Experiment #2

Group 3

Nazanin Sabri
810194346
nazanin.sabrii@gmail.com

Nima Jarrahiyan
810194292
jarrahian.nima76@gmail.com

***Abstract*—working with Verilog, working with FPGAs, ring oscillators, creating a steady rectangular (square) pulse.**

***Keywords*— clock divider, oscillator, frequency divider, FPGA, voltage converter, frequency regulator**

I. Introduction

After building a clock with steady voltage peak in previous experiment, it's time to make our clock steady in frequency.

All signal generators may act different is different situations like heat or impact and may alter the output a little. With the help of FPGA we can solve this problem by analysing the signal and measure the difference from steady state. To do so in the first part of the experiment we will develop the module for measuring and altering the signal and second part we will implement data on FPGA.

Hardware implementation is almost as same as previous experiment with difference of outputs and inputs of FPGA which is output of FPGA is connected to frequency divider as entry.

II. Methodology and Procedures

We started off by writing the Verilog code for a frequency regulator using the basic idea of the 3 always blocks given in the laboratory manual. We then continued by writing a test bench for the module and testing the two along side one another. A number of issues we faced while doing this part were:

A. Despite the instruction in the laboratory manual, with the help of our sessions TA, we came to realize that, in order for our signals duration to reach the setPeriod we need to do the following:

If: duration > setPeriod

Then: frequency of our signal < frequency of the target signal

So we have to decrease the duration in order to get closer to the target period. (Which was the opposite of what had been instructed in the manual).

B. Another problem we faced while simulating our code was when the signals were changed and if they really were in the order we wanted them to be changed. In order to fix this problem we had to go back and take a look at all our always blocks sensitivity lists.

C. In writing the test bench for the Verilog module, In addition to using the concept of feedback, we learned how to use the current period of the signal with the clock value to create the durations we meant to create in the test bench.

Now it was time to put the code to use and program an FPGA and connect it to our circuit on the board. For the sake of simplicity we made 4 bits of the counter loads (the 4 most significant bits) constants. This change required wiring these four loads to VCC and GND instead of the FPGA inputs and outputs as well as modifying the Verilog code. Every wire coming in or going out of the FPGA was connected to a level convertor to handle the 3.3v FPGA appropriate and 5v circuit appropriate VCC values.

Due to the largeness of the circuit, the number of wires used, noise in the wires and … we had to recreate our circuit from scratch this time using shorter, stronger and more reliable wiring!

After the circuit was functioning again we started experimenting with different load values. In order to calculate the setPeriod with was given as an input to the FPGA we used the formula below:

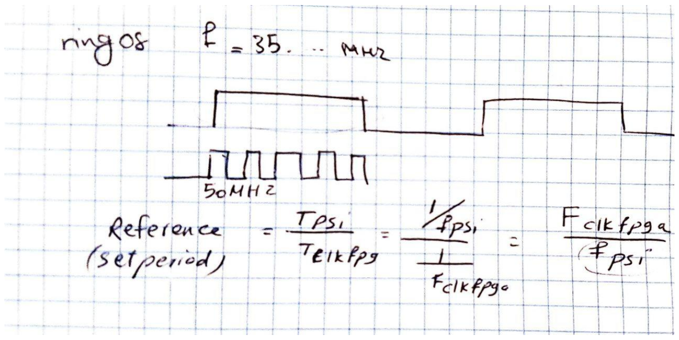

ring os
f = 35. .. MHz
50MHz
Reference (setperiod) = Tpsi / Tclkfpg = (1/fpsi) / (1/Fclkfpga) = Fclkfpga / fpsi

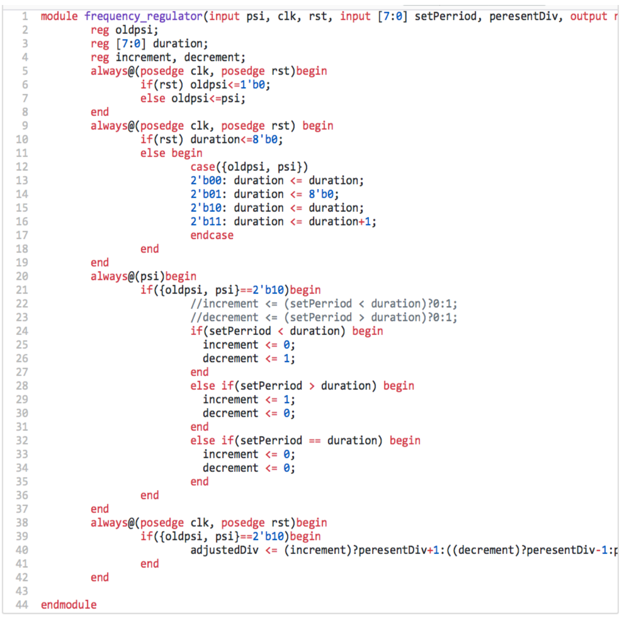
Fig. 1 Calculating T *psi*

The mathematical calculations for the output values in each individual step are available in part 2 of the results section.

III. Results

A. Part 1

The main body of module contains three ALWAYS statement. First one for defining and determining each sate of entry signal, second for comparing the entry signal with steady one and last one for implementing changes.



```verilog
module frequency_regulator(input psi, clk, rst, input [7:0] setPerriod, peresentDiv, output r
        reg oldpsi;
        reg [7:0] duration;
        reg increment, decrement;
        always@(posedge clk, posedge rst)begin
                if(rst) oldpsi<=1'b0;
                else oldpsi<=psi;
        end
        always@(posedge clk, posedge rst) begin
                if(rst) duration<=8'b0;
                else begin
                        case({oldpsi, psi})
                        2'b00: duration <= duration;
                        2'b01: duration <= 8'b0;
                        2'b10: duration <= duration;
                        2'b11: duration <= duration+1;
                        endcase
                end
        end
        always@(psi)begin
                if({oldpsi, psi}==2'b10)begin
                        //increment <= (setPerriod < duration)?0:1;
                        //decrement <= (setPerriod > duration)?0:1;
                        if(setPerriod < duration) begin
                          increment <= 0;
                          decrement <= 1;
                        end
                        else if(setPerriod > duration) begin
                          increment <= 1;
                          decrement <= 0;
                        end
                        else if(setPerriod == duration) begin
                          increment <= 0;
                          decrement <= 0;
                        end
                end
        end
        always@(posedge clk, posedge rst)begin
                if({oldpsi, psi}==2'b10)begin
                        adjustedDiv <= (increment)?peresentDiv+1:((decrement)?peresentDiv-1:p
                end
        end

endmodule
```

Fig. 2 Verilog Code of the Main Module

In first part, FPGA starts counting the entry signal named as PSI as long as PSI stays 1. The reason behind counting half of the PSI is that the flip flop would double the frequency. When PSI falls to zero the counter would reset or else the counter would remain same.

For the second part, the counter is compared with another entry number which is actually the steady frequency required by output named as 'setPerriod'. In this part two signals are generated named as 'Inc' and 'Dec', determining the procedure taken in next step.

In last part, if signal 'Inc' is on, output of FPGA would be incremented and if 'Dec' is one the output would be decremented. In this part we added another option (which was not required) as if Dec and Inc are both zero which would not alter the output.

Our first error with Quartus was not using any of the critical values (sensitivity list signals) like CLK and RST in our loop which were sensitive to always block which was resolved.

Our code worked well and was accepted by Quartus.

In the test bench we had to use different methods to get the operations going.

The test bench contains two loops within each other. One generates the clock and second one puts output to input of our system when PSI becomes 0.

The picture below is requirements for this part.

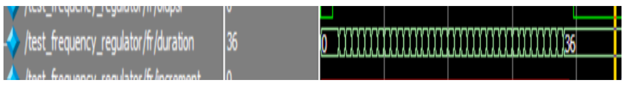


Fig.3 Starting value of Duration is 36

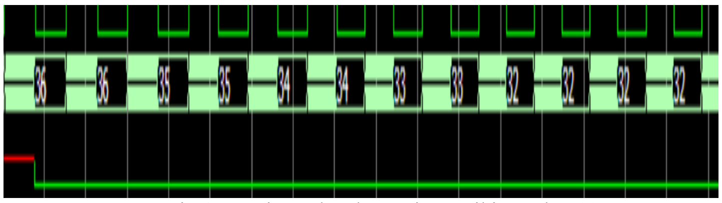


Fig.4 Duration value decreasing until it reaches 32

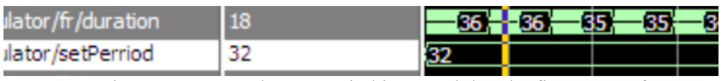


Fig.5 We can see that set period is 32 and that the first count of psi is 32 and that duration is decreasing

B. Part 2

This part was done in two sessions.

Since jumper wires are not trust worthy cables and the circuit was from the last experiment, we faced failure and our output was nothing but a 50 Hz sine signal even without using any input voltages.

To change the circuits, we had to use our own supplements and ICs and we did so. Our second circuit using simple wires worked fine and we were able to get a steady output from our Oscillator and flip flop. The only problem would be the FPGA.

First the need to understand the idea of the second part.

In the first part, we gave the FPGA all 8 inputs of Divider to change as mentioned but in the second part we set 4 most significant numbers set to 1 and only change the 4 least significant ones. So there might be a slightly different output as calculated.

Another problem that happened again was FPGA had loading effects on circuit. It seems that ground of FGPA should not be as same as the main circuit so we only attached the ground to the same ground of voltage converters and the problem was solved.

Another problem was level converters not working properly which solder took care of it.

Signal outputs that were changing by FPGA were not as desired. The problem was that we though outputs of Divider was connected to entry of FGPA which would not make sense of we wanted to change the inputs so we made changes and got our answers.

It seems that for frequencies higher than 200 KHz the difference between calculation and data rises significantly which may be due to setting most significant set to 1

For our records we have the table below which was accepted by head TA.

RO frequency	Set Period	Division	Parallel Load	TFF output
35 MHz	217	52	104 01101000	134 kHz
35 MHz	126	88	168 10101000	219.8 kHz

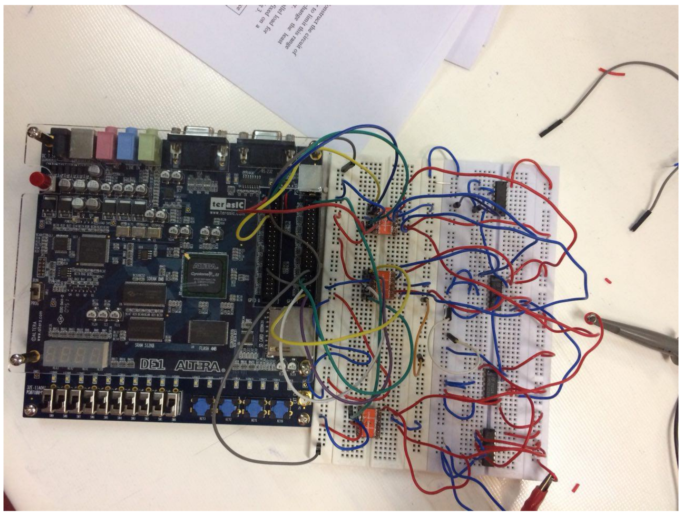


Fig.6 Implementation of circuit and FPGA

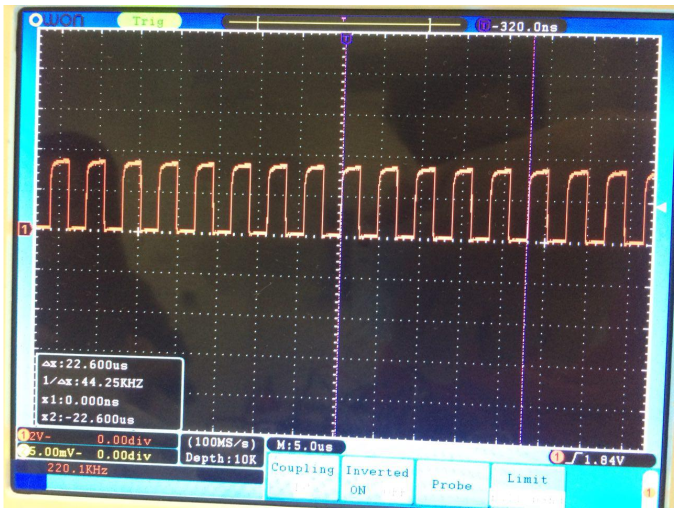


Fig. 7 Output signal divided