Experiment #2 – Frequency Regulation

Student Name: Marzieh Bagheri Nia Amiri
Student ID: 810197682

First Part: Design FPGA

1. In this part, we should write a Verilog code for FPGA.
To do this, according to the Lab instructions, we divide the whole process into three sections, which are as follows:

a) Measure input frequency

In this section, we must provide a method which can be used for calculating the input frequency. The following code has been prepared for this purpose.

```
always @(posedge clk, posedge rst) begin
   if (rst)
      prePSI <= 1'b0;
   else
    prePSI <= PSI;
end

always @(posedge clk) begin
   case ({prePSI, PSI})
        2'b00: duration <= duration;
        2'b01: duration <= 8'b0;
        2'b10: duration <= duration;
        2'b11: duration <= duration + 1;
   endcase
end
```

b) Comparision

In this section, we must provide a method for comparing the calculated input frequency with the desired frequency. The following code has been prepared for this purpose.

```
always @(posedge clk) begin
   if ({prePSI, PSI} == 2'b10)
       enable <= 1'b1;
   else
       if ({prePSI, PSI} == 2'b01)
           enable <= 1'b0;
end

always @(enable, setPeriod, duration) begin
        {increment, decrement, equal} = 3'b0;
        if (enable)
            if (setPeriod < duration)
                increment = 1'b1;
            else
                if (setPeriod > duration)
                    decrement = 1'b1;
                else
                    if (setPeriod == duration)
                        equal = 1'b1;
end
```

c) Set the counter Divider input

In this section, after the comparison result is determined, we must adjust the FPGA output (counter divider input) by performing the appropriate operations (addition or subtraction). The following code has been prepared for this purpose.

```
always @(enable, calcOut_2) begin
        if (~enable)
           calcOut_1 = 1'b0;
        else
            if (calcOut_2 == 1'b1)
                calcOut_1 = 1'b1;
end

always @(posedge clk, posedge rst) begin
        if (rst)
            adjustedDiv <= 8'b01111111;
        else
           if (enable & ~calcOut_1) begin
               if (increment) begin
                   adjustedDiv <= adjustedDiv + 1;
                   calcOut_2 <= 1'b1;
               end
               else begin
                   if (decrement) begin
                       adjustedDiv <= adjustedDiv - 1;
                       calcOut_2 <= 1'b1;
                   end
               end
           end
           else begin
               if (~enable)
                   calcOut_2 <= 1'b0;
           end
end
```

The general code is formed by combining these three sections which exists in LAB2/ FPGA/Modelsim folder.

2. For this part, I build a project in Quartus and Synthesize the above code in it.

3. I create a symbol from second part code. The block diagram of counter driver can be seen below:

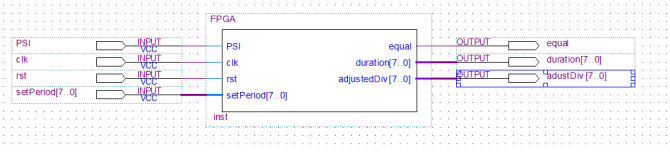

Figure 1

● A clearer picture of Figure1 exists in LAB2/FPGA/Quartus folder.

All the above project exists in LAB2/ FPGA/Quartus.

4. I create a new project in Quartus and implement Counter divider in it. The block diagram of counter driver can be seen below:

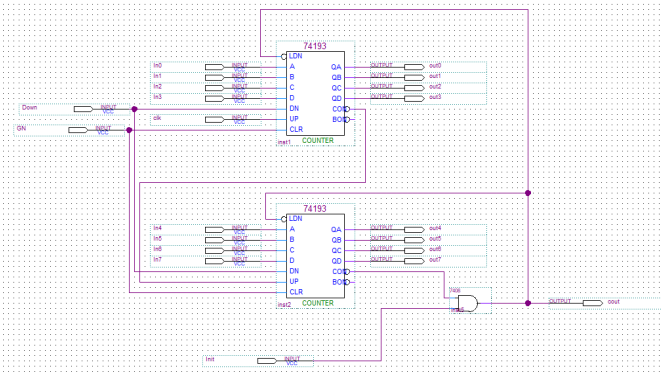

Figure 2

● A clearer picture of Figure2 exists in LAB2/ CounterDivider folder.

All the above project exists in LAB2/ CounterDivider folder.

5. I create a new project for in Quartus with "FrequencyRegulator" name. I combine FPGA and Counterdivider block diagram to build a Frequency regulator.

The block diagram of Frequency Regulator can be seen below:

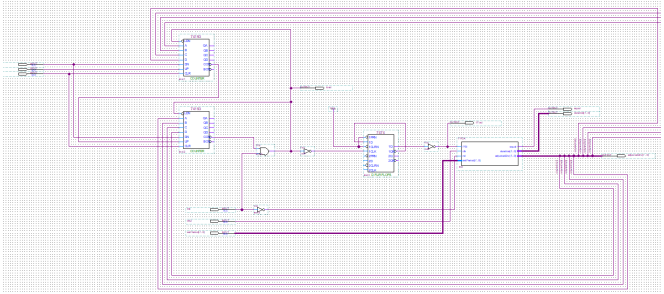

Figure 3

● A clearer picture of Figure3 exists in LAB2/ FrequencyRegulator folder.

6. I do this in FreqRegulator.vwf file.

7. Done!

8. This project files exist in LAB2/FrequencyRegulator folder.

Second Part: Design Simulation in Modelsim

1. For generating clock with 20MHz frequency, we use ring oscillator. The related files are in LAB2/ RingOscillator folder. Then we provide a testbench for our FrequencyRegulator which exists in LAB2/ FrequencyRegulator/Simulations/qsim folder.

2, 3. In this section, we tested different testcases, the results of them can be seen in the following table**.**

RingFreq	DesiredFreq	Final loads	Initial load	setPeriod
20.008MHz	50MHz	102	127	125

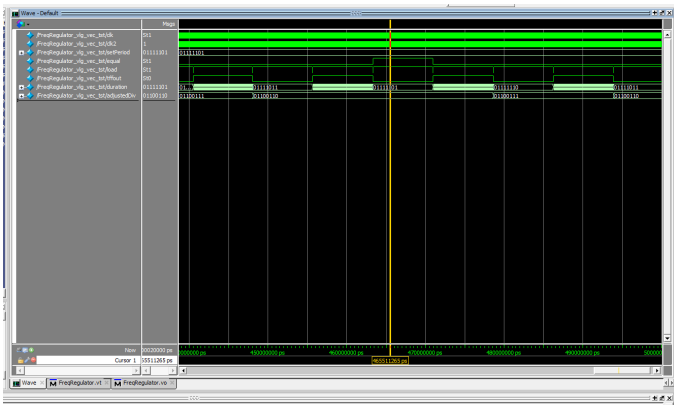

Figure 4

● Unfortunately, despite the many reviews and changes I made in the code, I still did not find the reason for the discrepancy between the answer I achieved and the answer exists in the Lab Instruction!