EECE 229 Fall 2016

Assignment #3

Tim Hamling
Partner: Michelle Blum
December 5, 2016

Overview

For this assignment, we were asked to look at different flip-flops and latches, and construct them using Quartus II. The circuits ranged from a SR latch in Question 1, a gated SR flip flop in Question 2, a D flip flop in Question 3, and a Master-Slave Edge triggered flip flop in Question 4. In addition to creating the circuits, we also constructed Verilog behavioral modules for the circuits in Questions 3 and 4.

In Question 5, we again implemented the D flip-flop to create a counter. We encountered problems due to switch bouncing, but were able to prevent that using a clock_divider module, as well as using the CLOCK2_50 input signal as our input. The counter output was connected to a HEX display, which allowed us to see the results as it counted.

In Question 6, we looked back at one of the Moore Machines we created in Assignment 3 Preparation, and implemented a Verilog module that represented it. We then tested it by running it on the Altera board and used switches and red LEDs to see the results. In addition, we used Green LEDs to display the state that the circuit was in so that we could better visualize what was happening.

Finally, in Question 7, we looked at a Moore Machine diagram, and used a template to implement it. The interesting thing about this circuit is that the inputs do not matter. They are continuous, and loop back to another state at the completion of one state. We added in a reset input so that we could control this continuous machine, and connected the output to an LED. In addition, we added in a counter that incremented whenever the circuit advanced from state D to state F, which we then hooked up to a HEX display so that we could see the counter running.

Question 1

We were given the circuit in **Figure 1** and asked to construct and test it. We decided to create it using a block diagram file, shown in **Figure 2**. Creating it was simple because we just had to copy the gates and inputs. We used two slide switches to represent S_n and R_n, and two LEDs to represent Q and QN.

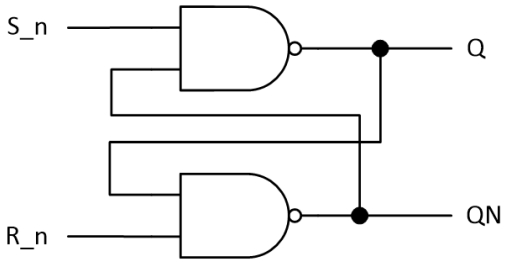


Figure 1 - Circuit for Question 1

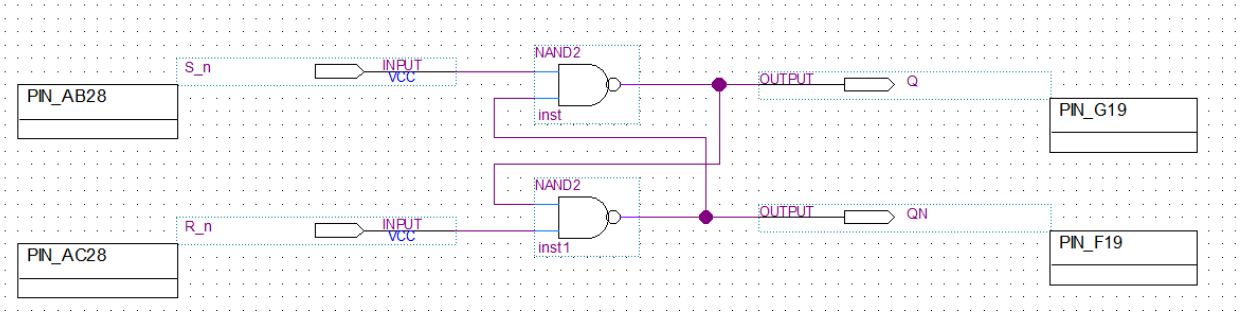


Figure 2 - BDF for Question 1

We then compiled and ran our file, and tested it using the Altera board. We then used our truth tables from the Assignment 3 Prep document to verify that all our output and input sequences were correct.

The RTL file for this circuit is shown in **Figure 3** below, and the Pin Assignment window and Compilation window are shown to the right in **Figure 4** and **Figure 5** respectively.

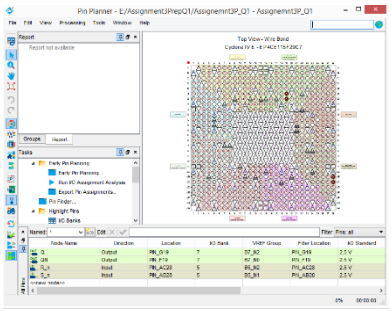


Figure 4 – Pin Assignments for Question 1

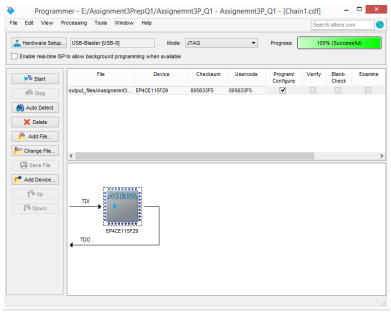


Figure 5 – Compilation Report for Question 1

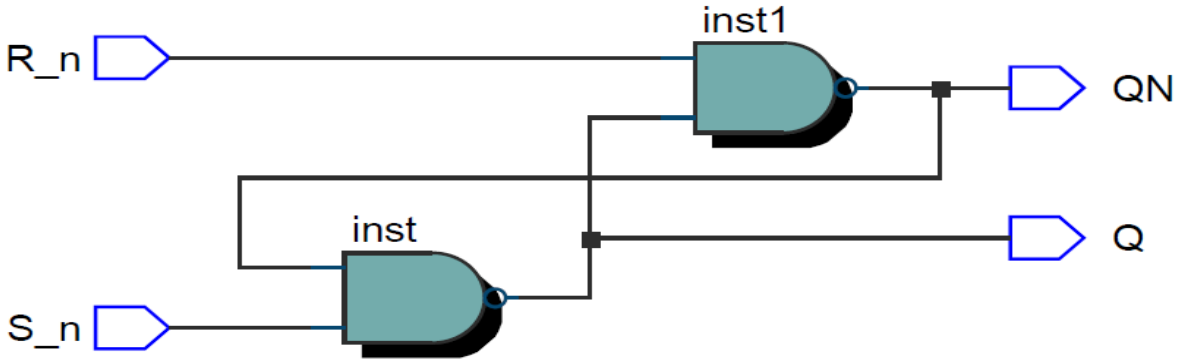


Figure 3 – RTL diagram for Question 1

Question 2

This question involved the circuit shown in **Figure 6**. Similar to the previous problem, we had to construct and test this circuit using Quartus II. We used another BDF file, shown below in **Figure 7**, and were able to modify the BDF used in **Figure 2** above to achieve it.

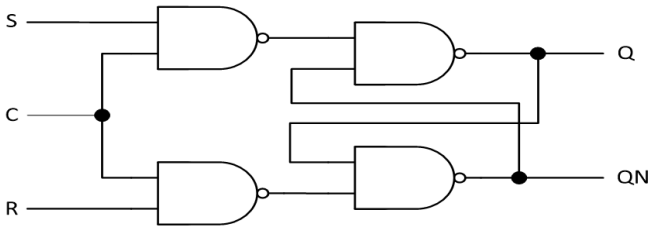


Figure 6 - Circuit for Question 2

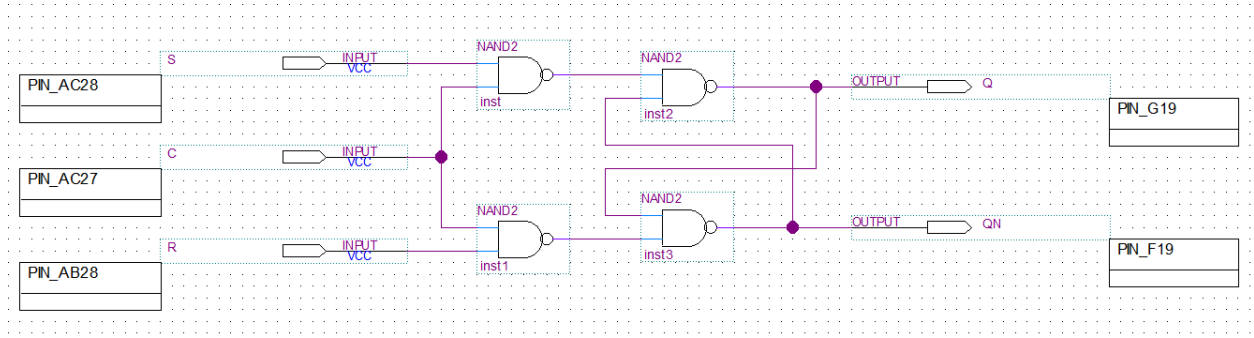


Figure 7 - BDF for Question 2

We used switches for the three inputs, and LEDs for the two outputs. We found that this circuit produced a change in the output only when the switch representing the clock was on. When this happened, Q followed the value of S and QN followed the value of R, which matched our findings in the truth table for a similar problem in Assignment 3 Preparation, showing that our circuit was indeed correct.

Our resulting RTL file is shown in **Figure 8** below. The Pin Assignments and Compilation window are shown in **Figure 9** and **Figure 10** below, respectively.

Figure 9 – Pin Assignments for Question 2

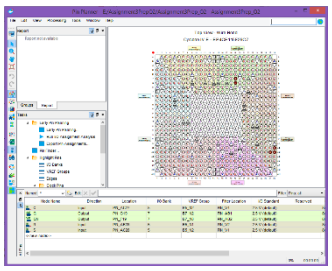
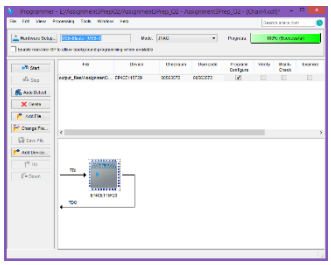


Figure 10 – Compilation Report for Question 2



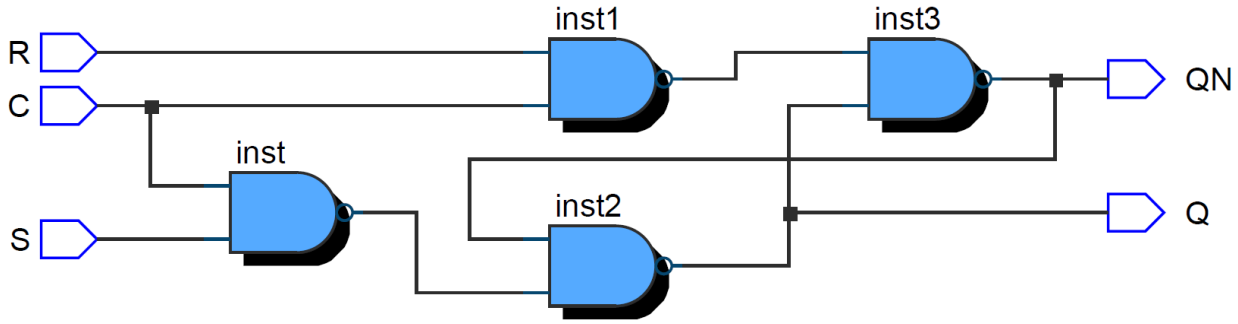


Figure 8 – RTL diagram for Question 2

Question 3

This question dealt with the circuit shown in **Figure 11**. We, again, had to implement this circuit in Quartus II and test it. In addition to this, we had to write a Verilog behavioral module that functioned identically to the circuit. We started by creating the circuit using a block diagram file, shown in **Figure 12** below.

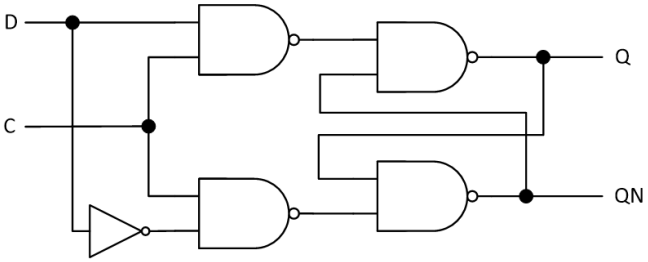


Figure 11 - Circuit for Question 3

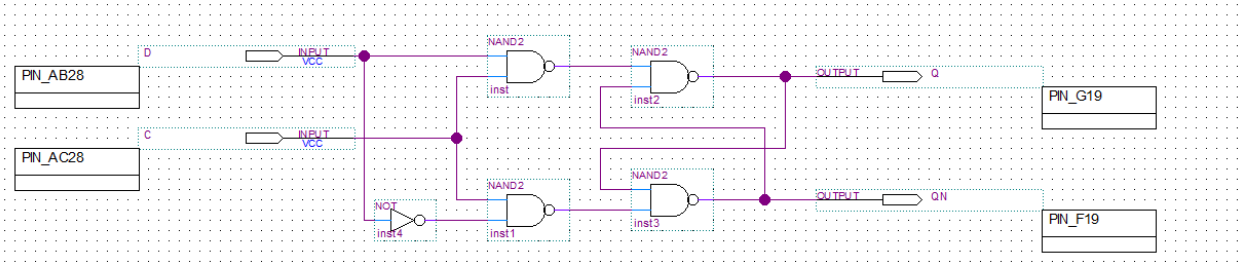


Figure 12 - BDF for Question 3

This circuit is a D Flip flop, and is controlled with a clock input. It functions very similarly to the SR latch in Question 2. To test it, we used two switches for the inputs and two LEDs for the outputs. It matched our truth table for the preparation: Q turned on only when D and C were flipped on, and QN was lit up when D was off and C was on.

From this, we moved on and constructed a Verilog module to represent the circuit. Using the description from the paragraph above, we could make a behavioral module, as shown in **Code Block 1** to the left. The outputs only change when the Clk input is on, and match up with the input of D.

```
module d_latch(D, Clk, Q, Qn)
input wire D;
input wire Clk;
output reg Q, Qn;

always @(posedge Clk)
begin
    if(Clk)
    begin
        Q = D;
       Qn = ~D;
    end
end
endmodule
```

Code Block 1 – Verilog module representing the circuit in Fig. 3

With the Verilog code implemented, we decided to test it again using the Altera board. We ran the module using the BDF shown below in **Figure 13**, and found that our results matched up identically to what we discovered when we ran the BDF in **Figure 12**.

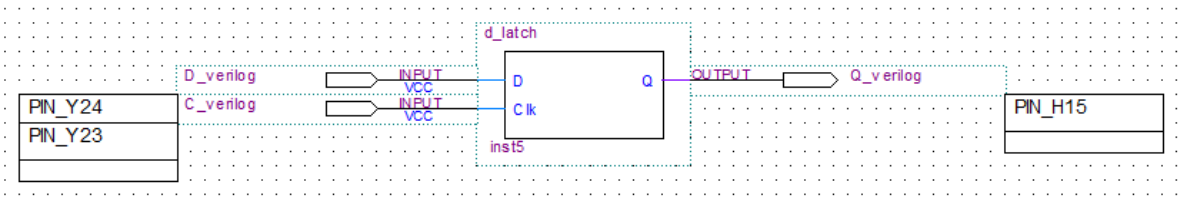


Figure 13 – BDF for the Verilog Module of Question 3

Shown in the figures on this page are all the RTL files, pin assignments, and compilation windows of this circuit. The RTL file for the circuit in **Figure 12** is shown in **Figure 14**, and the RTL file for the Verilog module is shown in **Figure 15**.

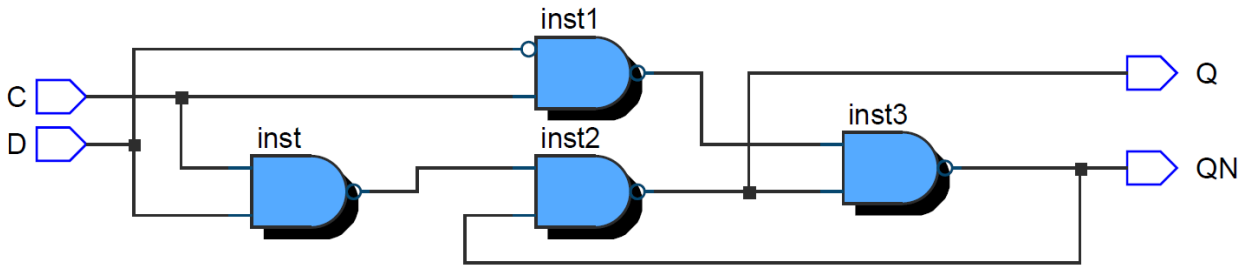


Figure 14 – RTL file representing the circuit of Question 3

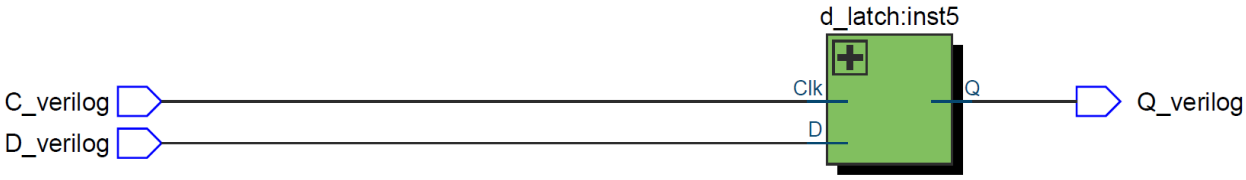


Figure 15 – RTL file representing the Verilog module of Question 3

The Pin Assignment window for this circuit is shown in **Figure 16** and the Compilation Report is shown in **Figure 17** below.

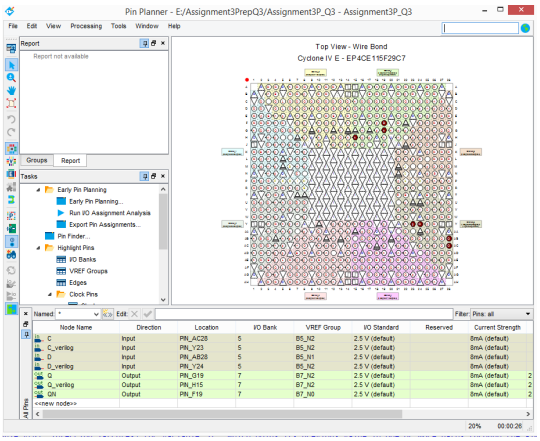


Figure 16 – Pin Assignments for Question 3

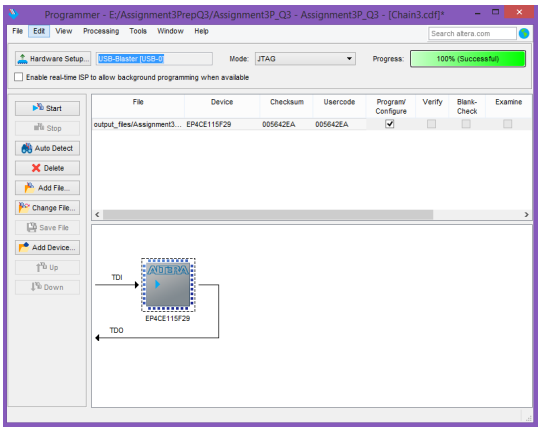


Figure 17 – Compilation Report for Question 3

Question 4

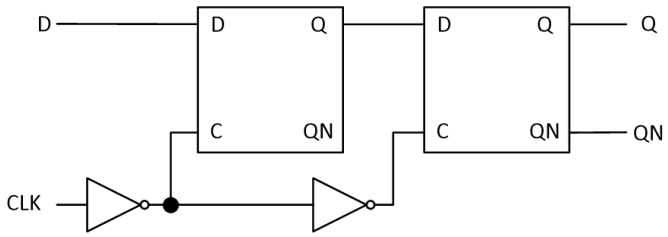


Figure 18 - Circuit for Question 4

This question involves the same module used in Question 3. The circuit is shown to the right in **Figure 18**. We had to create a circuit that incorporated the module created in **Code Block 1** twice, and link inputs and outputs to it. We used two switches for the two inputs, and two LEDs for the two outputs. The output of one of the modules was fed into the input of the other. The two modules were also controlled with one Clock input, with one receiving the opposite input as the other.

Because of the unique nature of the Clock inputs, this circuit is known as a Master-Slave flip-flop. When the input from the clock is 0, the first flip-flop is activated. When the input is 1, the second flip-flop is used.

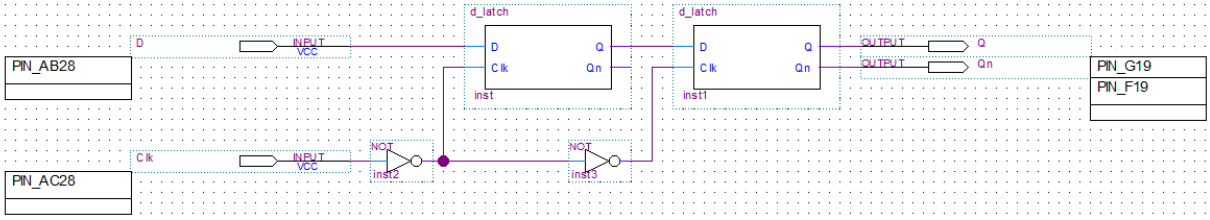


Figure 19 - BDF for Question 4

We copied the Verilog code from **Code Block 1** to create new modules, and placed two of them into a new block diagram file. Next, we added the inputs and outputs, as well as the negation gates. The diagram for this circuit is shown above in **Figure 19**. Next, we created a function table for the circuit based on our testing of it, the results of which are shown in **Table 1** below.

D	Clk	Q	QN
0	↑	0	1
0	↑	1	0
x	0	Last Q	Last QN
x	1	Last Q	Last QN

Table 1 – Function/Truth table for Question 4

```
module master_slave(D, Clk, Q, Qn)
input wire D;
input wire Clk;
output reg Q, Qn;

always @(*)
begin
    if(Clk)
    begin
        Q <= D;
       Qn <= ~D;
    end
end
endmodule
```

Code Block 2 – Verilog Code that represents the circuit for Question 4

One thing to note is that the circuit only changes the values of its outputs when the Clock changes from 0 to 1, which means that this is a Positive Edge Triggered flip-flop. Unlike the previous flip-flops, this one still works even when the Clock is 0 because it is a Master-Slave flip flop. This means that when the input is a high state, the rightmost flip-flop is activated. When the input is at a low state, the leftmost flip-flop is activated. No matter what state the clock is in, it will always activate one or the other

flip-flop. Still, the circuit will only change when the clock turns from 0 to 1 because of how the D flip-flop works.

Next, we had to create a single Verilog module that represented the function table. It is almost identical to the code used for problem 3. Our module is shown in **Code Block 2** above.

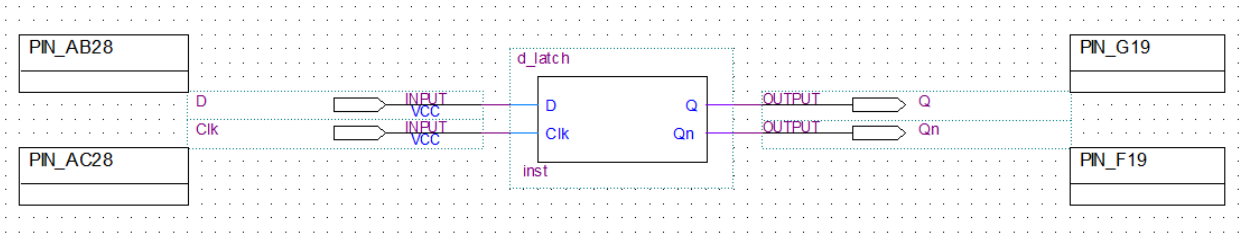


Figure 20 - BDF for the Verilog module used for Question 4

We then tested this code using another block diagram file, shown above in **Figure 20**. It uses similar inputs and outputs as the original circuit, and functions identically. Our RTL diagrams for both BDF files are shown below. The RTL for **Figure 19** is shown in **Figure 21**, and the RTL file for **Figure 20** is shown in **Figure 22** below.

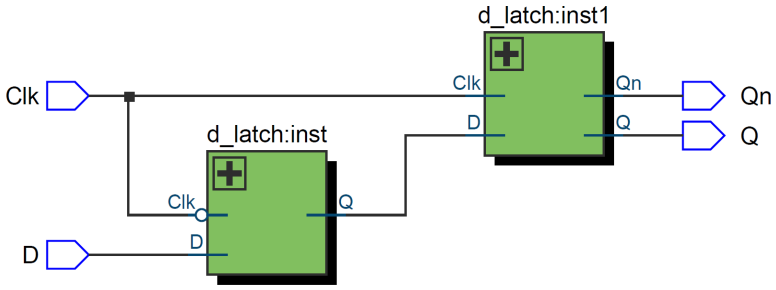


Figure 21 – RTL file used to represent the circuit for Question 4

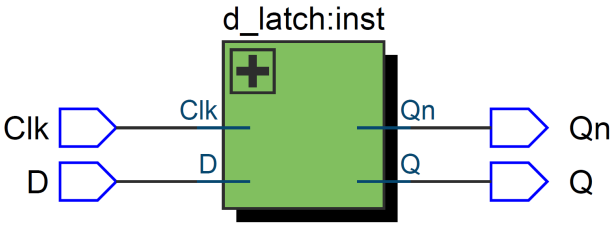


Figure 22 – RTL file used to represent the Verilog circuit representation of Question 4

Finally, we included our Pin Assignment window and Compilation Report window for the circuit; these are shown in **Figure 23** and **Figure 24** below.

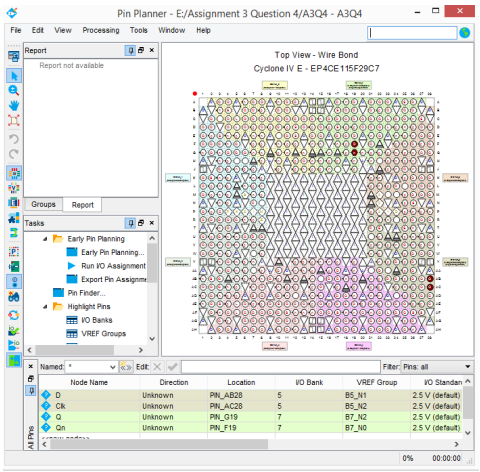


Figure 23 – Pin Assignments for Question 4

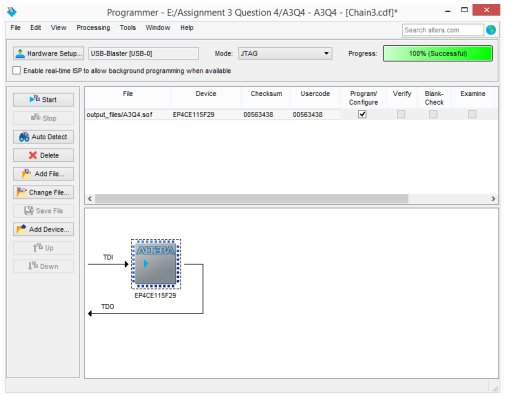


Figure 24 – Compilation Report for Question 4

Question 5

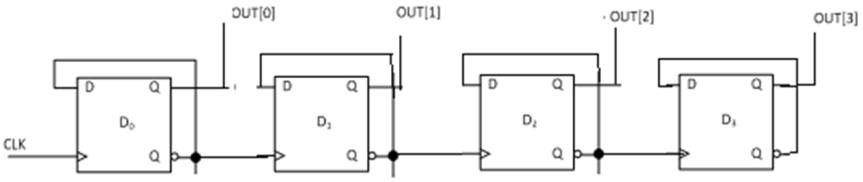


Figure 25 - Circuit for Question 5

This question had us use the master-slave D flip flop from Question 4, and use it four times in series to create a counter of sorts. The layout of this circuit is shown in **Figure 25**. We started by copying the Verilog code used in the previous question, and using it to create the four modules, D_0 D_1 D_2 D_3. Next, we connected the wires, as shown in the diagram, and attached inputs and outputs. For the input, we decided to use a Pushbutton, and for the outputs, we connected them to a bus and ran it into a 7 Segment decoder module, as used in Assignment 2. Our block diagram file for this is shown in **Figure 26**.

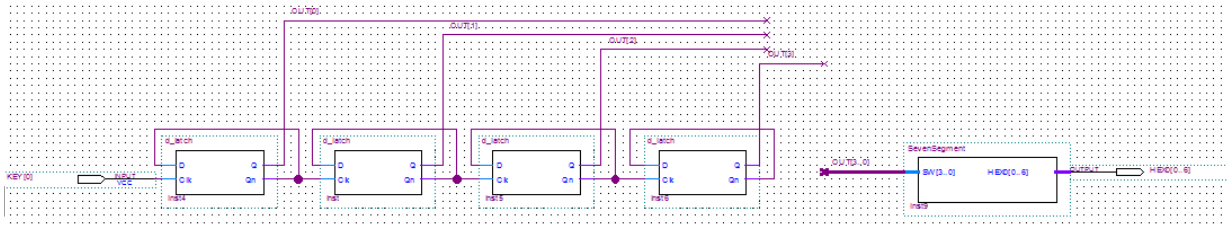


Figure 26 - BDF for Question 5

When we compiled and ran the circuit, we saw that it worked by incrementing the number displayed on the screen each time the button was pressed. It started at 0 and worked up to HEX F, before jumping back down to 0. However, sometimes the number displayed would jump two numbers, and sometimes the display screen flickered or dimmed. This is known as switch bouncing, and happened because the connections within the button do not make full contact instantaneously when pressed. This can cause the circuit to complete itself twice with one press, which caused the displayed value to jump. If we had used the slide switches, then the bouncing effect would not have happened because the connection is always made once, securely, with a switch.

To fix this problem, we used the provided clock_divider module. This fixed the issue by preventing the button from skipping a press and jumping a value. In addition, we connected the input to the CLOCK2_50 input on the Altera board, which resulted in a self-counting counter. Our RTL files for the circuit without and with the clock_divider are shown in **Figure 27** and **Figure 28** to the right.

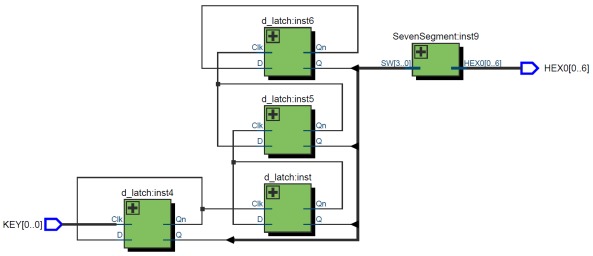
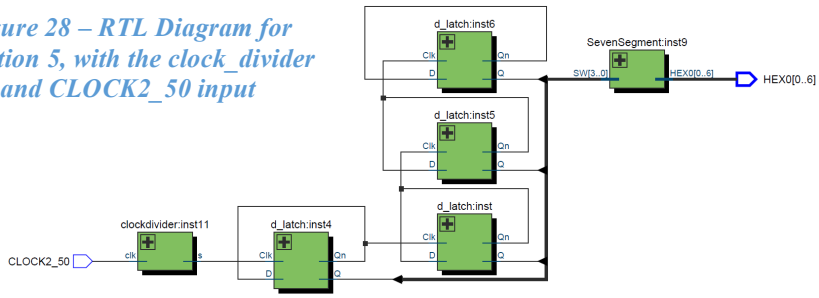


Figure 27 – RTL Diagram for Question 5, without the clock_divider

Figure 28 – RTL Diagram for Question 5, with the clock_divider and CLOCK2_50 input



To finish up this problem, we had to write a Verilog module that behaves identically to the circuit shown above. Coming up with the behavioral aspect of the code was easy because all we had to do was increment the output by one each time the clock input was at a value of 1. Our code is shown in **Code Block 3**. It is worth pointing out that the output will be a 4-bit number, meaning once it has the value 4b'1111, it will automatically roll over to become 4b'0000.

```
module counter(Clk, out)
input wire Clk;
output reg [3:0] out;

always @(posedge Clk)
begin
    out <= out + 1;
end
endmodule
```

Code Block 3 – A single Verilog module representing the full circuit of Question 5

Our block diagram file for this circuit is shown below in **Figure 29**, and the resulting RTL file is shown separately in **Figure 30**.

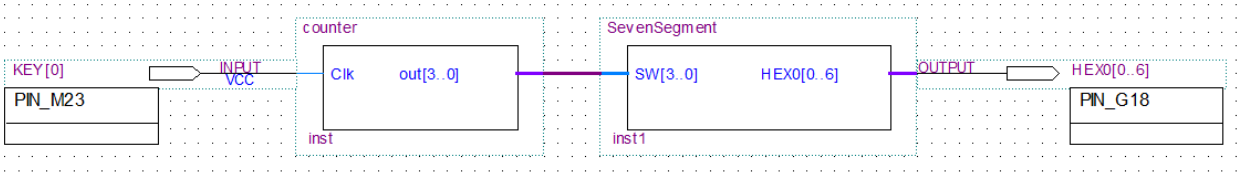


Figure 29 – BDF Diagram for Question 5, using a single Verilog counter module

Figure 30 – RTL Diagram for Question 5, using a single Verilog counter module

Question 6

For this question, we had to implement the Moore Machine circuit given in Assignment 3 Prep. The diagram for the circuit is shown in **Figure 31**. We had to implement our Verilog code and test the results using the Altera board.

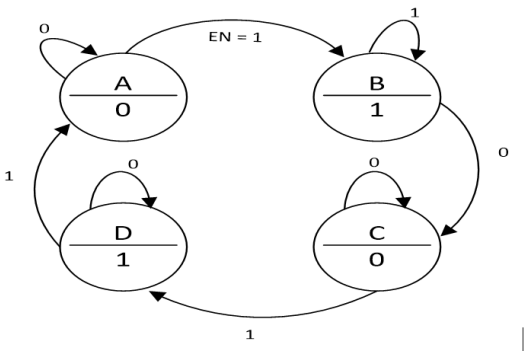


Figure 31 - Circuit for Question 6

```
module state(in, Clk, Q1,
                              Q0, out)
input wire in, Clk;
output reg Q0, Q1
output wire out;

always @(posedge Clk)
begin
    Q1 <= (~Q0 && Q1) || (Q0 && ~in);
    Q0 <= (~Q0 && in) || (~Q1 && in)
            || (Q0 && Q1 && ~in);
end
endmodule
```

Code Block 4 – A Verilog module representing the circuit of Question 6

The Verilog code we used is shown in **Code Block 4**. We then connected inputs and outputs to test it. For the BDF diagram shown in **Figure 32**, it is worth pointing out that the outputs Q0 and Q1 are connected to Green LEDs that are used to tell us what state the machine is in when we run it on the Altera board. The output is connected to a single Red LED, and the inputs are controlled with slide switches.

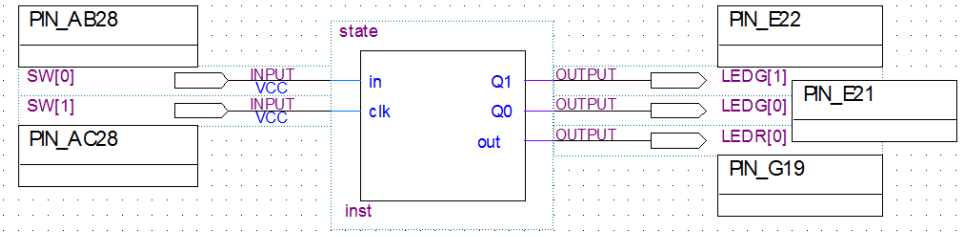


Figure 32 – BDF for Question 6

When we tested our Verilog module on the Altera board, we found that it matched the diagram perfectly. State A was represented as 00 binary, so when it was in that state, the output was 0. State B was represented as 01 binary, so when it was in that state, the output was 1. So on with States C and D. We used the switch representing the Clock to advance the circuit to the next state, and the input switch to send the input signal. When we tested it, it advanced in accordance to the required inputs, and produced the correct output.

Our RTL file for this circuit is shown in **Figure 33** to the right.

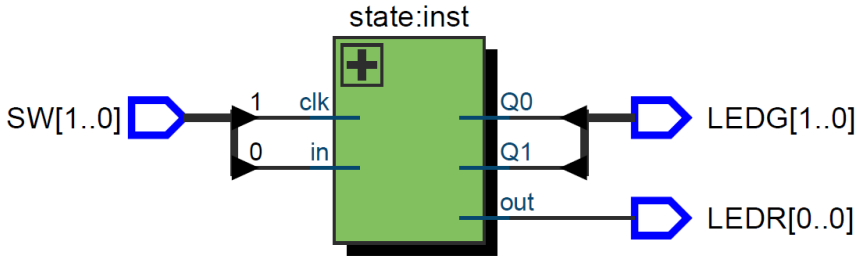


Figure 33 – RTL diagram for Question 6

Question 7

This question was labeled as Question 10 on the assignment. We were given the circuit in **Figure 34** to the right, and asked to make a behavioral module representing it.

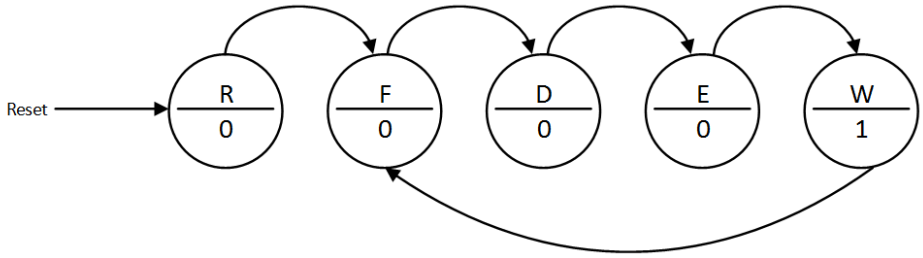


Figure 34 – Circuit Diagram for Question 7

Since there are no inputs shown on the diagram, this means that the circuit simply progresses from one state to the next with no need for inputs. In order to help visualize it, our circuit used Green LEDs to let us know which state the machine is on (000 to 100 in binary). We also had to add a reset flag in that brought the circuit to the starting state whenever it was hit. We decided to use a Push Button for this. So, whenever the button was pressed and the clock was instructed to move to the next state, the circuit would reset to the starting state.

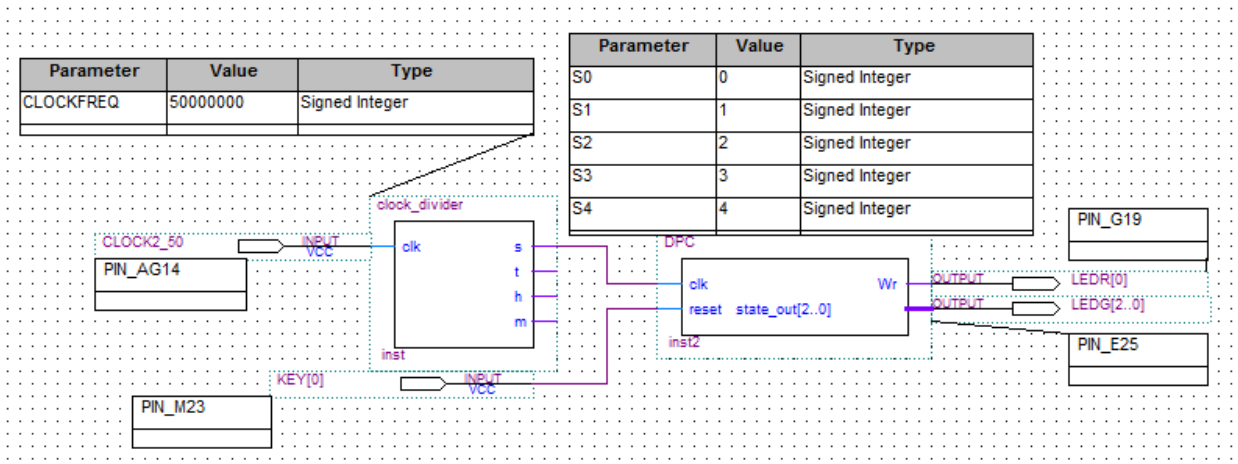


Figure 35 – BDF Diagram for Question 7

The BDF of this circuit is shown above in **Figure 35**. We used a CLOCK2_50 as the clock input, and added the clock_divider module to work with it. We then used a Push Button as the reset value, and projected the state in a binary form using Green LEDs. The output was shown on a single Red LED because it would either be 0 for off, or 1 for on.

Our circuit worked perfectly. Each time the clock progressed to the positive edge, it advanced to the next state. When it was at the final state, State W, it outputted a 1 value, then regressed back to State F. And when we hit the reset button, it went to the first state, State R. The RTL Diagram for this circuit is shown above in **Figure 36**. The Verilog behavioral module for this question is shown on the next page in **Code Block 5**.

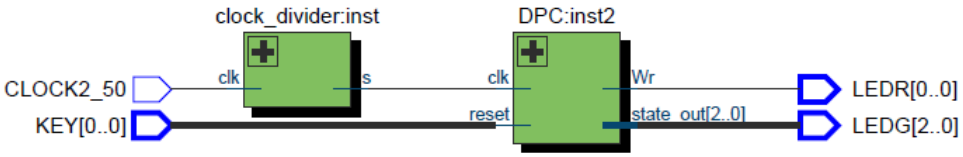


Figure 36 – BDF Diagram for Question 7

```
module DPC
(
    input clk, reset,
    output reg Wr,
    output reg [2:0] state_out
);
    reg [2:0]state; // Declare state register

    // Declare states
    parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4;

    // Output depends only on the state
    always @ (state) begin
        case (state)
            S0: Wr = 0;
            S1: Wr = 0;
            S2: Wr = 0;
            S3: Wr = 0;
            S4: Wr = 1;
            default: Wr = 0;
        endcase
    end

    // Determine the next state
    always @ (posedge clk)
      begin
        if (~reset) begin
           state <= S0;
           state_out <= S0;
        end
        else
            case (state)
                S0: begin
                  state <= S1;
                  state_out <= S1;
                    end
                S1: begin
                  state <= S2;
                  state_out <= S2;
                    end
                S2: begin
                  state <= S3;
                  state_out <= S3;
                    end
                S3: begin
                  state <= S4;
                  state_out <= S4;
                    end
                S4: begin
                  state <= S1;
                  state_out <= S1;
                    end
            endcase
      end
endmodule
```

Our Verilog code uses two always() blocks, one to determine the output of the machine depending on which state it is in, and one to get the next state and the state output assignments.

Code Block 5 – Verilog module representing the circuit for Question 7

The second part of this question asked us to add a counter that counts how many times the circuit has looped. We could then use this counter and output it to a HEX display to show how many times the circuit progressed through its states. For this instance, we had to count the number of times the circuit went from State F to State D. This counter would be outputted as an 8-bit wire so that it could be read to the HEX displays. It had to match the provided module given in **Figure 37**.

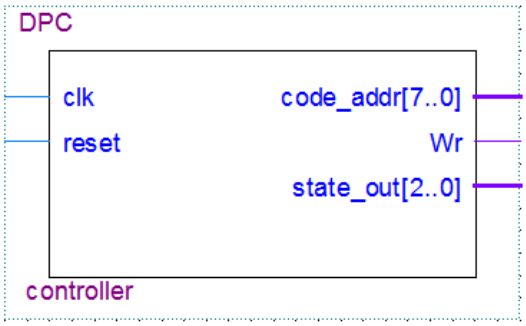
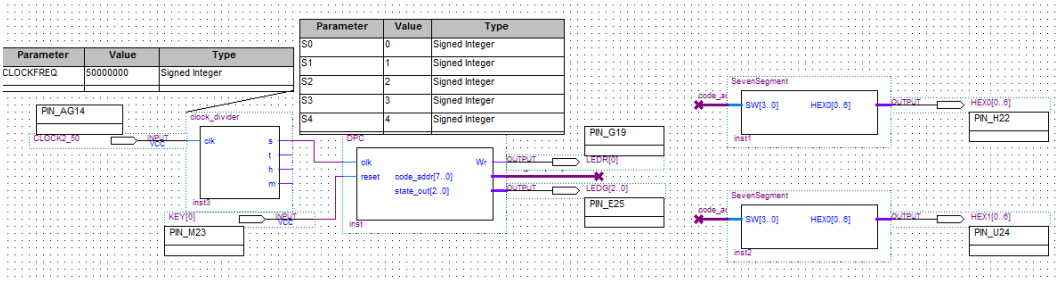


Figure 37 – Module needed for Question 7

Figure 38 – BDF Diagram for the revision of Question 7



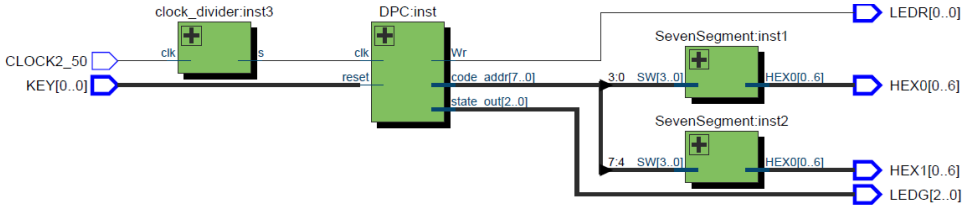


Figure 39 – RTL Diagram for the revision of Question 7

Our resulting BDF file is shown above in **Figure 38**, and the RTL file created from this BDF file is shown in **Figure 39**.

The code for the clock_divider was provided to us, and the code for the 7 Segment Display was reused from Assignment 2. When we connected it to the Altera board, it worked just as intended. The counter incremented every time the state changed from State F to State D, and the resulting counter was shown across two HEX Displays. The Reset button also allowed us to stop and reset the machine. Our full Verilog code for this updated module is shown on the next page **Code Block 6**.

```
module DPC(
    input clk, reset,
    output reg Wr,
    output wire [7:0] code_addr,
    output reg [2:0] state_out);

    reg [2:0]state;
    reg [7:0] PC;

    parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4;

    always @ (state) begin
        case (state)
            S0: Wr = 0;
            S1: Wr = 0;
            S2: Wr = 0;
            S3: Wr = 0;
            S4: Wr = 1;
            default: Wr = 0;
        endcase
    end

    always @ (posedge clk) begin
        if (~reset) begin
            state <= S0;
            state_out <= S0;
            PC[7:0] = 0;
        end
        else
            case (state)
                S0:begin
                    state <= S1;
                    state_out <= S1;
                  end
                S1:begin
                    state <= S2;
                    PC = PC + 1;
                    state_out <= S2;
                  end
                S2:begin
                    state <= S3;
                    state_out <= S3;
                  end
                S3:begin
                    state <= S4;
                    state_out <= S4;
                  end
                S4:begin
                    state <= S1;
                    state_out <= S1;
                  end
            endcase
    end

    assign code_addr[7:0] = PC[7:0]; //Connect to 2 hex digits using bus

endmodule
```

Code Block 6 – Verilog module representing the redesigned counting circuit for Question 7