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 5: Counters

## 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 Counters** ........ **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 ........ 6

3.3 Part III ........ 7

3.4 Part IV ........ 9

3.5 Part V ........ 11

**4 Conclusion** ........ **14**

# 2 Counters

## 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 circuits and are widely used in various applications, such as frequency dividers, time delay generators, and event counters. They are essential for creating synchronous digital systems, where different parts of the circuit are synchronized to a common clock signal. Counters can be implemented using different types of latches and flip-flops, such as D flip-flops, T flip-flops, and JK flip-flops.

In this lab, we aim to build and use counters using Verilog hardware description language and implement them on Intel FPGA DE10-Standard Board. The lab exercises will involve designing and implementing different types of counters, including binary counters, up/down counters, and ring counters. Additionally, we will explore the different types of latches and flip-flops and their functionality.

## 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. Write a Verilog file that defines an 8-bit counter by using the structure depicted in Figure 1. Your code should include a T flip-flop module that is instantiated eight times to create the counter. Compile the circuit. How many logic elements (LEs) are used to implement your circuit?
2. Simulate your circuit to verify its correctness.
3. Augment your Verilog file to use the pushbutton KEY0 as the Clock input and switches SW1 and SW0 as Enable and Clear inputs, and 7-segment displays HEX1-0 to display the hexadecimal count as your circuit operates. Make the necessary pin assignments needed to implement the circuit on your DE-series board and compile the circuit.
4. Download your circuit into the FPGA chip and test its functionality by operating the switches.
5. Implement a four-bit version of your circuit and use the Quartus RTL Viewer to see how the Quartus software synthesized the circuit. What are the differences in comparison with Figure 1?

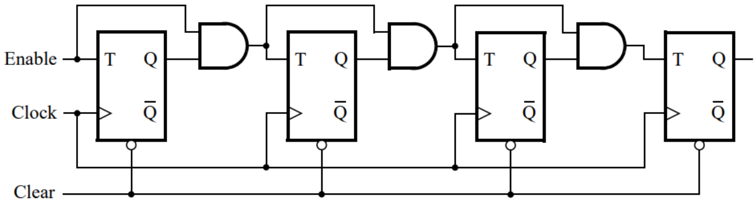
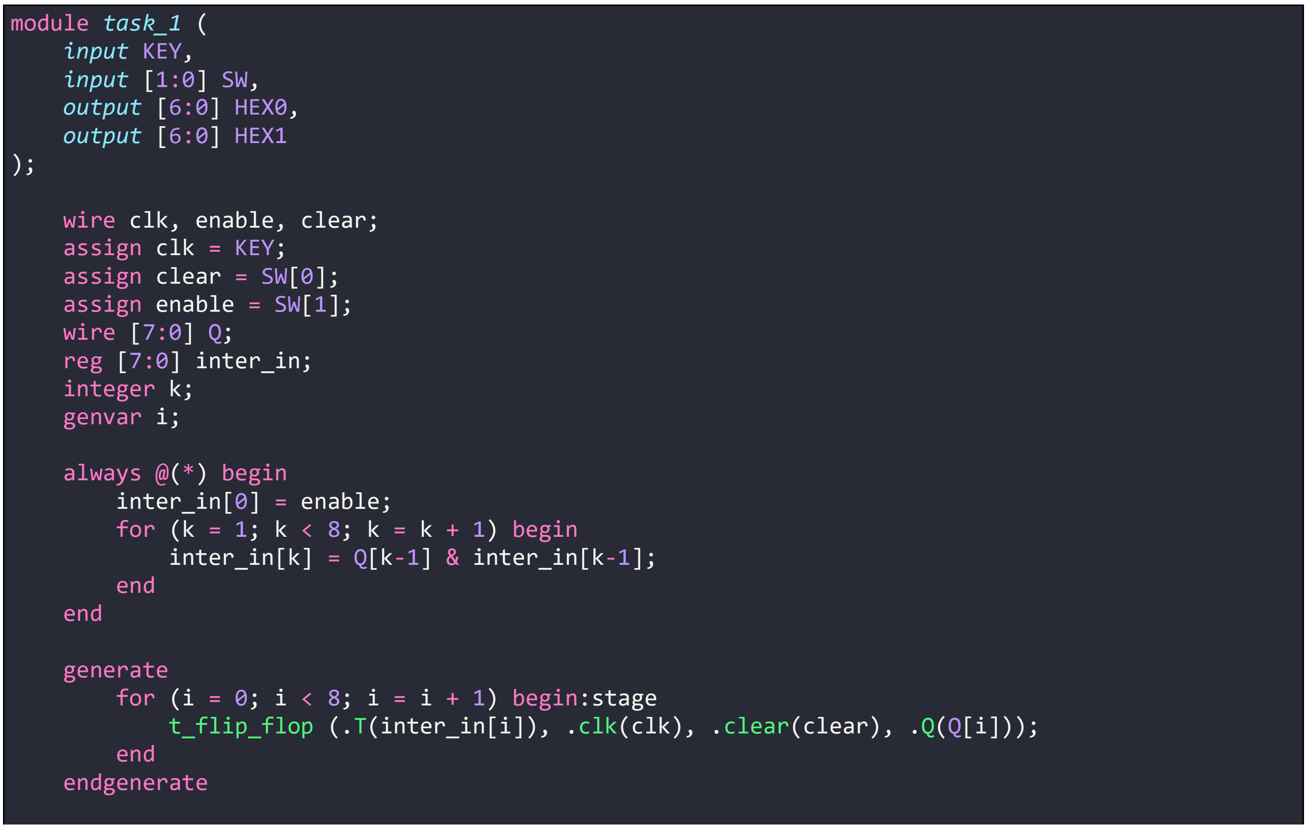
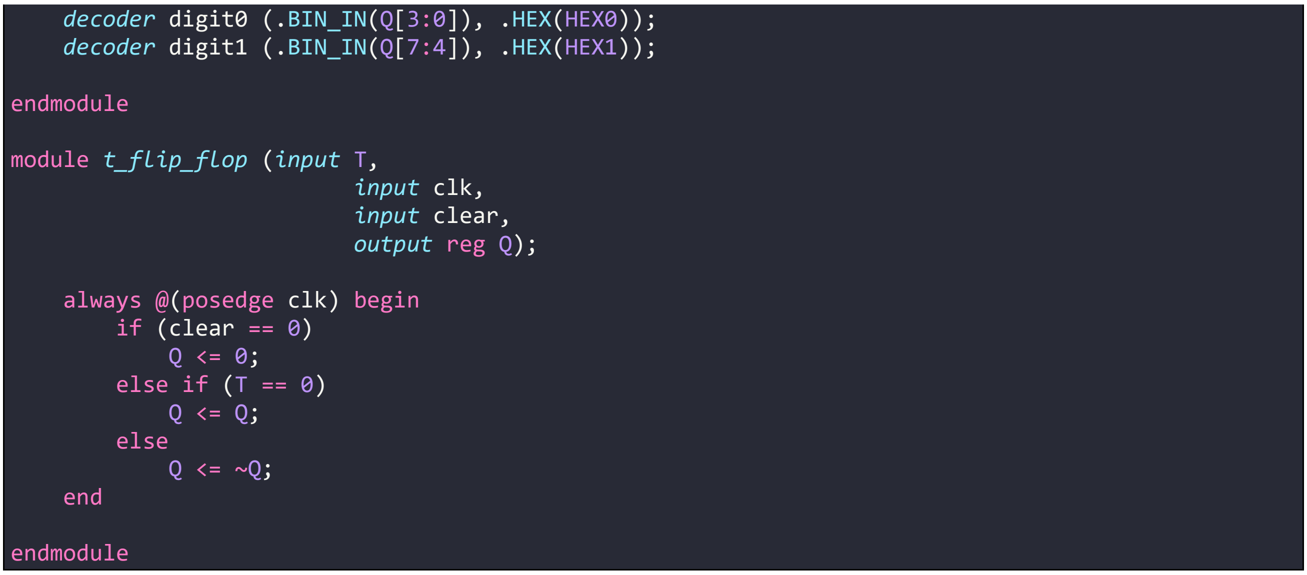


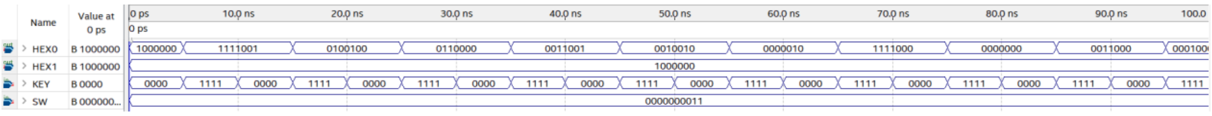
Figure 1: A 4-bit counter.

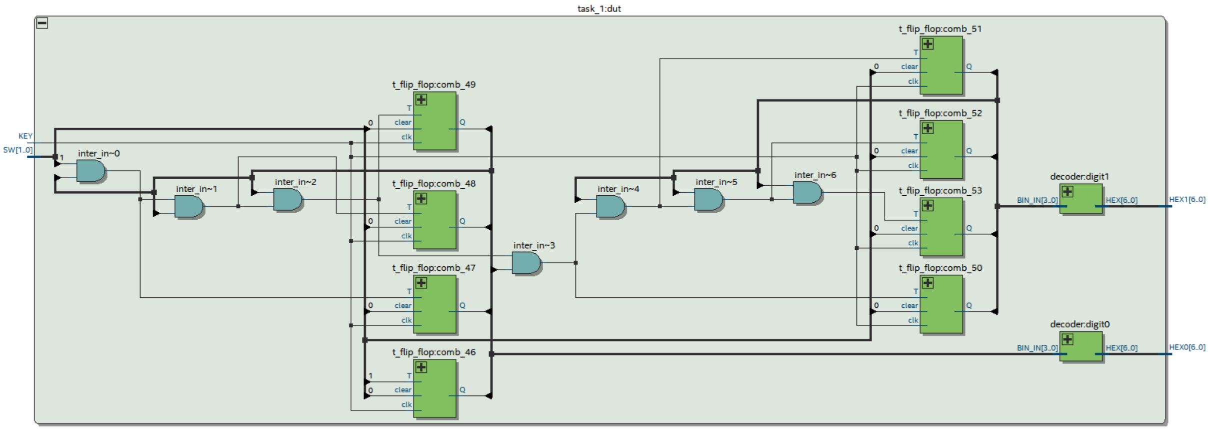


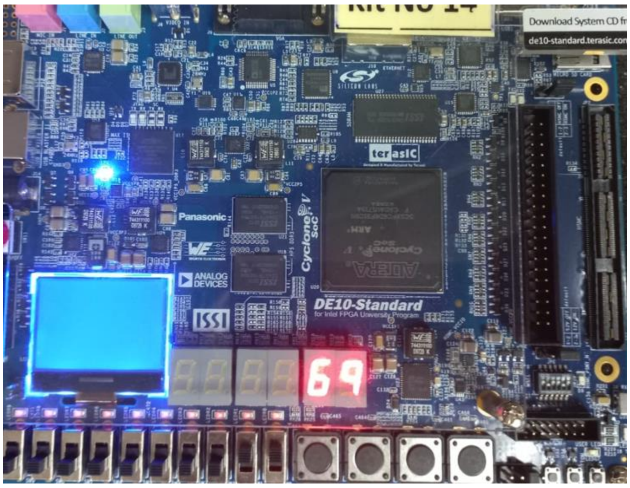
```
module task_1 (
    input KEY,
    input [1:0] SW,
    output [6:0] HEX0,
    output [6:0] HEX1
);

    wire clk, enable, clear;
    assign clk = KEY;
    assign clear = SW[0];
    assign enable = SW[1];
    wire [7:0] Q;
    reg [7:0] inter_in;
    integer k;
    genvar i;

    always @(*) begin
        inter_in[0] = enable;
        for (k = 1; k < 8; k = k + 1) begin
            inter_in[k] = Q[k-1] & inter_in[k-1];
        end
    end

    generate
        for (i = 0; i < 8; i = i + 1) begin:stage
            t_flip_flop (.T(inter_in[i]), .clk(clk), .clear(clear), .Q(Q[i]));
        end
    endgenerate
```

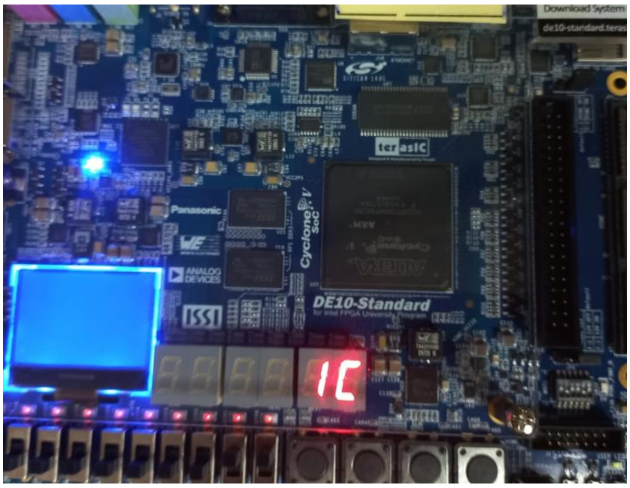


```
    decoder digit0 (.BIN_IN(Q[3:0]), .HEX(HEX0));
    decoder digit1 (.BIN_IN(Q[7:4]), .HEX(HEX1));

endmodule

module t_flip_flop (input T,
                          input clk,
                          input clear,
                          output reg Q);

    always @(posedge clk) begin
        if (clear == 0)
            Q <= 0;
        else if (T == 0)
            Q <= Q;
        else
            Q <= ~Q;
    end

endmodule
```









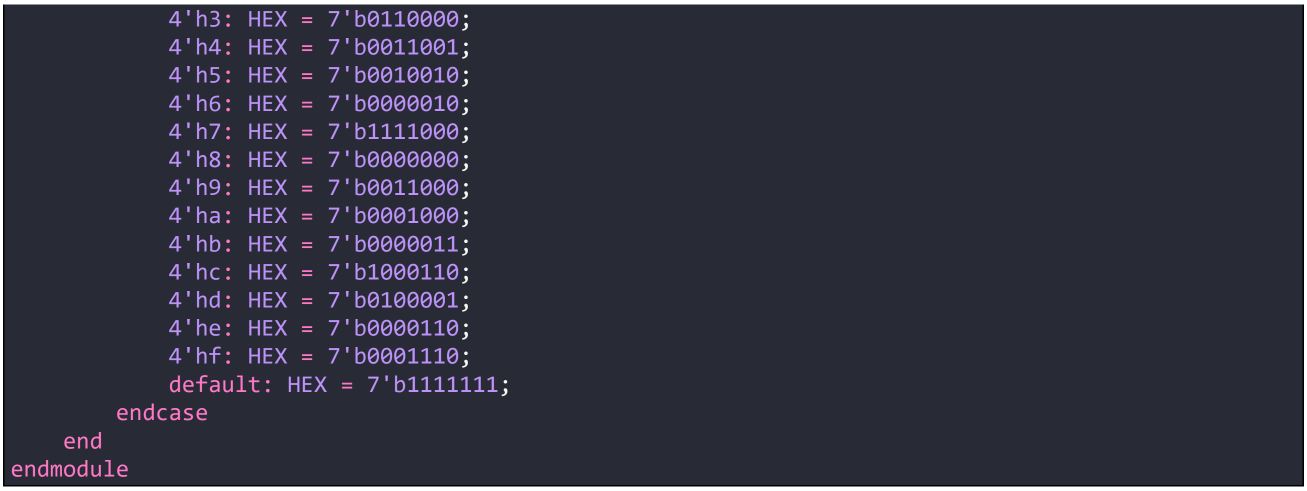
## 3.2 Part II

Another way to specify a counter is by using a register and adding 1 to its value. This can be accomplished using the following Verilog statement:

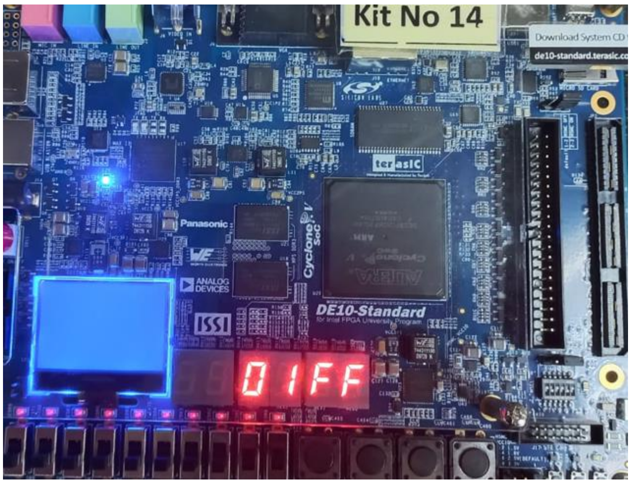
Q <= Q + 1;

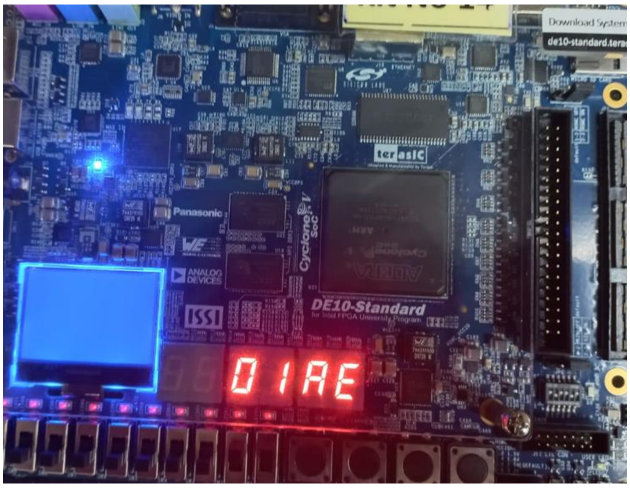
Compile a 16-bit version of this counter and determine the number of LEs needed. Use the RTL Viewer to see the structure of this implementation and comment on the differences with the design from Part I. Implement the counter on your DE-series board, using the displays HEX3-0 to show the counter value.

```verilog
module task_2 (
    input  [9:0] SW,
    input  [3:0] KEY,
    output [6:0] HEX0,
    output [6:0] HEX1,
    output [6:0] HEX2,
    output [6:0] HEX3
);

    reg [15:0] Counter_out;
    wire clk = KEY[0];
    wire clear = SW[0];
    wire enable = SW[1];

    always @(posedge clk) begin

        if (!clear) Counter_out <= 0;
        else if (enable) Counter_out <= Counter_out + 1;
        else Counter_out <= Counter_out;
    end

    BIN2HEX h0 (
        .Bin(Counter_out[3:0]),
        .HEX(HEX0)
    );
    BIN2HEX h1 (
        .Bin(Counter_out[7:4]),
        .HEX(HEX1)
    );
    BIN2HEX h2 (
        .Bin(Counter_out[11:8]),
        .HEX(HEX2)
    );
    BIN2HEX h3 (
        .Bin(Counter_out[15:12]),
        .HEX(HEX3)
    );

endmodule

module BIN2HEX (
    input [3:0] Bin,
    output reg [6:0] HEX
);
    always @(*) begin
        case (Bin)
            4'h0: HEX = 7'b1000000;
            4'h1: HEX = 7'b1111001;
            4'h2: HEX = 7'b0100100;
```



```
            4'h3: HEX = 7'b0110000;
            4'h4: HEX = 7'b0011001;
            4'h5: HEX = 7'b0010010;
            4'h6: HEX = 7'b0000010;
            4'h7: HEX = 7'b1111000;
            4'h8: HEX = 7'b0000000;
            4'h9: HEX = 7'b0011000;
            4'ha: HEX = 7'b0001000;
            4'hb: HEX = 7'b0000011;
            4'hc: HEX = 7'b1000110;
            4'hd: HEX = 7'b0100001;
            4'he: HEX = 7'b0000110;
            4'hf: HEX = 7'b0001110;
            default: HEX = 7'b1111111;
        endcase
    end
endmodule
```





## 3.3 Part III

Design and implement a circuit that successively flashes digits 0 through 9 on the 7-segment display HEX0. Each digit should be displayed for about one second. Use a counter to determine the one-second intervals. The counter should be incremented by the 50-MHz clock signal provided on the DE-series boards. Do not derive any other clock signals in your design–make sure that all flip-flops in your circuit are clocked directly by the 50-MHz clock signal. A partial design of the required circuit is shown in Figure 2.

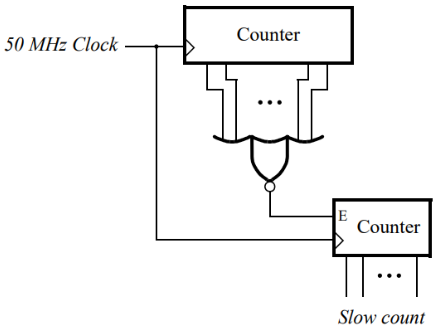
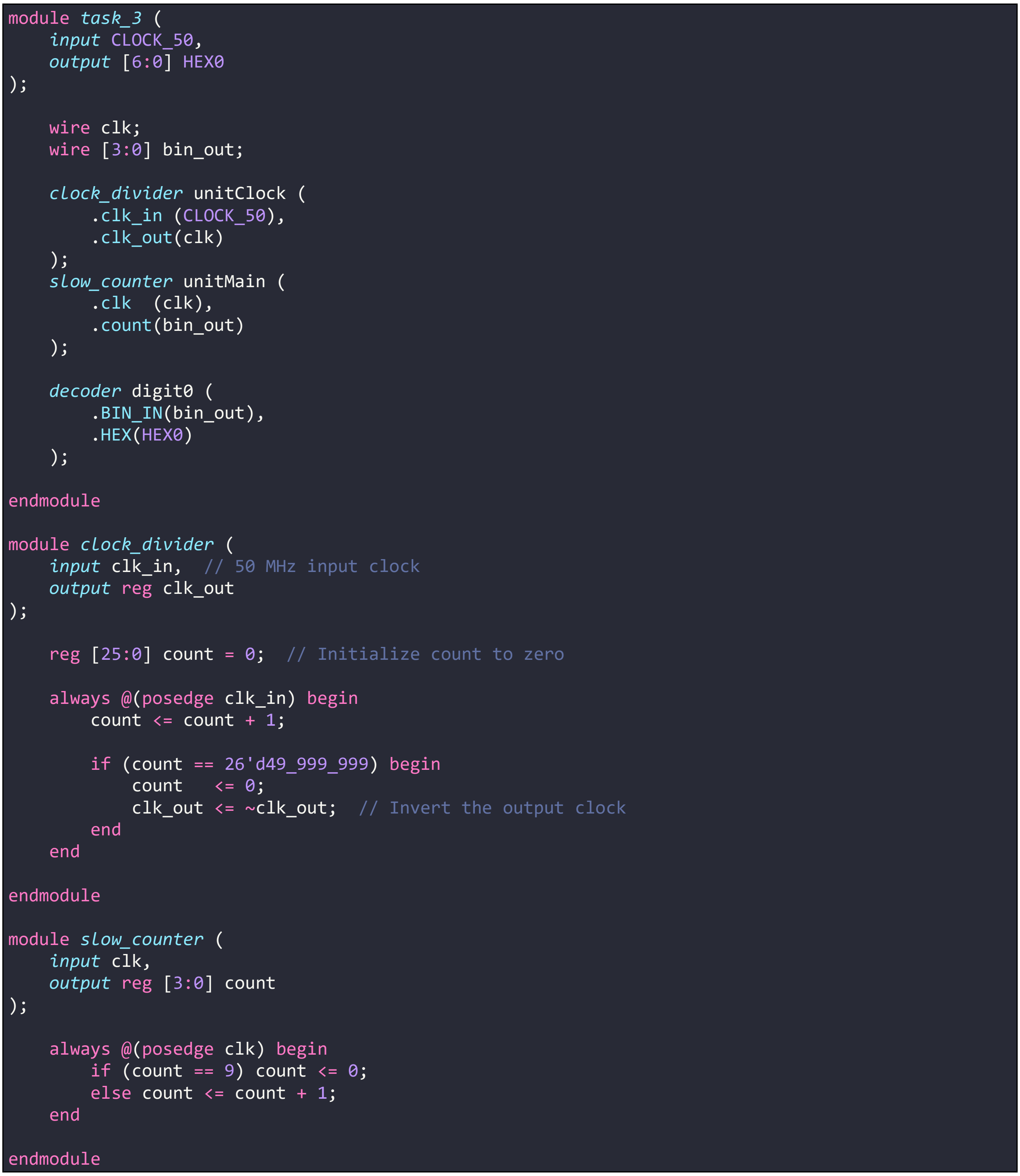
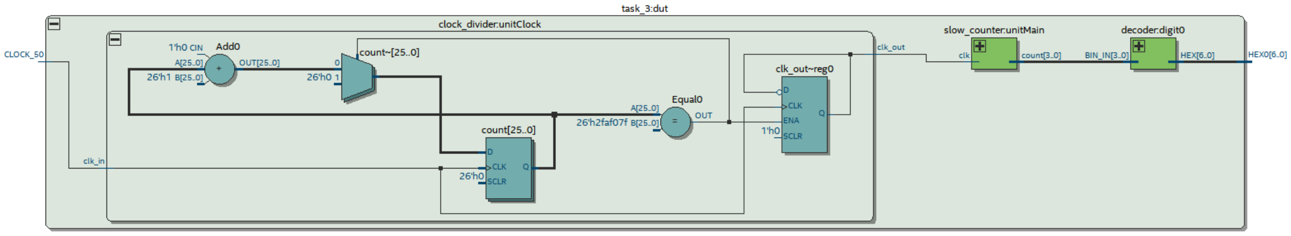
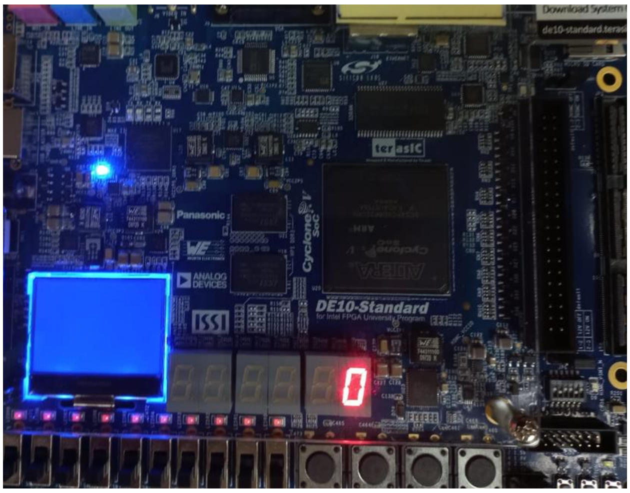


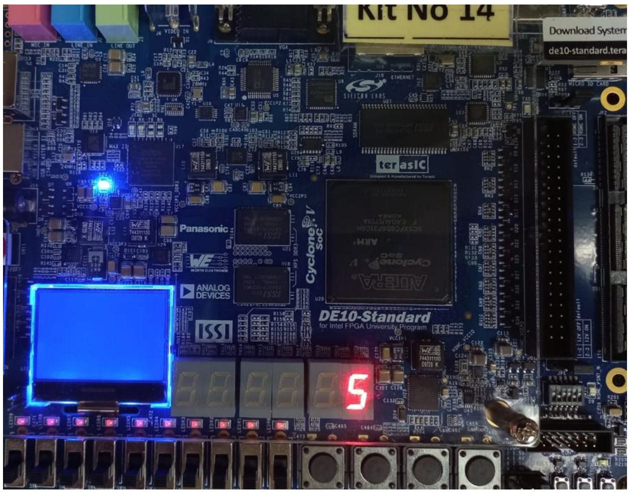
Figure 2: Making a slow counter.



```
module task_3 (
    input CLOCK_50,
    output [6:0] HEX0
);

    wire clk;
    wire [3:0] bin_out;

    clock_divider unitClock (
        .clk_in (CLOCK_50),
        .clk_out(clk)
    );
    slow_counter unitMain (
        .clk  (clk),
        .count(bin_out)
    );

    decoder digit0 (
        .BIN_IN(bin_out),
        .HEX(HEX0)
    );

endmodule

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

    reg [25:0] count = 0;  // Initialize count to zero

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

        if (count == 26'd49_999_999) begin
            count   <= 0;
            clk_out <= ~clk_out;  // Invert the output clock
        end
    end

endmodule

module slow_counter (
    input clk,
    output reg [3:0] count
);

    always @(posedge clk) begin
        if (count == 9) count <= 0;
        else count <= count + 1;
    end

endmodule
```







### 3.4 Part IV

Design and implement a circuit that displays a word on four 7-segment displays HEX3 − 0. The word to be displayed for your DE-series board is given in Table 1. Make the letters rotate from right to left in intervals of about one second. The rotating pattern for the DE10-Lite is given in Table 2. If you are using the DE0-CV, DE1- SoC, or DE2-115, use the word given in Table 1. There are many ways to design the required circuit. One solution is to re-use the Verilog code designed in Laboratory Exercise 1, Part V. Using that code, the main change needed is to replace the two switches that are used to select the characters being rotated on the displays with a 2-bit counter that increments at one-second intervals.

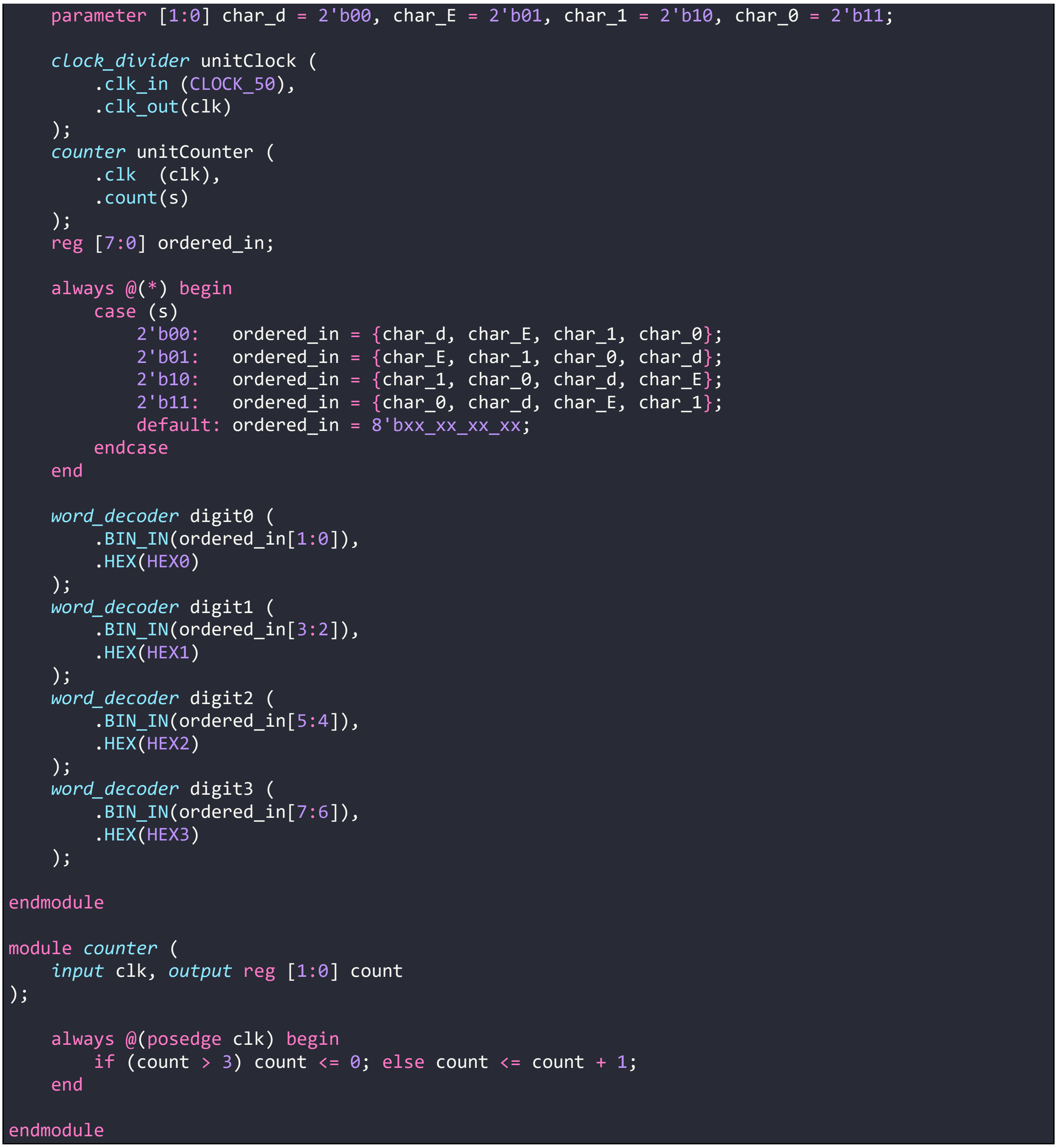
| Board | Word |
|---|---|
| DE10-Lite | dE10 |
| DE0-CV | dE0 |
| DE1-SoC | dE1 |
| DE2-115 | dE2 |

Table 1: DE-series boards and corresponding word to display

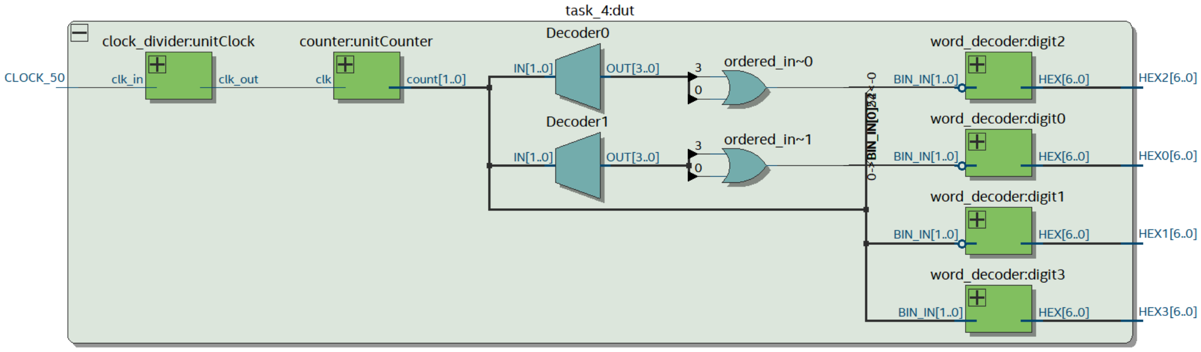
| Count | Characters | | | |
|---|---|---|---|---|
| 00 | d | E | 1 | 0 |
| 01 | E | 1 | 0 | d |
| 10 | 1 | 0 | d | E |
| 11 | 0 | d | E | 1 |

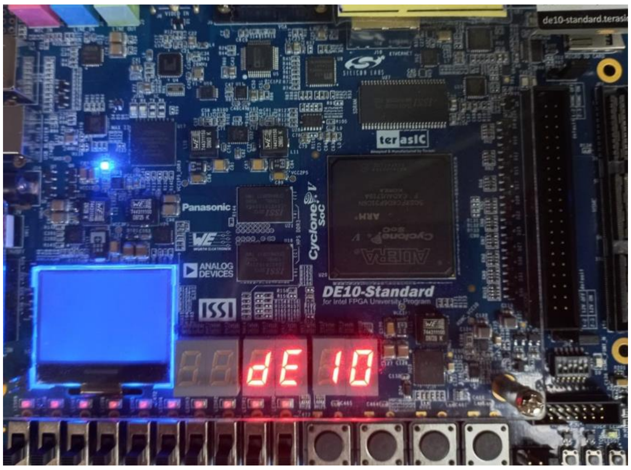
Table 2: Rotating the word dE10 on four displays.

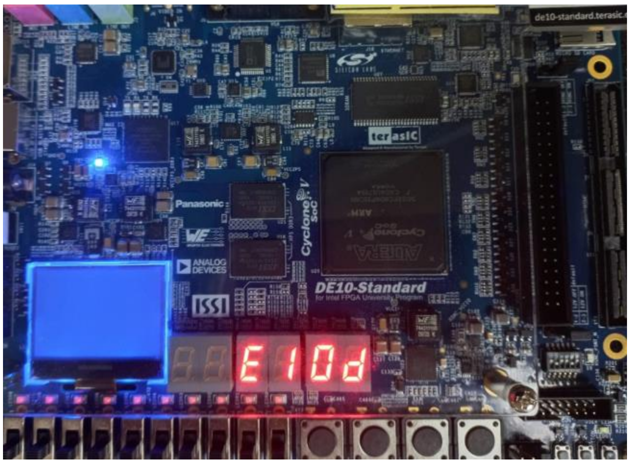
```verilog
module task_4 (
    input CLOCK_50,
    output [6:0] HEX0, output [6:0] HEX1,
    output [6:0] HEX2, output [6:0] HEX3
);

    wire [1:0] s;
    wire clk;
```



```
    parameter [1:0] char_d = 2'b00, char_E = 2'b01, char_1 = 2'b10, char_0 = 2'b11;

    clock_divider unitClock (
        .clk_in (CLOCK_50),
        .clk_out(clk)
    );
    counter unitCounter (
        .clk  (clk),
        .count(s)
    );
    reg [7:0] ordered_in;

    always @(*) begin
        case (s)
            2'b00:   ordered_in = {char_d, char_E, char_1, char_0};
            2'b01:   ordered_in = {char_E, char_1, char_0, char_d};
            2'b10:   ordered_in = {char_1, char_0, char_d, char_E};
            2'b11:   ordered_in = {char_0, char_d, char_E, char_1};
            default: ordered_in = 8'bxx_xx_xx_xx;
        endcase
    end

    word_decoder digit0 (
        .BIN_IN(ordered_in[1:0]),
        .HEX(HEX0)
    );
    word_decoder digit1 (
        .BIN_IN(ordered_in[3:2]),
        .HEX(HEX1)
    );
    word_decoder digit2 (
        .BIN_IN(ordered_in[5:4]),
        .HEX(HEX2)
    );
    word_decoder digit3 (
        .BIN_IN(ordered_in[7:6]),
        .HEX(HEX3)
    );

endmodule

module counter (
    input clk, output reg [1:0] count
);

    always @(posedge clk) begin
        if (count > 3) count <= 0; else count <= count + 1;
    end

endmodule
```







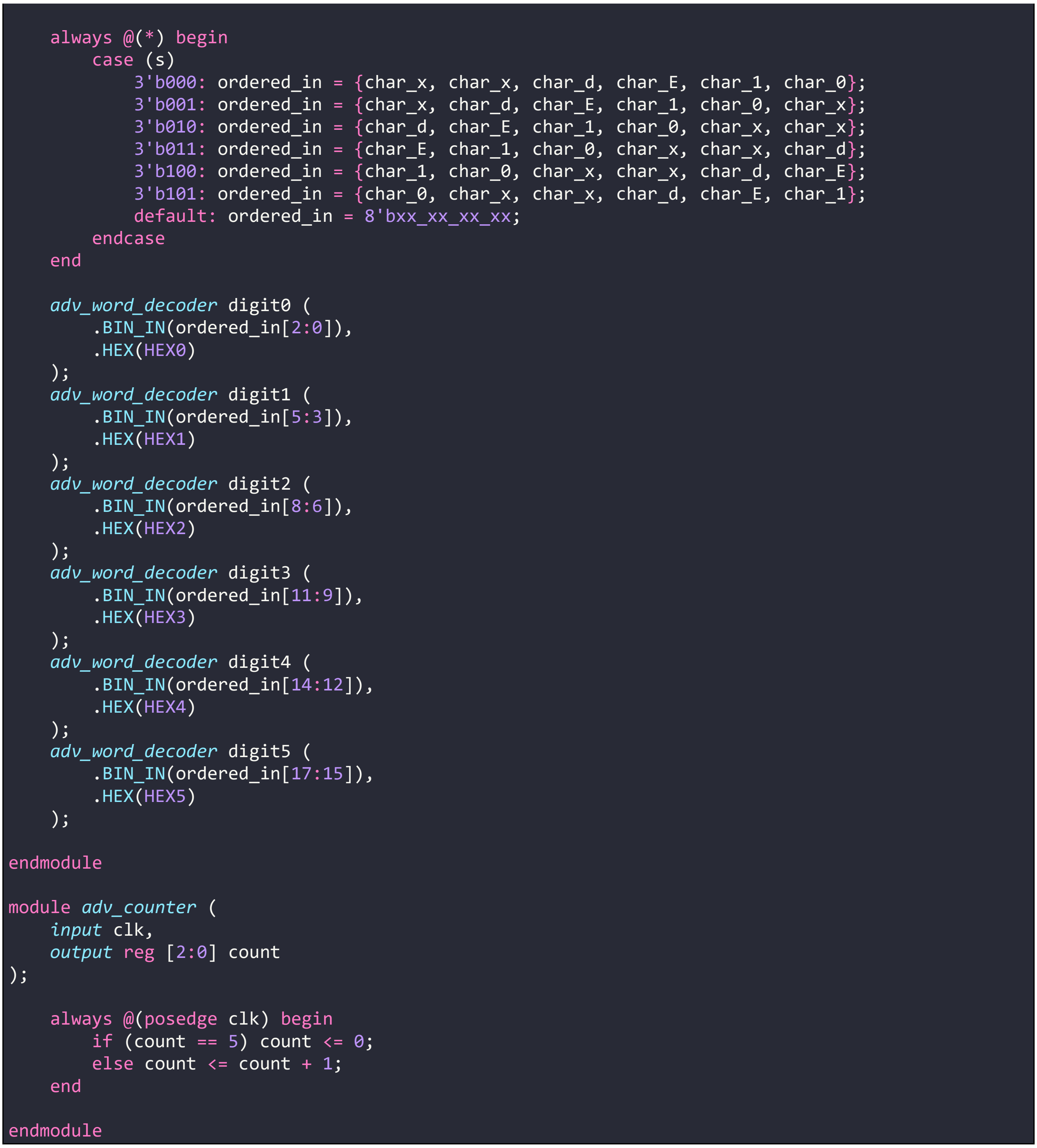
### 3.5 Part V

Augment your circuit from Part IV so that it can rotate the word over all of the 7-segment displays on your DE-series board. The shifting pattern for the DE10-Lite is shown in Table 3.

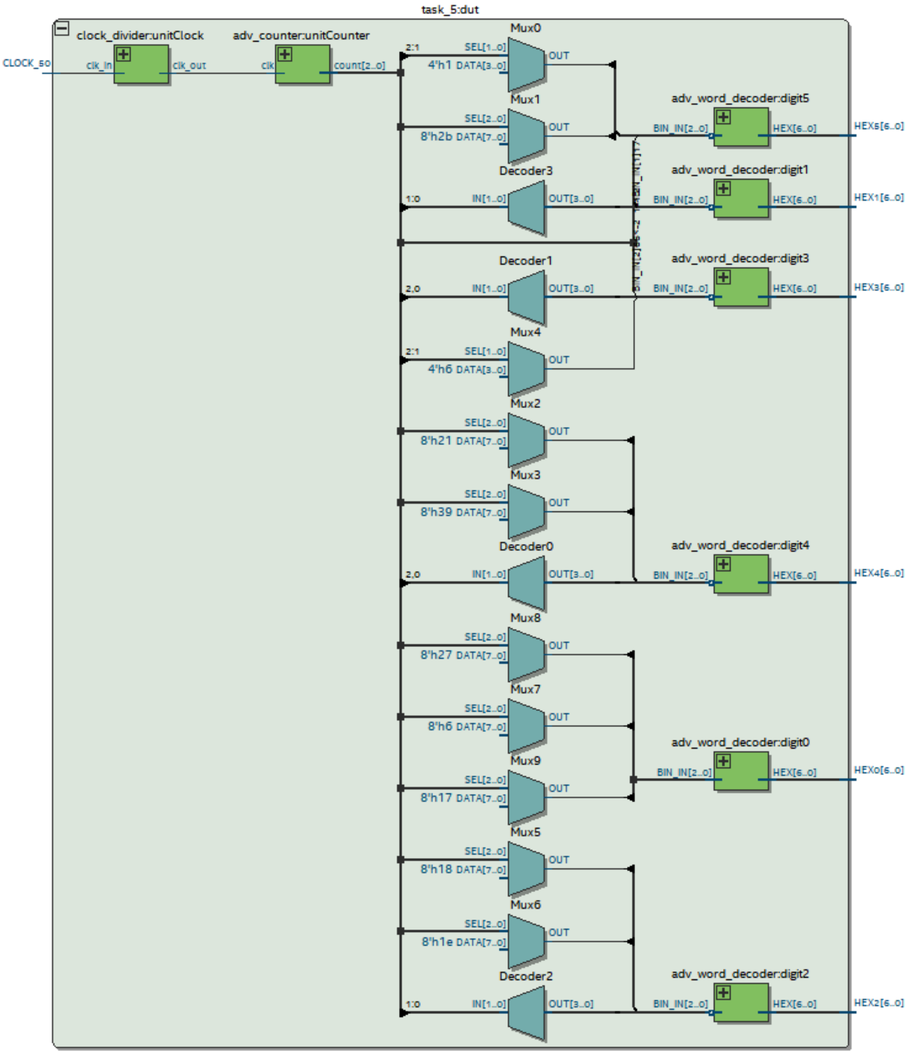
| Count | Character pattern | | | | | |
|---|---|---|---|---|---|---|
| 000 | | | d | E | 1 | 0 |
| 001 | | d | E | 1 | 0 | |
| 010 | d | E | 1 | 0 | | |
| 011 | E | 1 | 0 | | | d |
| 100 | 1 | 0 | | | d | E |
| 101 | 0 | | | d | E | 1 |

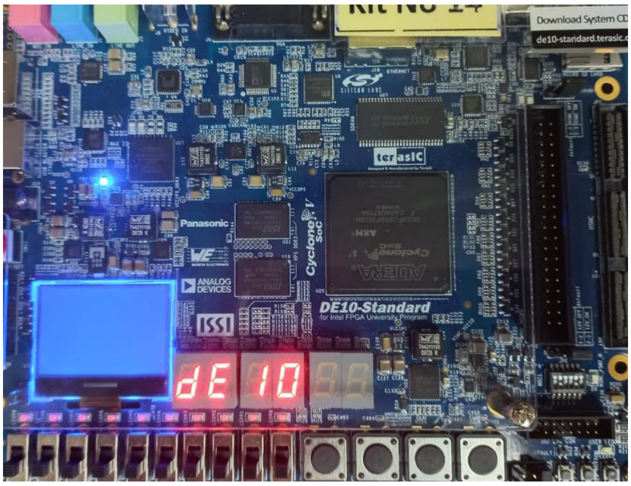
Table 3: Rotating the word dE10 on six displays.

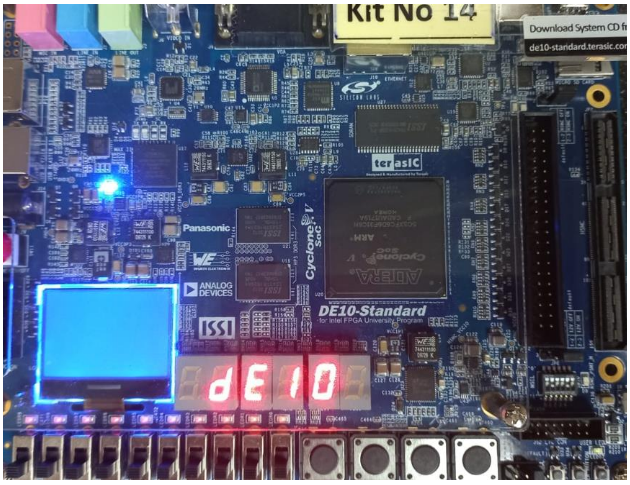
```verilog
module task_5 (
    input CLOCK_50,
    output [6:0] HEX0,
    output [6:0] HEX1,
    output [6:0] HEX2,
    output [6:0] HEX3,
    output [6:0] HEX4,
    output [6:0] HEX5
);

    wire [2:0] s;
    wire clk;

    parameter [2:0] char_d = 3'b000, char_E = 3'b001, char_1 = 3'b010,
                    char_0 = 3'b011, char_x = 3'b111;

    clock_divider unitClock (
        .clk_in (CLOCK_50),
        .clk_out(clk)
    );
    adv_counter unitCounter (
        .clk  (clk),
        .count(s)
    );

    reg [17:0] ordered_in;
```

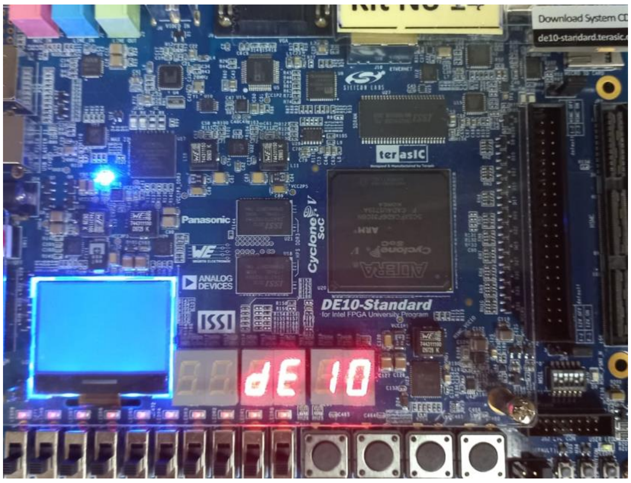


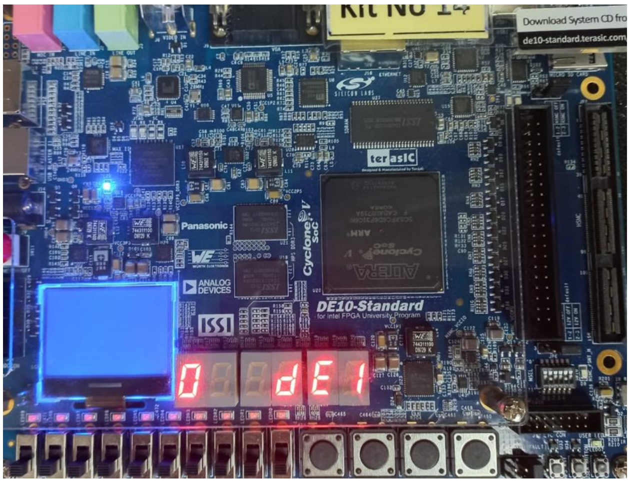
```
    always @(*) begin
        case (s)
            3'b000: ordered_in = {char_x, char_x, char_d, char_E, char_1, char_0};
            3'b001: ordered_in = {char_x, char_d, char_E, char_1, char_0, char_x};
            3'b010: ordered_in = {char_d, char_E, char_1, char_0, char_x, char_x};
            3'b011: ordered_in = {char_E, char_1, char_0, char_x, char_x, char_d};
            3'b100: ordered_in = {char_1, char_0, char_x, char_x, char_d, char_E};
            3'b101: ordered_in = {char_0, char_x, char_x, char_d, char_E, char_1};
            default: ordered_in = 8'bxx_xx_xx_xx;
        endcase
    end

    adv_word_decoder digit0 (
        .BIN_IN(ordered_in[2:0]),
        .HEX(HEX0)
    );
    adv_word_decoder digit1 (
        .BIN_IN(ordered_in[5:3]),
        .HEX(HEX1)
    );
    adv_word_decoder digit2 (
        .BIN_IN(ordered_in[8:6]),
        .HEX(HEX2)
    );
    adv_word_decoder digit3 (
        .BIN_IN(ordered_in[11:9]),
        .HEX(HEX3)
    );
    adv_word_decoder digit4 (
        .BIN_IN(ordered_in[14:12]),
        .HEX(HEX4)
    );
    adv_word_decoder digit5 (
        .BIN_IN(ordered_in[17:15]),
        .HEX(HEX5)
    );

endmodule

module adv_counter (
    input clk,
    output reg [2:0] count
);

    always @(posedge clk) begin
        if (count == 5) count <= 0;
        else count <= count + 1;
    end

endmodule
```


task_5:dut
clock_divider:unitClock
adv_counter:unitCounter
CLOCK_50
clk_in
clk_out
clk
count[2..0]
Mux0
2:1
SEL[1..0]
4'h1 DATA[3..0]
OUT
Mux1
SEL[2..0]
8'h2b DATA[7..0]
OUT
Decoder3
1:0
IN[1..0]
OUT[3..0]
Decoder1
2:0
IN[1..0]
OUT[3..0]
Mux4
2:1
SEL[1..0]
4'h6 DATA[3..0]
OUT
Mux2
SEL[2..0]
8'h21 DATA[7..0]
OUT
Mux3
SEL[2..0]
8'h39 DATA[7..0]
OUT
Decoder0
2:0
IN[1..0]
OUT[3..0]
Mux8
SEL[2..0]
8'h27 DATA[7..0]
OUT
Mux7
SEL[2..0]
8'h6 DATA[7..0]
OUT
Mux9
SEL[2..0]
8'h17 DATA[7..0]
OUT
Mux5
SEL[2..0]
8'h18 DATA[7..0]
OUT
Mux6
SEL[2..0]
8'h1e DATA[7..0]
OUT
Decoder2
1:0
IN[1..0]
OUT[3..0]
adv_word_decoder:digit5
BIN_IN[2..0]
HEX[6..0]
HEX5[6..0]
adv_word_decoder:digit1
BIN_IN[2..0]
HEX[6..0]
HEX1[6..0]
adv_word_decoder:digit3
BIN_IN[2..0]
HEX[6..0]
HEX3[6..0]
adv_word_decoder:digit4
BIN_IN[2..0]
HEX[6..0]
HEX4[6..0]
adv_word_decoder:digit0
BIN_IN[2..0]
HEX[6..0]
HEX0[6..0]
adv_word_decoder:digit2
BIN_IN[2..0]
HEX[6..0]
HEX2[6..0]


Download System CD
de10-standard.terasic.c
DE10-Standard
dE 10


Kit No 14
Download System CD
de10-standard.terasic.com
DE10-Standard
dE 10





## 4 Conclusion

In conclusion, this lab has provided a valuable learning experience in designing and implementing digital counters using Verilog and FPGA boards. We have successfully implemented binary counters, as well as a clock divider to produce accurate delays in our circuit. We have also reimplemented the rolling "DE10" keyword on the HEX displays using a delayed counter. This has enhanced our skills in using Verilog to describe digital circuits and the practical application of digital systems.