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: 8/03/2023

Semester: 6th Section: BEE 12C

EE-421: Digital System Design

Lab 6: Timers and Real-Time Clock

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 Timers and Real-Time Clock

2.1 Objectives

The purpose of this exercise is to build and use counters. The designed circuits are to be implemented on an Intel FPGA DE10-Lite, DE0-CV, DE1-SoC, or DE2-115 Board.

Students are expected to have a basic understanding of counters and sufficient familiarity with the Verilog hardware description language to implement various types of latches and flip-flops.

2.2 Introduction

Counters are fundamental building blocks of digital systems that are used to count clock cycles, pulses, or events. In this lab report, the objective is to design and implement different types of counters on an Intel FPGA board. The lab exercise requires a basic understanding of counters and a sufficient familiarity with the Verilog hardware description language to implement various types of latches and flip-flops. The objective of this exercise is to provide students with hands-on experience with designing and implementing counters on FPGA boards.

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.

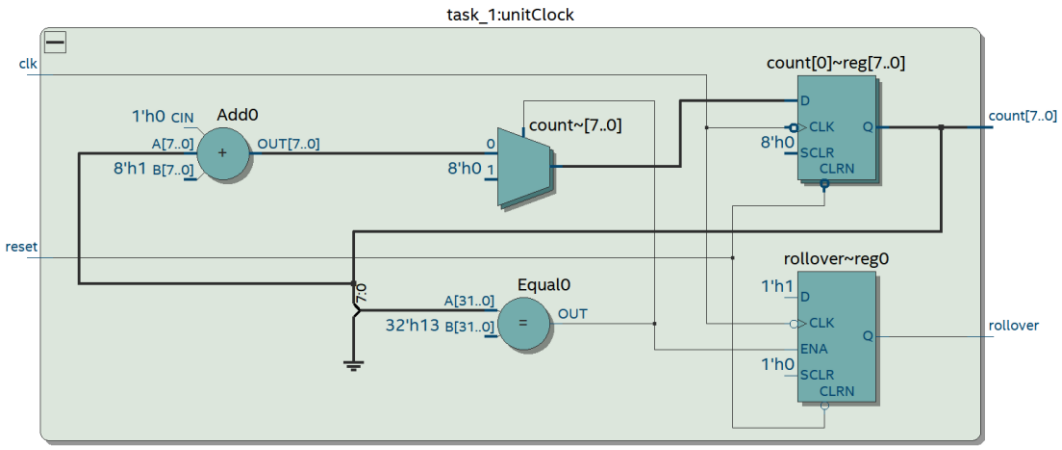
3 Lab Procedure

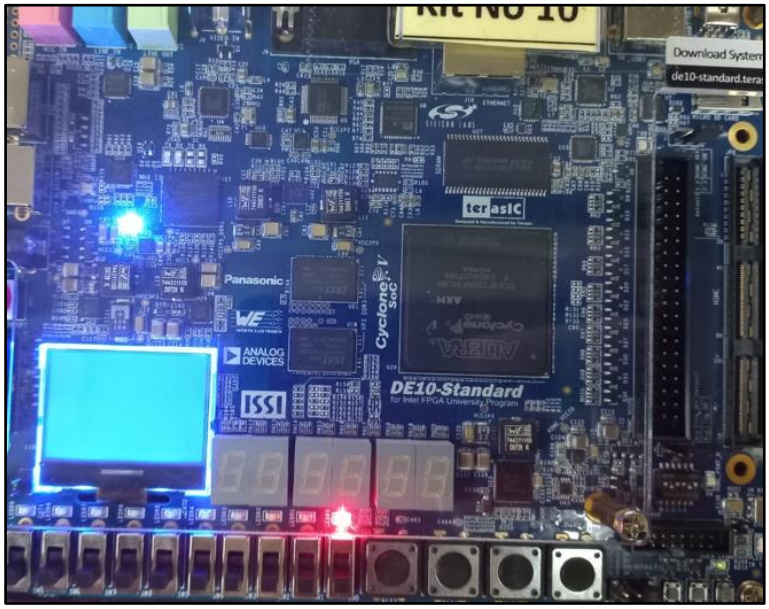
3.1 Part I

Create a modulo-k counter by modifying the design of an 8-bit counter to contain an additional parameter. The counter should count from 0 to k−1. When the counter reaches the value k−1, then the next counter value should be 0. Include an output from the counter called rollover and set this output to 1 in the clock cycle where the count value is equal to k − 1.
Perform the following steps:

1. Create a new Quartus project which will be used to implement the desired circuit on your DE-series board 1.
2. Write a Verilog file that specifies the circuit for k = 20, and an appropriate value of n. Your circuit should use pushbutton KEY0 as an asynchronous reset and KEY1 as a manual clock input. The contents of the counter should be displayed on the red lights LEDR. Also display the rollover signal on one of the LEDR lights.

```
module task_1 #(
    parameter n = 8,
    k = 20
) (
    input clk, input reset,
    output reg [n-1:0] count, output reg rollover
);
    always @(negedge clk, negedge reset) begin
        if (reset == 0) begin
            count <= 0;
            rollover <= 0;
        end else if (count == k - 1) begin
            count <= 0;
            rollover <= 1;
        end else count <= count + 1;
    end

endmodule
```



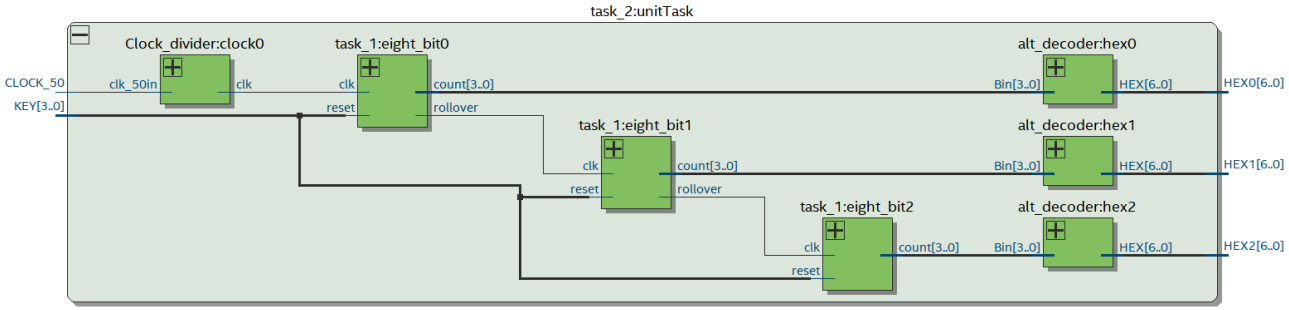


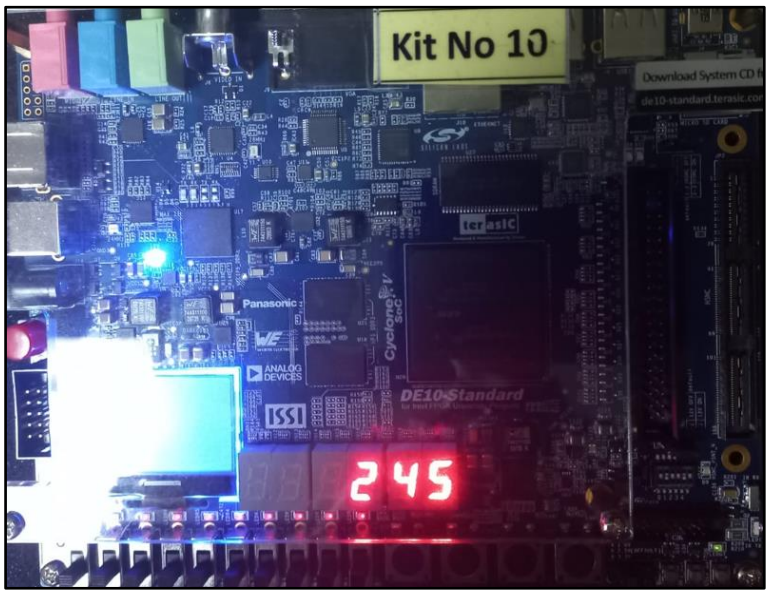
3.2 Part II

Using your modulo-counter from Part I as a subcircuit, implement a 3-digit BCD counter (hint: use multiple counters, not just one). Display the contents of the counter on the 7-segment displays, HEX2−0. Connect all of the counters in your circuit to the 50-MHz clock signal on your DE-series board and make the BCD counter increment at one-second intervals. Use the pushbutton switch KEY0 to reset the BCD counter to 0.

```
module task_2 (
    input CLOCK_50,
    input [3:0] KEY,
    output [6:0] HEX0,
    output [6:0] HEX1,
    output [6:0] HEX2
);

    wire clk, Reset;
    wire [3:0] Count0;
    wire [3:0] Count1;
    wire [3:0] Count2;
    wire rollover0;
    wire rollover1;
    wire rollover2;
    assign Reset = KEY[0];

    Clock_divider clock0 (
        .clk_50in(CLOCK_50),
        .clk(clk)
    );

    defparam clock0.c = 25000000;
    task_1 eight_bit0 (
        .clk(clk),
        .reset(Reset),
        .count(Count0),
```

```
        .rollover(rollover0)
    );

    defparam eight_bit0.n = 4; defparam eight_bit0.k = 10;
    task_1 eight_bit1 (
        .clk(rollover0),
        .reset(Reset),
        .count(Count1),
        .rollover(rollover1)
    );

    defparam eight_bit1.n = 4; defparam eight_bit1.k = 10;
    task_1 eight_bit2 (
        .clk(rollover1),
        .reset(Reset),
        .count(Count2),
        .rollover(rollover2)
    );

    defparam eight_bit2.n = 4; defparam eight_bit2.k = 10;
    alt_decoder hex0 (
        .Bin(Count0),
        .HEX(HEX0)
    );

    alt_decoder hex1 (
        .Bin(Count1),
        .HEX(HEX1)
    );

    alt_decoder hex2 (
        .Bin(Count2),
        .HEX(HEX2)
    );

endmodule

module Clock_divider(
    input clk_50in,
    output reg clk
);

    parameter [31:0] c;
    reg [31:0] count_reg = 32'd00000000;

    always @(posedge clk_50in) begin
    count_reg <= count_reg + 1;

    if (count_reg == c - 1'd1) begin
        count_reg <= 32'd00000000;
        clk <= ~clk;
    end

end
endmodule
```





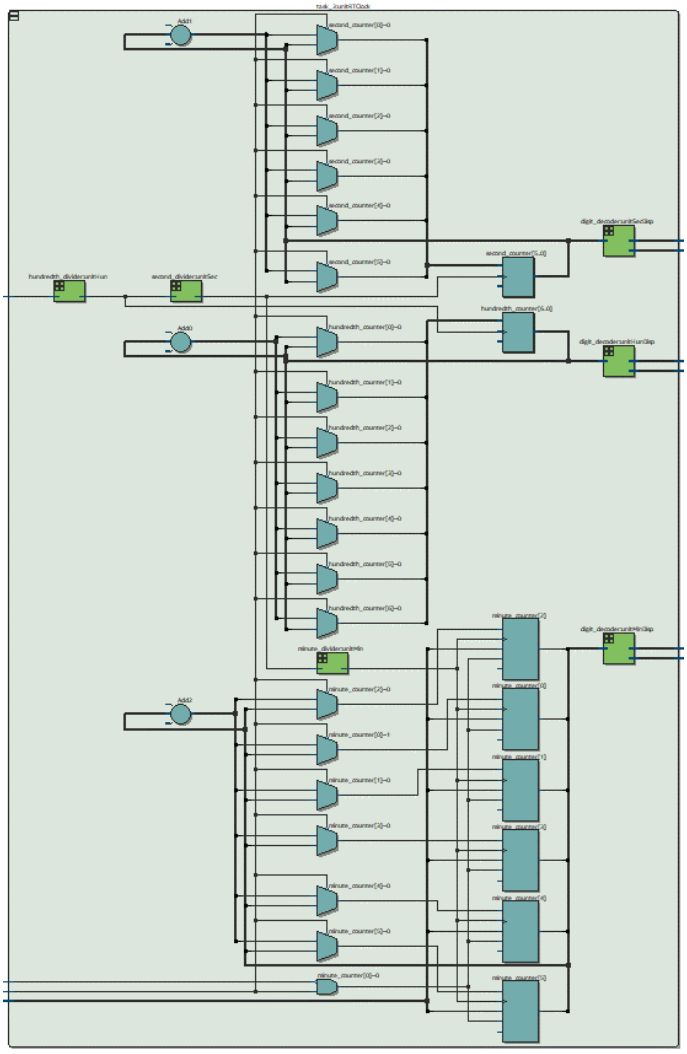
3.3 Part III

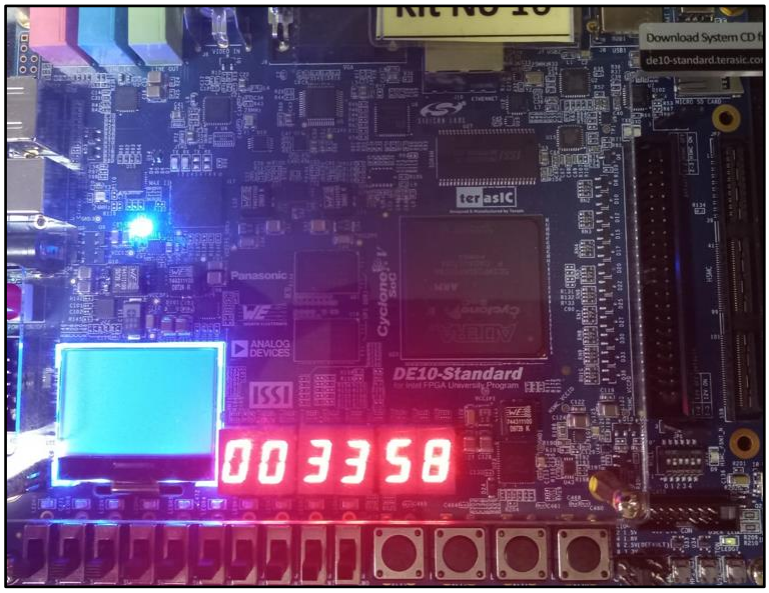
Design and implement a circuit on your DE-series board that acts as a real-time clock. It should display the minutes (from 0 to 59) on HEX5 − 4, the seconds (from 0 to 59) on HEX3 − 2, and hundredths of a second (from 0 to 99) on HEX1 − 0. Use the switches SW7−0 to preset the minute part of the time displayed by the clock when KEY1 is pressed. Stop the clock whenever KEY0 is being pressed and continue the clock when KEY0 is released.

```
module task_3 (
    input CLOCK_50,
    input preset,
    input stop,
    input [7:0] preset_value,
    output [6:0] HEX0,
    output [6:0] HEX1,
    output [6:0] HEX2,
    output [6:0] HEX3,
    output [6:0] HEX4,
    output [6:0] HEX5
);

    reg [6:0] hundredth_counter;
    reg [5:0] second_counter, minute_counter;
```

```
    wire hun_clock, sec_clock, min_clock;

    minute_divider unitMin (
        .clk_in (sec_clock),
        .clk_out(min_clock)
    );
    second_divider unitSec (
        .clk_in (hun_clock),
        .clk_out(sec_clock)
    );
    hundredth_divider unitHun (
        .clk_in (CLOCK_50),
        .clk_out(hun_clock)
    );

    always @(posedge hun_clock, negedge stop) begin
        if (stop == 0) hundredth_counter <= hundredth_counter;
        else if (stop == 100 - 1) hundredth_counter <= 0;
        else hundredth_counter <= hundredth_counter + 1;
    end

    always @(posedge sec_clock, negedge stop) begin
        if (stop == 0) second_counter <= second_counter;
        else if (stop == 60 - 1) second_counter <= 0;
        else second_counter <= second_counter + 1;
    end

    always @(posedge min_clock, negedge stop, negedge preset) begin
        if (stop == 0) minute_counter <= minute_counter;
        else if (preset == 0) minute_counter <= preset_value;
        else if (stop == 60 - 1) minute_counter <= 0;
        else minute_counter <= minute_counter + 1;
    end

    digit_decoder unitHunDisp (
        .in  (hundredth_counter),
        .HEX0(HEX0),
        .HEX1(HEX1)
    );
    digit_decoder unitSecDisp (
        .in  (second_counter),
        .HEX0(HEX2),
        .HEX1(HEX3)
    );
    digit_decoder unitMinDisp (
        .in  (minute_counter),
        .HEX0(HEX4),
        .HEX1(HEX5)
    );

endmodule


module minute_divider (
    input clk_in,  // Second clock
    output reg clk_out
);
```

```
    reg [5:0] count = 0;  // Initialize count to zero

    always @(posedge clk_in) begin
        count <= count + 1;

        if (count == 6'd59) begin
            count   <= 0;
            clk_out <= ~clk_out;  // Invert the output clock
        end
    end

endmodule

module second_divider (
    input clk_in,  // Hundredth clock
    output reg clk_out
);

    reg [6:0] count = 0;  // Initialize count to zero

    always @(posedge clk_in) begin
        count <= count + 1;

        if (count == 7'd49) begin
            count   <= 0;
            clk_out <= ~clk_out;  // Invert the output clock
        end
    end

endmodule

module hundredth_divider (
    input clk_in,  // 50 MHz input clock
    output reg clk_out
);

    reg [22:0] count = 0;  // Initialize count to zero

    always @(posedge clk_in) begin
        count <= count + 1;

        if (count == 23'd249999) begin
            count   <= 0;
            clk_out <= ~clk_out;  // Invert the output clock
        end
    end

endmodule
```




Download System CD
terasIC
DE10-Standard
00 33 58

3.4 Part IV

An early method of telegraph communication was based on Morse code. This code uses patterns of short and long pulses to represent a message. Each letter is represented as a sequence of dots (a short pulse), and dashes (a long pulse). For example, the first eight letters of the alphabet have the following representation:

A	• —
B	— • • •
C	— • — •
D	— • •
E	•
F	• • — •
G	— — •
H	• • • •

Design and implement a circuit that takes as input one of the first eight letters of the alphabet and displays the Morse code for it on a red LED. Your circuit should use switches SW2−0 and pushbuttons KEY1−0 as inputs. When a user presses KEY1, the circuit should display the Morse code for a letter specified by SW2−0 (000 for A, 001 for B, etc.), using 0.5-second pulses to represent dots, and 1.5-second pulses to represent dashes. Pushbutton KEY0 should function as an asynchronous reset. A high-level schematic diagram of the circuit is shown in Figure 2.

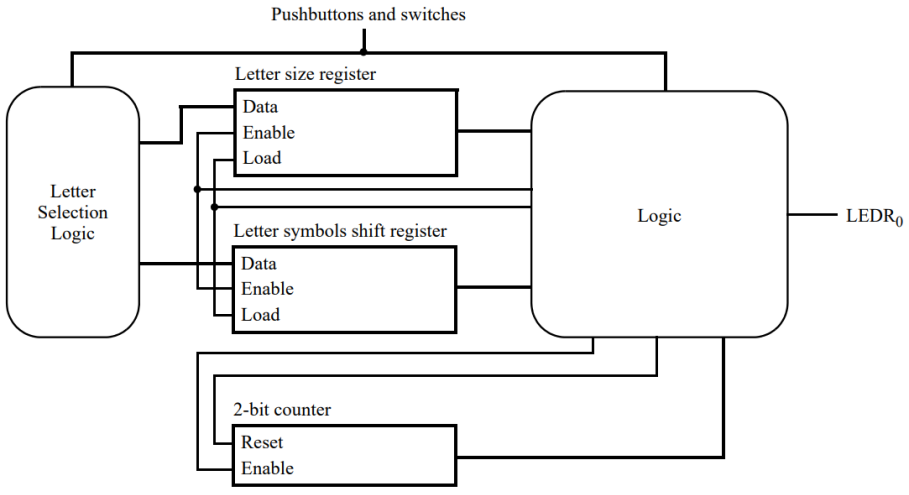
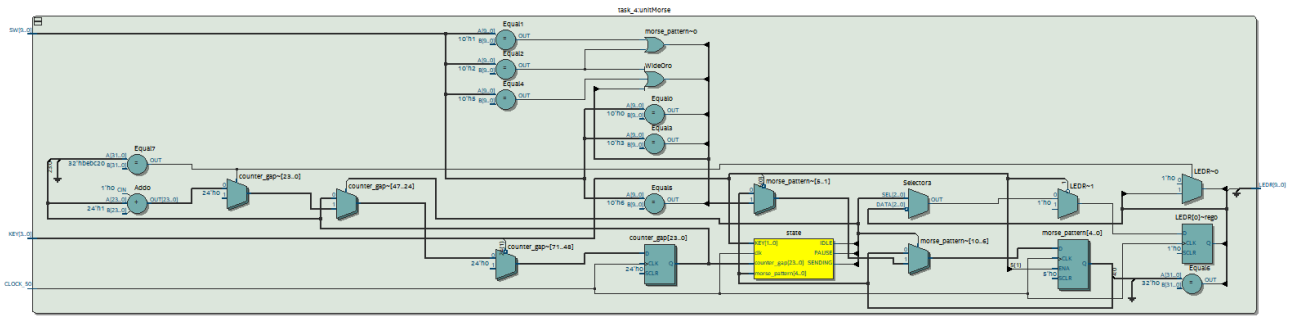


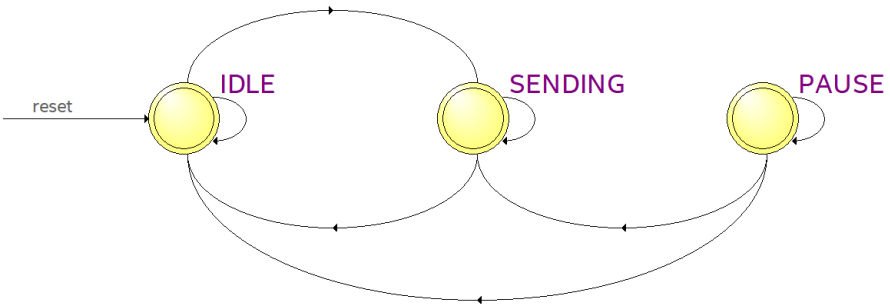
Figure 2: High-level schematic diagram of the circuit for part IV.

```
module task_4 (
    input CLOCK_50,
    input [9:0] SW,
    input [3:0] KEY,
    output reg [9:0] LEDR
);

    // Define the Morse code patterns for each letter from A to H.
    parameter [4:0] A = 5'b00001;
```

```
    parameter [4:0] B = 5'b10000;
    parameter [4:0] C = 5'b10100;
    parameter [4:0] D = 5'b1000;
    parameter [4:0] E = 5'b0;
    parameter [4:0] F = 5'b00100;
    parameter [4:0] G = 5'b110;
    parameter [4:0] H = 5'b00000;

    // Define the counter parameters for the dot, dash, and gap durations.
    parameter CNT_05S = 25000000;  // 0.5 second at 50 MHz
    parameter CNT_15S = 75000000;  // 1.5 seconds at 50 MHz
    parameter CNT_05S_GAP = 12500000;  // 0.5 second gap at 50 MHz

    // Define the state variables for the state machine.
    parameter IDLE = 2'b00;
    parameter SENDING = 2'b01;
    parameter PAUSE = 2'b10;
    reg [ 1:0] state = IDLE;

    // Define the variables for the counters and the Morse code pattern.
    reg [23:0] counter_dot = 0;
    reg [23:0] counter_dash = 0;
    reg [23:0] counter_gap = 0;
    reg [ 4:0] morse_pattern = 0;

    always @(posedge CLOCK_50) begin
        // Check for reset button and reset the state machine and counters.
        if (KEY[1] == 1'b0) begin
            state <= IDLE;
            counter_dot <= 0;
            counter_dash <= 0;
            counter_gap <= 0;
            LEDR[0] <= 1'b0;
        end else begin
            case (state)
                IDLE: begin
                    if (KEY[0] == 1'b0) begin
                        case (SW)
                            3'b000:  morse_pattern <= A;
                            3'b001:  morse_pattern <= B;
                            3'b010:  morse_pattern <= C;
                            3'b011:  morse_pattern <= D;
                            3'b100:  morse_pattern <= E;
                            3'b101:  morse_pattern <= F;
                            3'b110:  morse_pattern <= G;
                            3'b111:  morse_pattern <= H;
                            default: morse_pattern <= 5'b0;
                        endcase
                        state <= SENDING;
                    end
                end
                SENDING: begin
                    if (morse_pattern == 0) begin
                        state   <= IDLE;
                        LEDR[0] <= 1'b0;
                    end else begin
                        case (morse_pattern[0])
```

```
                        1'b0: begin  // dot
                            if (counter_dot == CNT_05S) begin
                                counter_dot <= 0;
                                LEDR[0] <= 1'b0;
                                counter_gap <= counter_gap + 1;
                                if (counter_gap == CNT_05S_GAP) begin
                                    counter_gap   <= 0;
                                    morse_pattern <= morse_pattern >> 1;
                                end
                            end else begin
                                counter_dot <= counter_dot + 1;
                                LEDR[0] <= 1'b1;
                            end
                        end
                        1'b1: begin  // dash
                            if (counter_dash == CNT_15S) begin
                                counter_dash <= 0;
                                LEDR[0] <= 1'b0;
                                counter_gap <= counter_gap + 1;
                                if (counter_gap == CNT_05S_GAP) begin
                                    counter_gap   <= 0;
                                    morse_pattern <= morse_pattern >> 1;
                                end
                            end else begin
                                counter_dash <= counter_dash + 1;
                                LEDR[0] <= 1'b1;
                            end
                        end
                    endcase
                end
            end
            PAUSE: begin
                // Pause for the gap duration.
                if (counter_gap == CNT_05S_GAP) begin
                    counter_gap <= 0;
                    state <= SENDING;
                end else begin
                    counter_gap <= counter_gap + 1;
                    LEDR[0] <= 1'b0;
                end
            end
        endcase
    end
end

endmodule
```





4 Conclusion

In conclusion, this lab exercise was designed to provide students with practical experience in building and using counters. The implementation of different types of latches and flip-flops using the Verilog hardware description language on Intel FPGA boards has helped students gain a better understanding of digital systems. By implementing these circuits on the FPGA boards, students have been able to observe the behavior of the counters in real-time and gain insight into the functionality of digital systems. Overall, this exercise has provided students with a practical understanding of counters and their applications in digital systems.