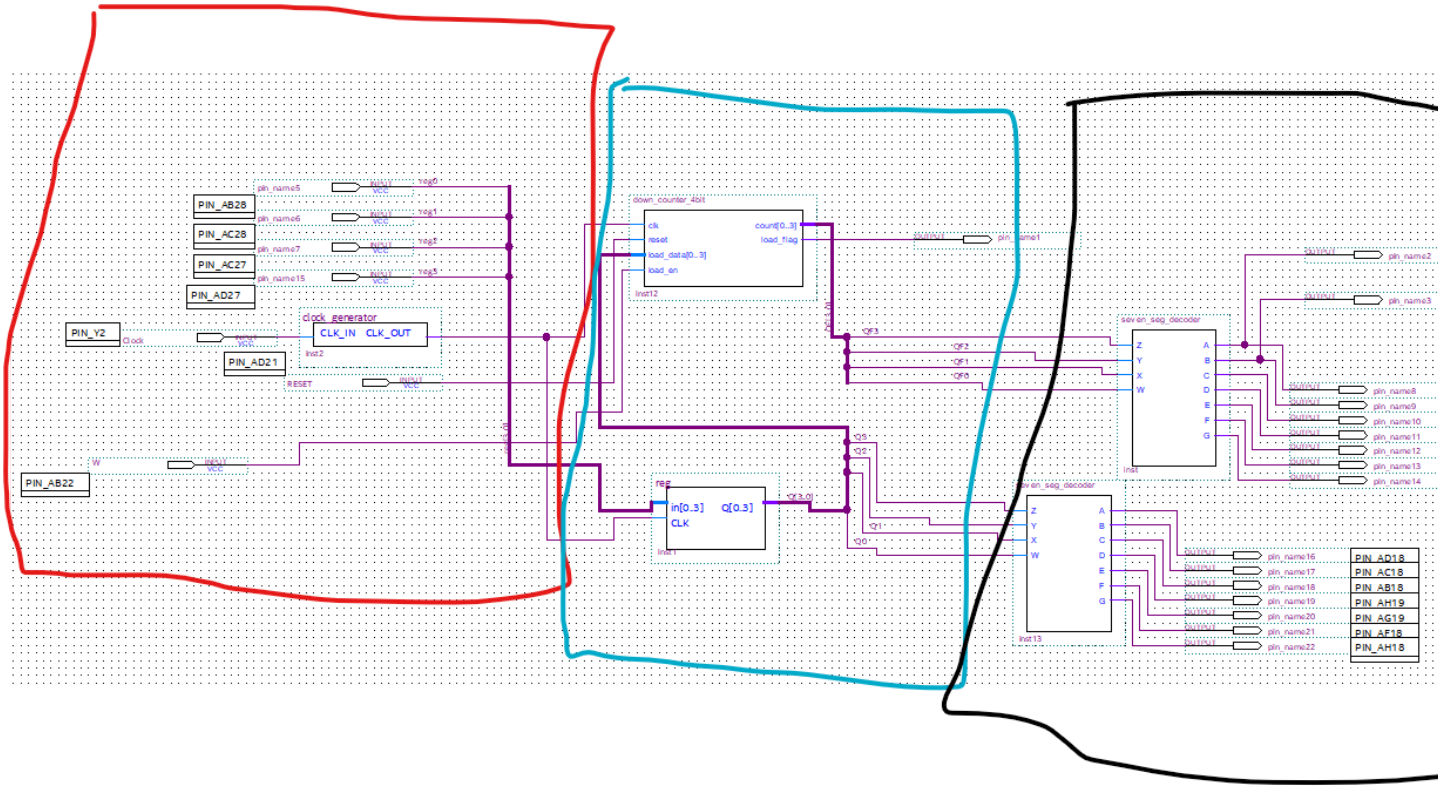
Final Project Report

CPR E 281
Student ID: 936458540

I designed a Time Counter that will be able to set and countdown times given. You enter a number using the side switches, which appear on the seven-segment display, showing the numbers input from the switches. The register file will be used to lock in a specific number, or it can be used as a clear button to input a different number. Then the rocket button will start the timer and as well act as a pause for the timer, when it equals zero it will pause the timer and when it equals one it will resume the timer using a t flip-flop it to reverse between the numbers. This project will show the use between the FSM, register files ensures efficient management of timer states and user inputs as well as utilizing T flip-flops to stop and resume the timer . Additionally, with LED indicators and 7-segment displays, users can easily track the countdown progress and be notified when the timer reaches critical points or completes, enhancing user experience and functionality.

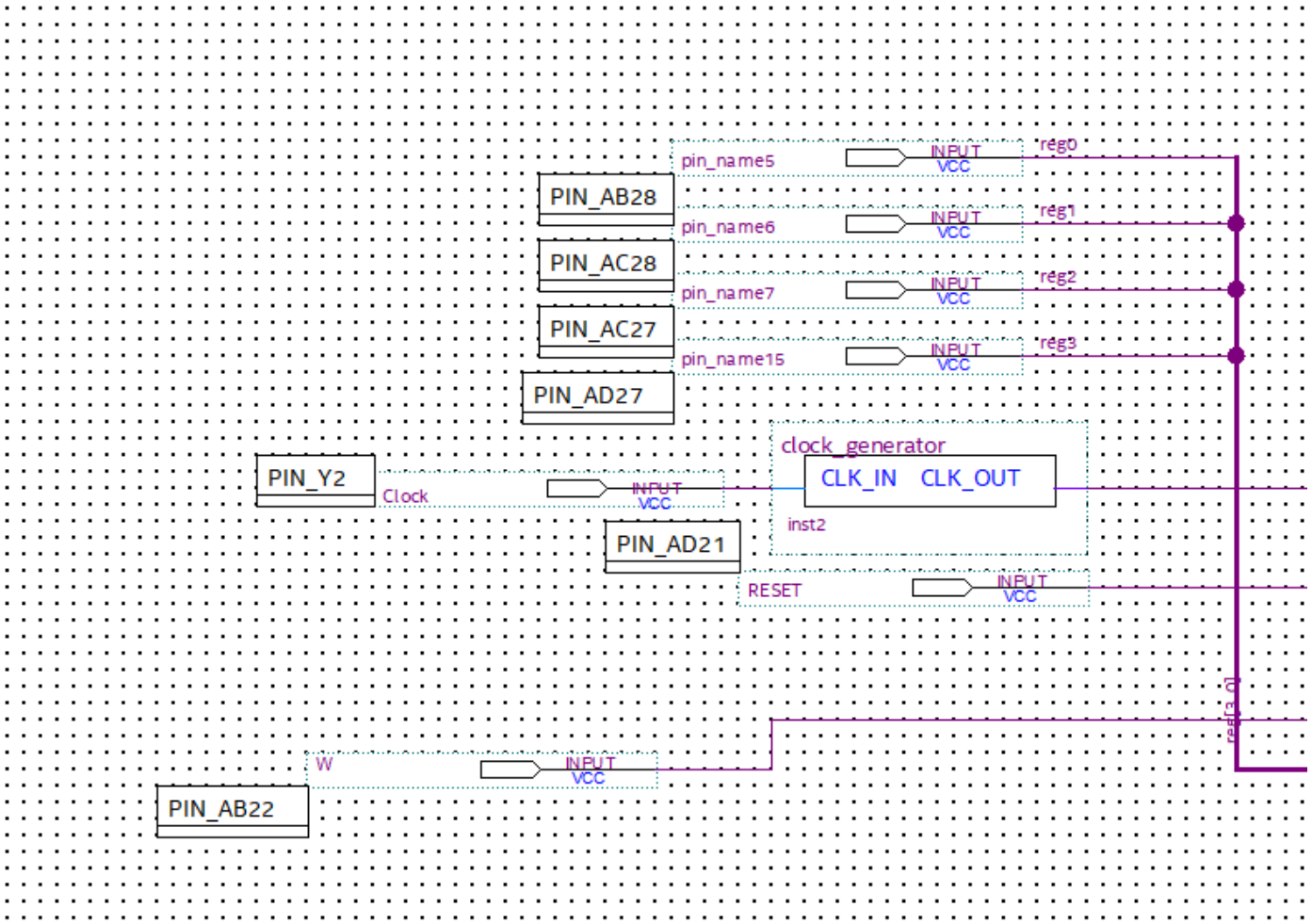
This report details all the workings required to build the circuit and Verilog modules contained in the project. It shows general view of the module, followed by a description of the inner workings of each module that explains submodule involved

Top Level Diagram



This is an overall display of the circuit. Since there is so much put in a picture its jaw-dropping I gave the pleasure in spliting into three different parts. Part 1, 2, and 3 going into how this project came to be .

Part 1

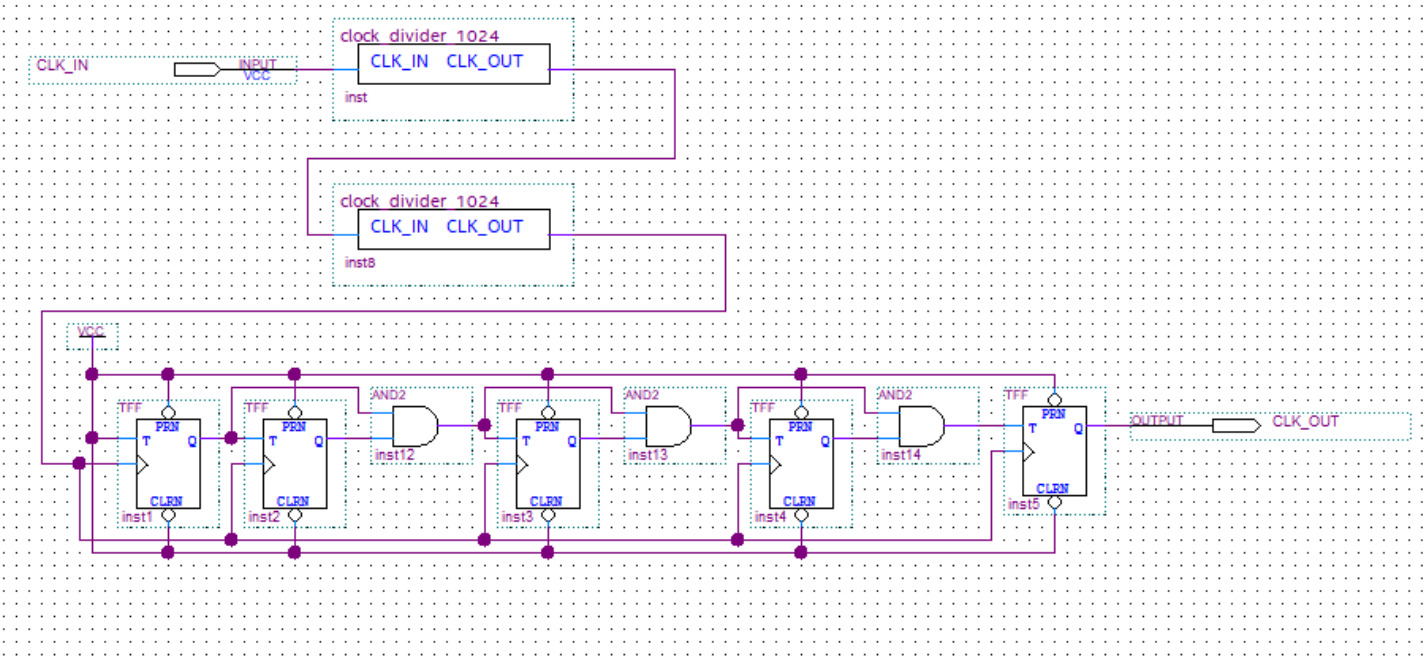


In this part, the inputs are user-base to allow the user to set a specific time on the timer, The clock ran automatically through 15 to 0 counting down. The clock frequency is divided via the **clock_divider_1024** to give a mor estable flow of how the clock is

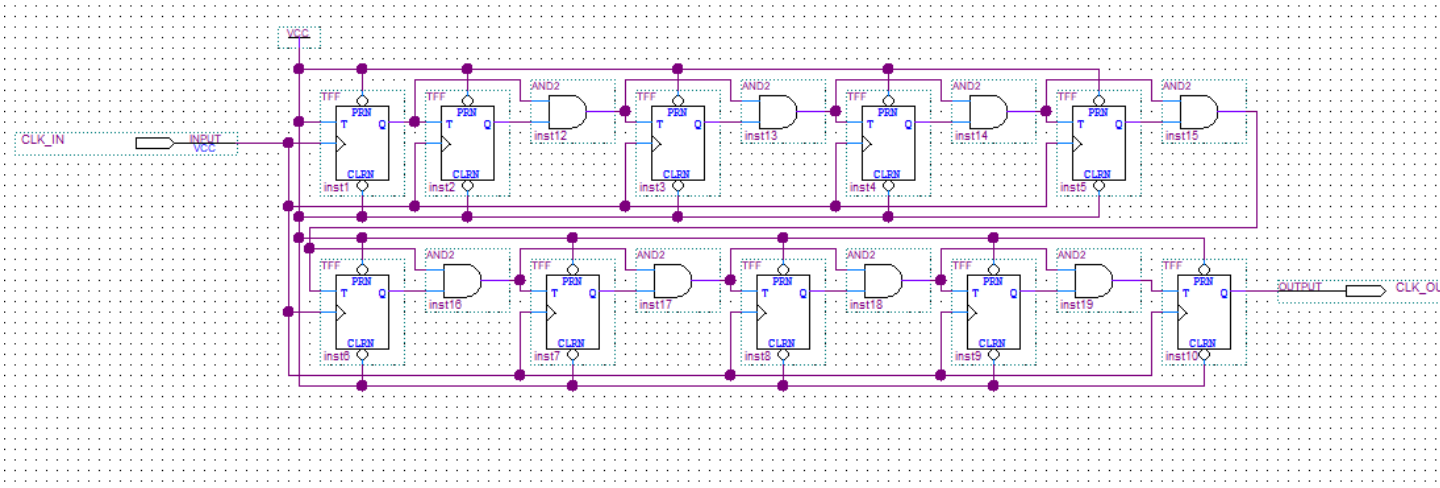
moving. Additionally, the reset button is used to reset the timer to the specified number the user has chosen from the register file(not shown). The w switch fsm used a **FSM** to iterate down the clock. When w = 0, it allows the timer to countdown from the number, when equaling to 1 it puts a stop to the clock display which enables a change to the number or continuing to down.

Clock_generator

A clock generator is a device that generates a clock signal, which is a periodic square wave that is used to synchronize the operations of digital circuits. It was used to run through the clock and be displayed on the **seven-segment display,** starting from 15 counting down to 0

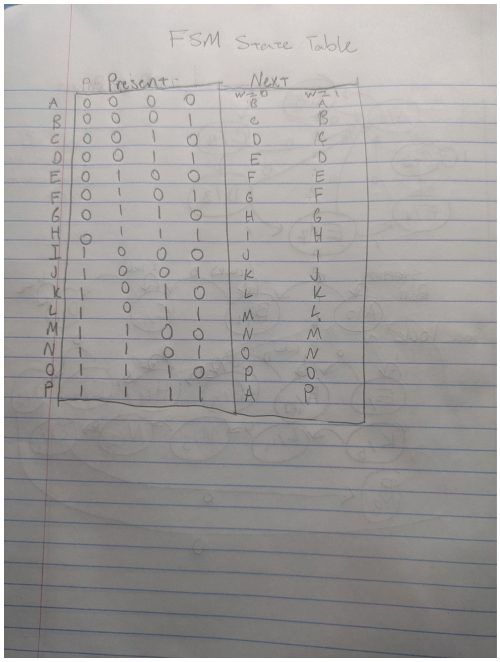


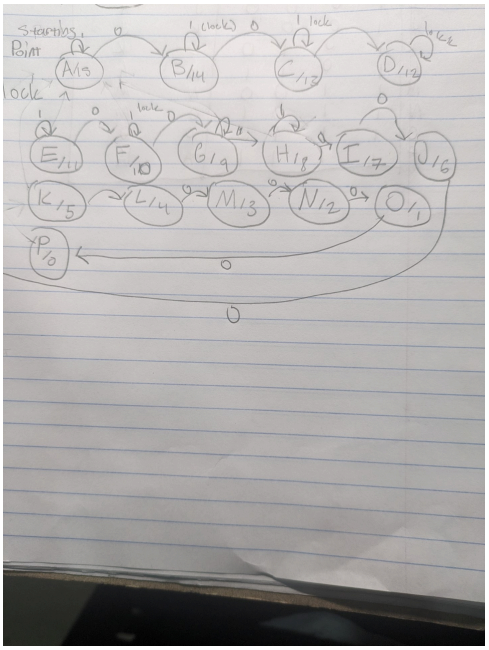
Clock_divider_1024 is a clock dividing module. It is a series of T-Flip-Flops that each divide the frequency of 2, allowing the clock to be shown at a more natural frequency to be shown, it works as ten-bit counter where the tenth bit changing between 1 and 0 is the new clock signal. The 1st flip-flop output changes at the rate of the clock, while the second changes at half the rate of the clock; the AND gates make sure that the next flip-flop does not change until the outputs of the previous two are equal to 1; this means the flip-flop is changing at the speed of the preceding one and a fourth the speed one before that.Since there are ten T-Flip-Flops, the frequency is divided by 1024.



FSM

This represents the logic composed of the state machine, initially lets look at the diagram

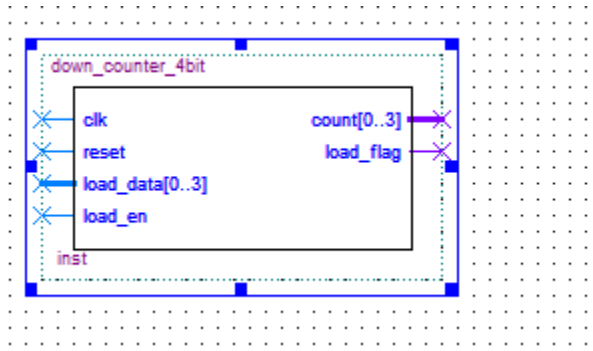




The system initiates at State 15 when the clock registers zero. Under the condition where w equals 0, it will sequentially progress through the states until it reaches the final state. Upon reaching this termination point, the system will seamlessly loop back to State A, fulfilling a dual role as both the starting point and an enabler for further transitions. Conversely, when w is set to 1, the system will halt at the current state, preserving its status. This pause prompts the user or system to engage with the register file, facilitating the loading of a new state. When it hitting it certain numbers on the timer it prompts a light from LED display This dynamic mechanism ensures that the system operates efficiently while maintaining flexibility for state management and transitions.

Part 2

Down_counter_4bit




```
module down_counter_4bit (
    input wire clk,
    input wire reset,
    input wire [0:3] load_data,
    input wire load_en,
    output reg [0:3] count,
    output reg load_flag
);

always @(posedge clk or posedge reset)
begin
    if (reset)
        count <= load_data;
    else if (load_en)
    begin
        load_flag <= 1'b1;
    end
    else if (load_flag == 1'b1)
        count <= count - 1;
    else if (count != 0)
        count <= count - 1;
    else
        load_flag <= 1'b0;
end

endmodule
```

The **down_counter_4bit** that describes a 4-bit synchronous down counter with a load input. The counter has an input clock signal clk, an input reset signal reset, a 4-bit input signal load_data, an input enable signal load_en, and output signals count and load_flag.

The count signal is a 4-bit register that holds the current count value. The load_flag signal is a 1-bit register that indicates when new data has been loaded into the counter.

The module uses a single always block with a sensitivity list that includes the clk and reset signals. This means that the block is sensitive to both the rising edge of the clock signal and the rising edge of the reset signal.

The always block contains an if statement with four conditions. The first condition checks if the reset signal is high. If so, the count register is loaded with the value of the load_data input. The second condition checks if the load_en signal is high. If so, the load_flag register is set to 1, indicating that new data has been loaded into the counter. The third condition checks if the load_flag register is high. If so, the count register is decremented by 1. The fourth condition checks if the count register is not equal to 0. If not, the count register is decremented by 1. If the count register is equal to 0, the load_flag register is set to 0, indicating that the counter has reached its minimum value.

Overall, the down_counter_4bit module is a synchronous 4-bit down counter that can be loaded with a new value when the load_en signal is high. The reset signal can be used to reset the counter to a specific value. The load_flag signal can be used to indicate when new data has been loaded into the counter.

Register File

This is a schematic diagram of a digital circuit that uses four D flip-flops (DFFs) to store and shift a 4-bit binary value. The circuit has a single input clock signal (CLK) and four input data signals (in[0..3]). The output of the circuit is a 4-bit binary value (Q[0..3]).

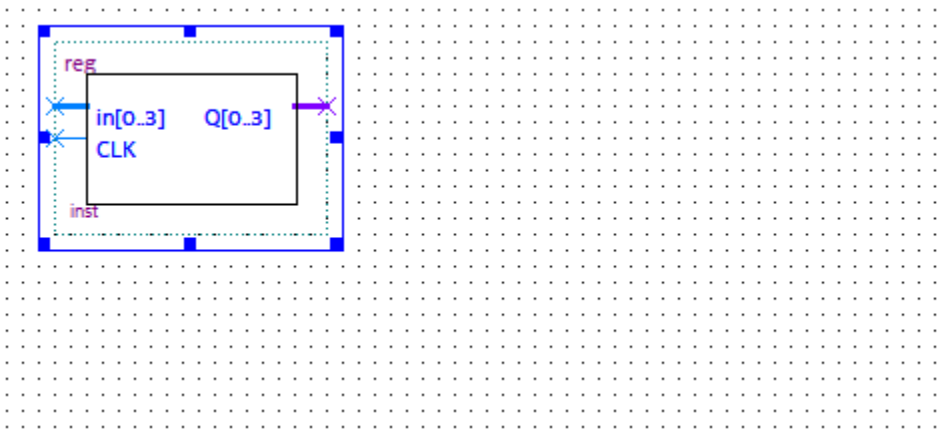
The circuit consists of four DFFs, each with a clock input (CLK), a data input (D), a clear input (CLRN), and a preset input (PRN). The CLRN input is used to clear the DFF, while

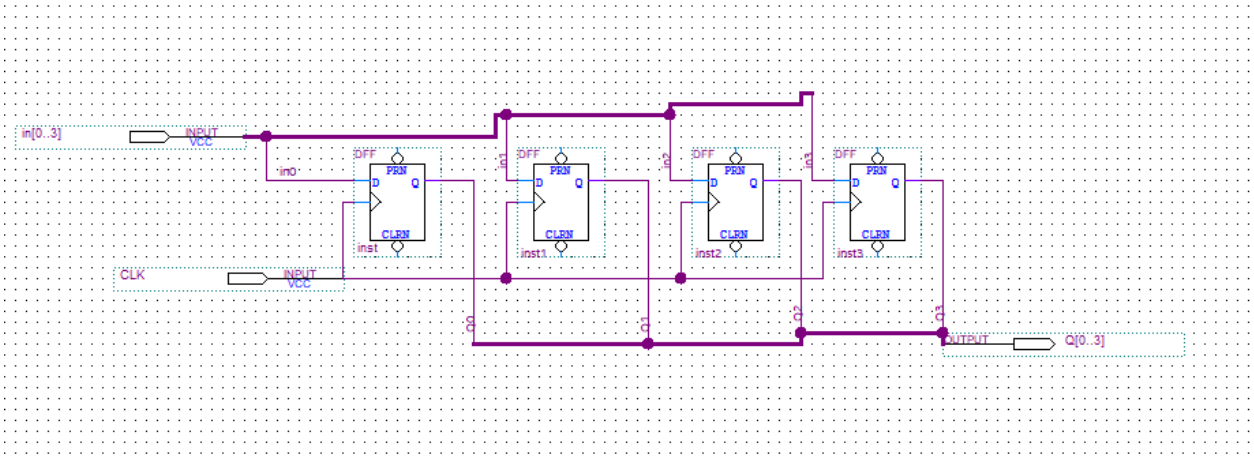
the PRN input is used to preset the DFF to a specific value. The D input is used to store the input data, while the Q output is used to output the stored data.

The four DFFs are connected in series, with the output of each DFF connected to the input of the next DFF. The first DFF (inst) is connected to the input data signals (in[0..3]), while the last DFF (inst3) is connected to the output signals (Q[0..3]). The CLK input of each DFF is connected to the CLK input of the circuit.

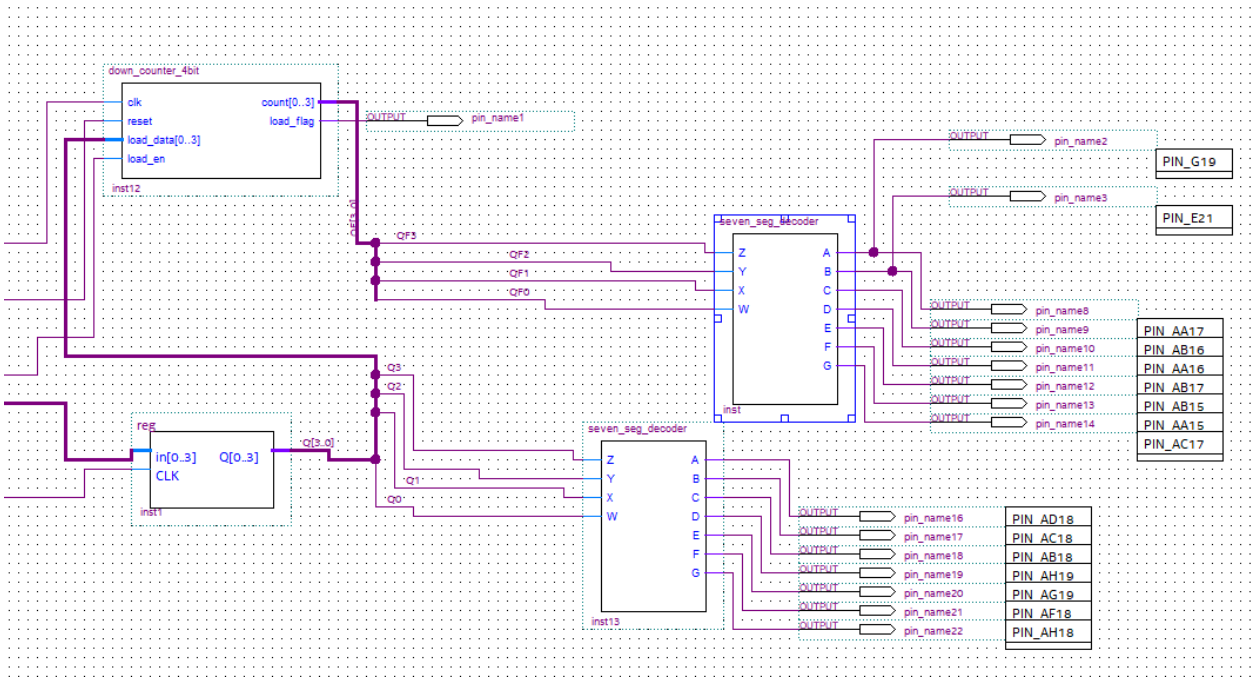
The circuit also includes several connectors that are used to connect the input and output signals to other parts of the circuit. The connectors are labeled with aliases that are used to identify them in the schematic diagram.

Overall, this circuit is a simple 4-bit shift register that can be used to store and shift a 4-bit binary value. The circuit can be clocked using the CLK input, and the input data can be loaded into the circuit using the input data signals (in[0..3]). The output of the circuit is a 4-bit binary value (Q[0..3]) that can be used for further processing.





Part 3



Part 3 was mostly devoted to displaying the enter code to the seven-segment displays. The **down counter** and **reg** made two bus lines. The segments then were attached to the each of the bus lines. The **down_counter** takes one-bit from the register and allows

it to reset and load in the specified number by the inputs of the side switches. Additionally, the **reg** has a bus line containing the user input stored in the register which would be displayed on the seven-segment display which can reset and load into the main display(the out of the way outputs are just led for special effects, to jazz it up a little).

Seven-Segment decoder

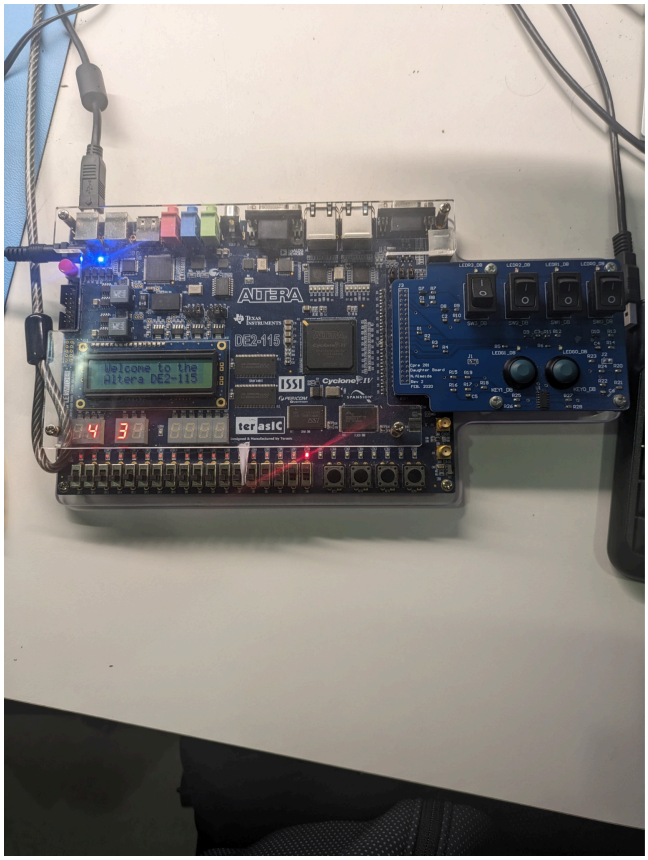
This module for a seven-segment decoder, which is a combinational logic circuit that converts a 4-bit binary input into a seven-segment output to display a decimal digit. The module has seven output signals (A, B, C, D, E, F, G) and four input signals (Z, Y, X, W).

The module uses a case statement to assign the output signals based on the input signals. The case statement uses a 4-bit vector (ZYXW) as the input, which is created using the concatenation operator ({}). The case statement has 16 cases, one for each possible value of the input vector. Each case assigns a 7-bit vector to the output signals, which corresponds to a specific decimal digit.The input signals (Z, Y, X, W) are the binary representation of the decimal digit to be displayed. The input signals are connected to the four least significant bits of a 4-bit binary counter, which counts from 0 to 15. The output signals of the seven-segment decoder are connected to the seven

segments of a seven-segment display. This is the screenshots of both displays and it outputting on the board

```
module seven_seg_decoder(Z, Y, X, W, A, B, C, D, E, F, G);
   input Z, Y, X, W;
   output reg A, B, C, D, E, F, G;

   always @(Z or Y or X or W)
   begin
      case({Z, Y, X, W})
         //truth tabling
         4'b0000: {A, B, C, D, E, F, G} = 7'b0000001;
         4'b0001: {A, B, C, D, E, F, G} = 7'b1001111;
         4'b0010: {A, B, C, D, E, F, G} = 7'b0010010;
         4'b0011: {A, B, C, D, E, F, G} = 7'b0000110;
         4'b0100: {A, B, C, D, E, F, G} = 7'b1001100;
         4'b0101: {A, B, C, D, E, F, G} = 7'b0100100;
         4'b0110: {A, B, C, D, E, F, G} = 7'b0100000;
         4'b0111: {A, B, C, D, E, F, G} = 7'b0001111;
         4'b1000: {A, B, C, D, E, F, G} = 7'b0000000;
         4'b1001: {A, B, C, D, E, F, G} = 7'b0000100;
         4'b1010: {A, B, C, D, E, F, G} = 7'b0001000;
         4'b1011: {A, B, C, D, E, F, G} = 7'b1100000;
         4'b1100: {A, B, C, D, E, F, G} = 7'b0110001;
         4'b1101: {A, B, C, D, E, F, G} = 7'b1000010;
         4'b1110: {A, B, C, D, E, F, G} = 7'b0110000;
         4'b1111: {A, B, C, D, E, F, G} = 7'b0111000;
      endcase
   end

endmodule
```

```
module seven_seg_decoder(Z, Y, X, W, A, B, C, D, E, F, G);
   input Z, Y, X, W;
   output reg A, B, C, D, E, F, G;

   always @(Z or Y or X or W)
   begin
      case({Z, Y, X, W})
         //truth tabling
         4'b0000: {A, B, C, D, E, F, G} = 7'b0000001;
         4'b0001: {A, B, C, D, E, F, G} = 7'b1001111;
         4'b0010: {A, B, C, D, E, F, G} = 7'b0010010;
         4'b0011: {A, B, C, D, E, F, G} = 7'b0000110;
         4'b0100: {A, B, C, D, E, F, G} = 7'b1001100;
         4'b0101: {A, B, C, D, E, F, G} = 7'b0100100;
         4'b0110: {A, B, C, D, E, F, G} = 7'b0100000;
         4'b0111: {A, B, C, D, E, F, G} = 7'b0001111;
         4'b1000: {A, B, C, D, E, F, G} = 7'b0000000;
         4'b1001: {A, B, C, D, E, F, G} = 7'b0000100;
         4'b1010: {A, B, C, D, E, F, G} = 7'b0001000;
         4'b1011: {A, B, C, D, E, F, G} = 7'b1100000;
         4'b1100: {A, B, C, D, E, F, G} = 7'b0110001;
         4'b1101: {A, B, C, D, E, F, G} = 7'b1000010;
         4'b1110: {A, B, C, D, E, F, G} = 7'b0110000;
         4'b1111: {A, B, C, D, E, F, G} = 7'b0111000;
      endcase
   end

endmodule
```


ALTERA
DE2-115
Welcome to the
Altera DE2-115
terasIC