       if (reset == 0) begin
          out <= 0;
       end else if (en == 1) begin
          out <= in;
       end else begin
          out <= out;
       end
```

```
   end

endmodule

module decoder (
    input [3:0] in,
    output reg [7:0] HEX
);

   always @(*) begin
      case (in)
         4'b0000: HEX <= 8'b1000000;
         4'b0001: HEX <= 8'b1111001;
         4'b0010: HEX <= 8'b0100100;
         4'b0011: HEX <= 8'b0110000;
         4'b0100: HEX <= 8'b0011001;
         4'b0101: HEX <= 8'b0010010;
         4'b0110: HEX <= 8'b0000010;
         4'b0111: HEX <= 8'b1111000;
         4'b1000: HEX <= 8'b0000000;
         4'b1001: HEX <= 8'b0010000;
         4'b1010: HEX <= 8'b0100000;
         4'b1011: HEX <= 8'b0000011;
         4'b1100: HEX <= 8'b1000110;
         4'b1101: HEX <= 8'b0100001;
         4'b1110: HEX <= 8'b0000110;
         4'b1111: HEX <= 8'b0001110;
      endcase
   end

endmodule
```







4 Conclusion

Through this lab exercise, we have gained a deeper understanding of latches, flip-flops, and registers and their behavior on the DE10 FPGA board. We have observed how these basic building blocks can be combined to create more complex digital circuits that can perform sophisticated operations. We have also explored the practical applications of these circuits in digital design, including memory, counters, and state machines. By understanding the principles behind these components and their applications, we can design more efficient and effective digital systems.