**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 |  |  |  |  |  |

**Table of Contents**

[2 Timers and Real-Time Clock 3](#_Toc131542346)

[2.1 Objectives 3](#_Toc131542347)

[2.2 Introduction 3](#_Toc131542348)

[2.3 Software 3](#_Toc131542349)

[3 Lab Procedure 4](#_Toc131542350)

[3.1 Part I 4](#_Toc131542351)

[3.2 Part II 5](#_Toc131542352)

[3.3 Part III 7](#_Toc131542353)

[3.4 Part IV 11](#_Toc131542354)

[4 Conclusion 14](#_Toc131542355)

# Timers and Real-Time Clock

## 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.

## 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.

## 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.

# Lab Procedure

## 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

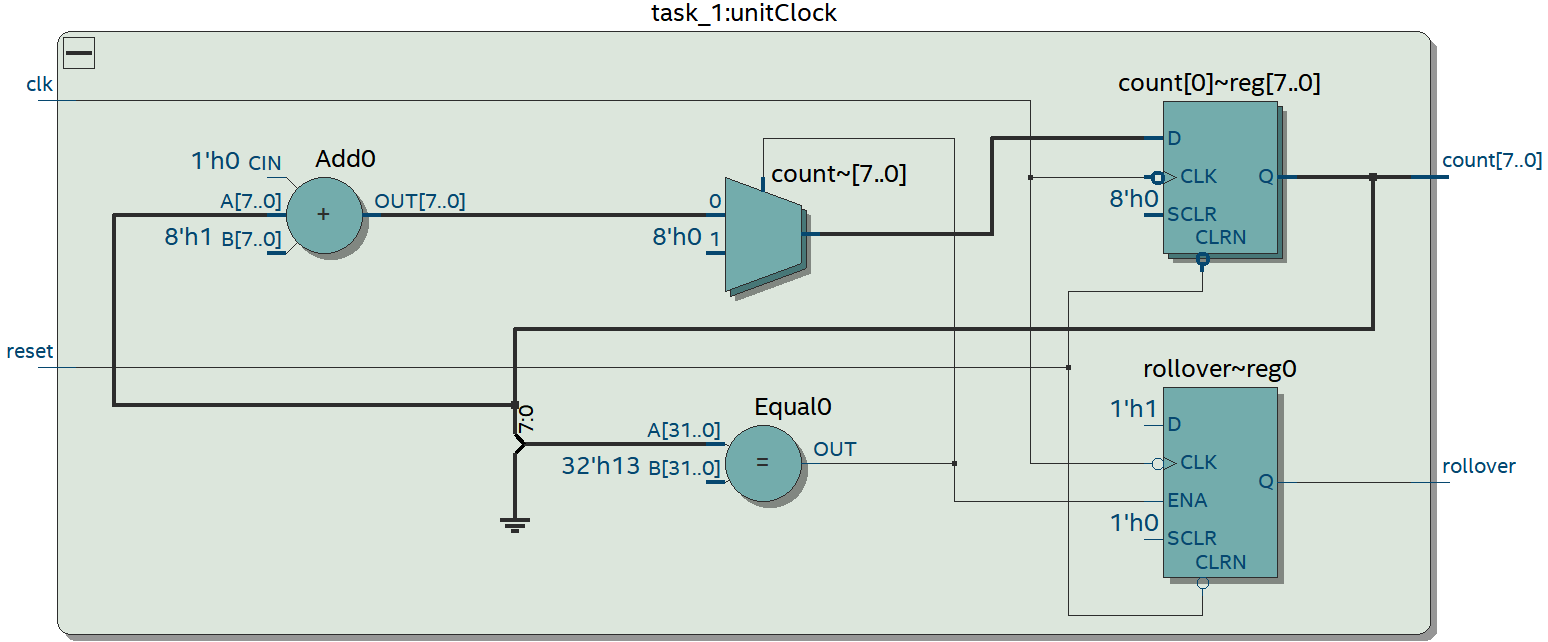
            count <= 0;

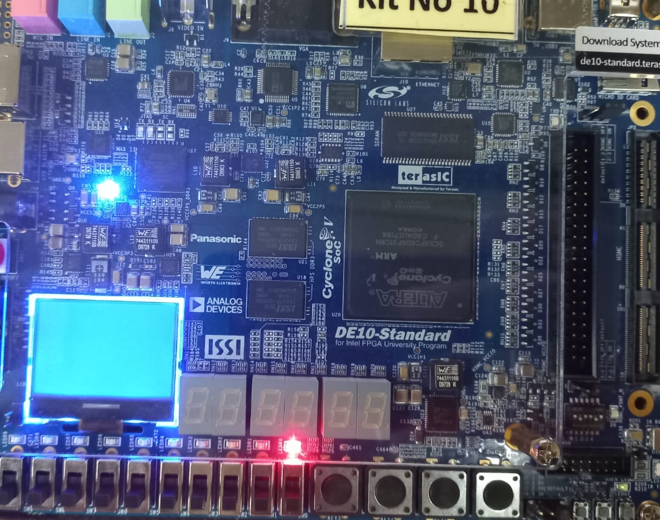
            rollover <= 1;

        end else count <= count + 1;

    end

endmodule





## 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

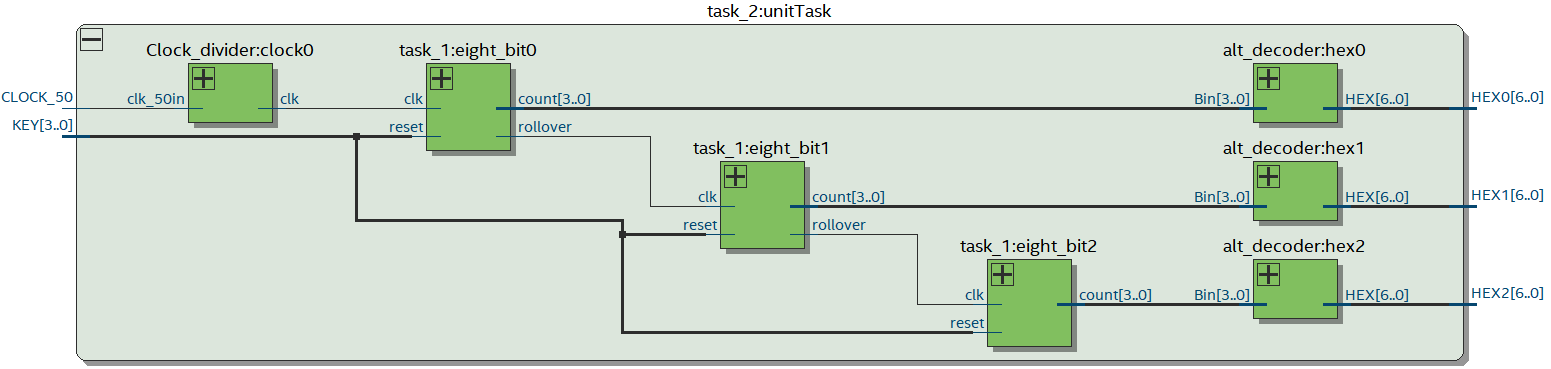
        count\_reg <= 32'd00000000;

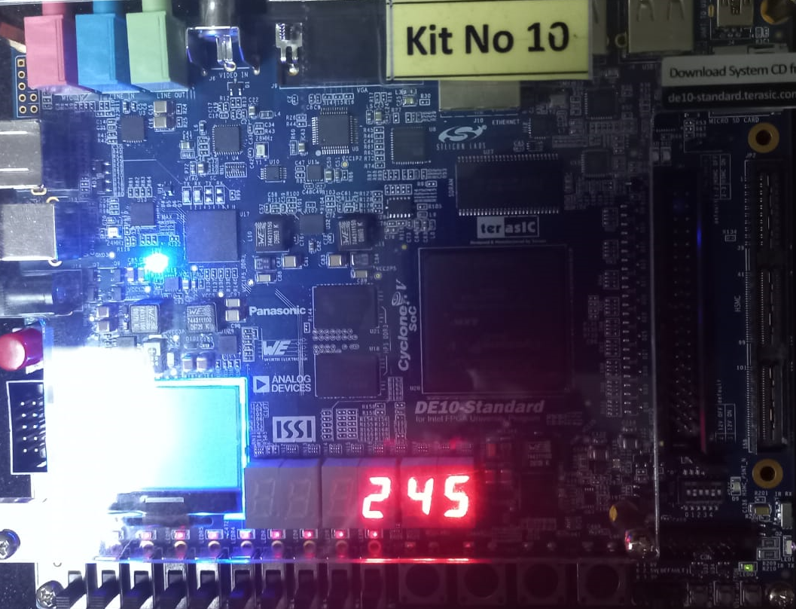
        clk <= ~clk;

    end

end

endmodule





## 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

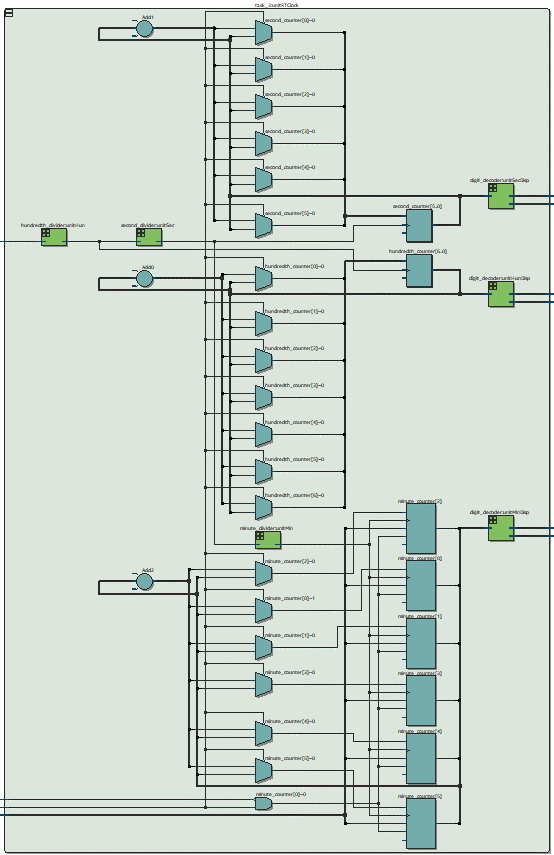
            count   <= 0;

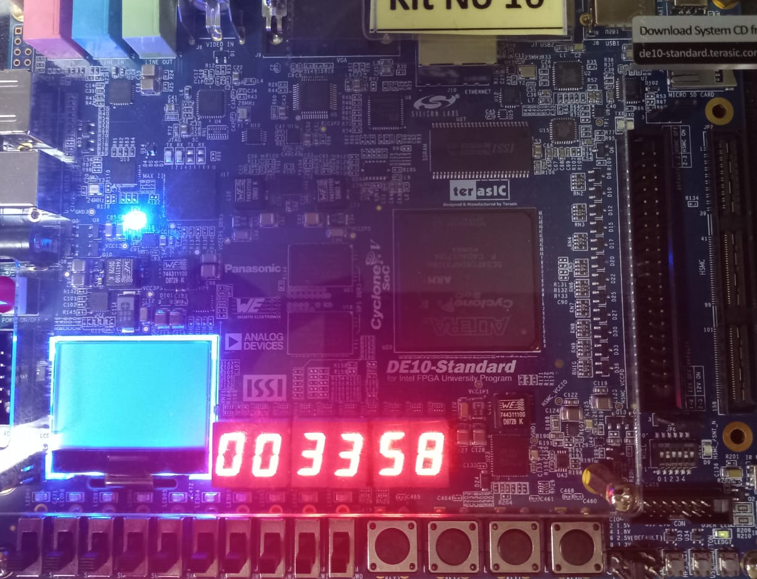
            clk\_out <= ~clk\_out;  // Invert the output clock

        end

    end

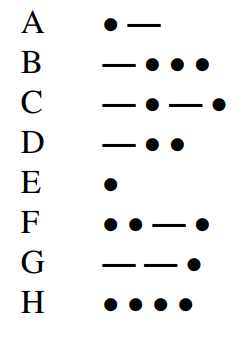
endmodule



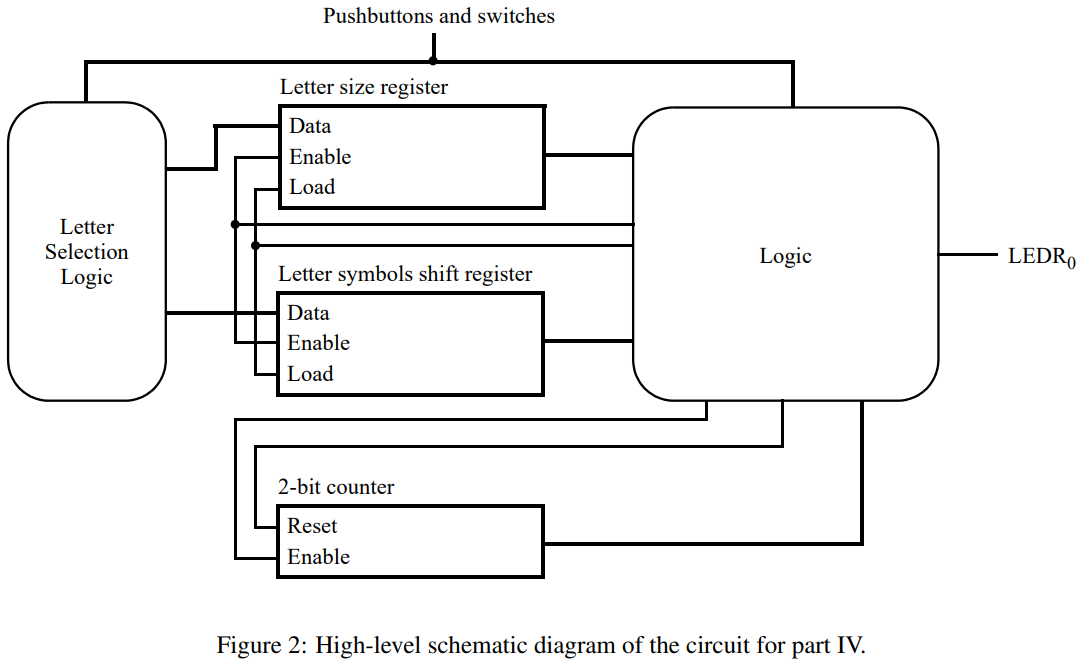


## Part IV

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:



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.



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

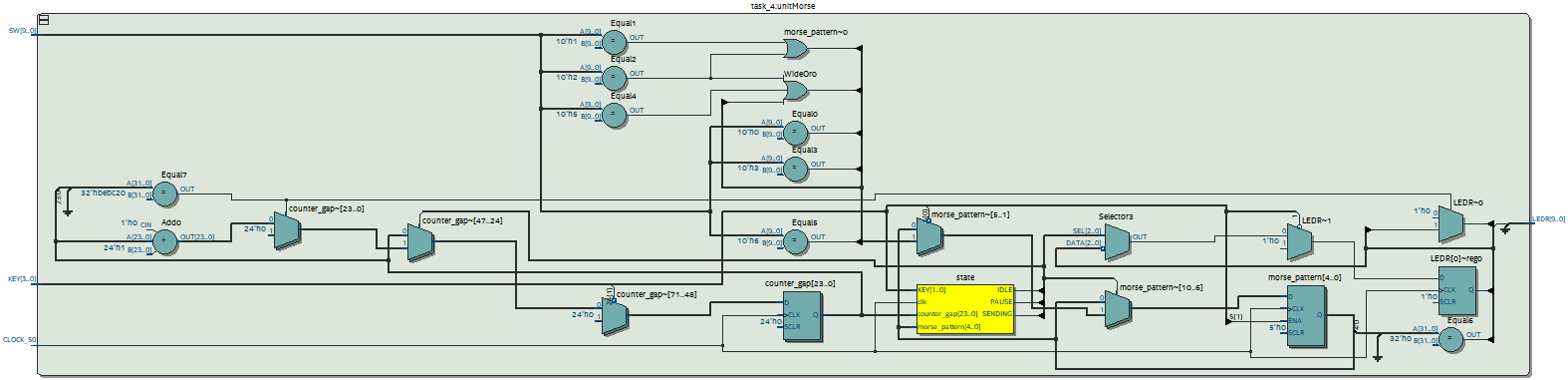
                end

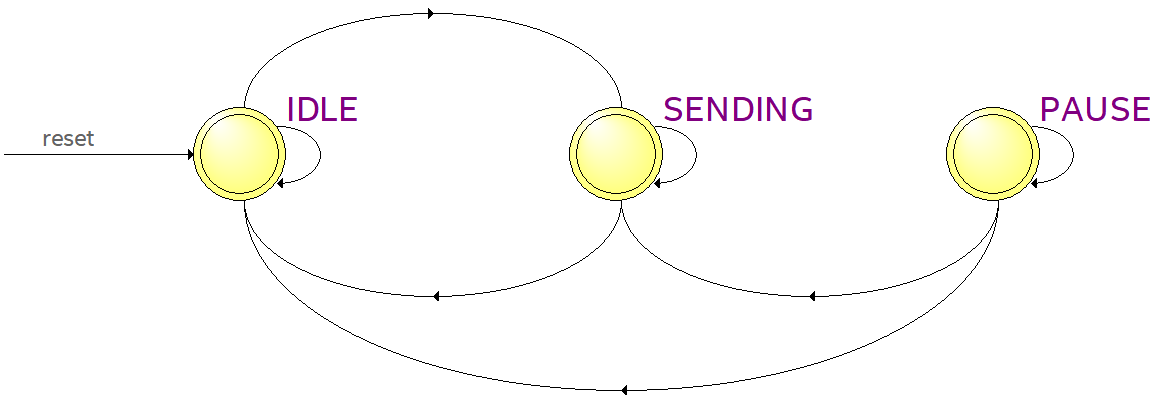
            endcase

        end

    end

endmodule





# 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.