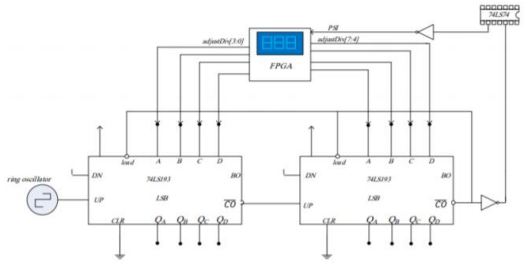
Experiment 2 – Frequency Regulator

Mohadeseh
Azari,
810196404

***Abstract*— this lab is about changing in frequency, first of all let's see what does that mean, "changing in frequency". Well, in the previous lab we've talked about one of the most important role in digital logic design, and that was the clock. We introduced lots of different ways to build the clock; But the problem is that our clock frequency is not a constant value and there are lots of factors that can affect its frequency. Changing in frequency is a crucial issue because synchronisation between the clocks is as important as the clocks themselves. In part1 we design a frequency regulator to change the initial value and so the frequency and finally in part2 we synthesis our design and change the value of set Period.**

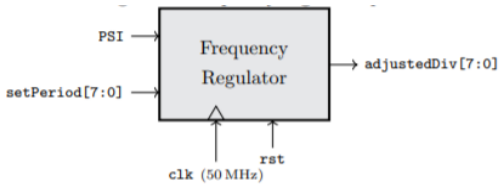
***Keywords*— synchronisation, frequency, adjusted clock**

I. Design Description and Synthesis



Clock Adjusting System

The processor should calculate the input frequency and compare it with a desired reference signal. If the reference and the input's frequencies are not the same, then it should change the division value.



Frequency regulator pins

Decide_when_to_count_and_count

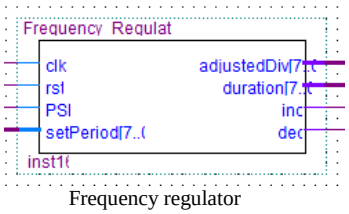
```
always @(posedge clk, posedge rst) begin: decide_when_to_count_and_count
  if(rst)
    duration<=9'b0;
  else begin
    case({previous_PSI,PSI})
      2'b01 : duration<=9'b0;           //zero to one
      2'b11 : duration<=duration+1;     //steady at one
      2'b10 : duration<=duration;       //one to zero
      2'b00 : duration<=duration;       //steady at zero
    endcase
  end
end

```

Comparison

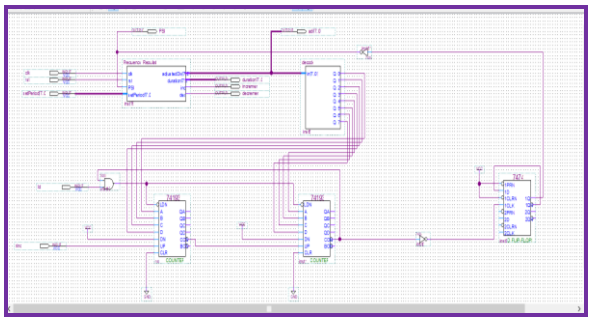
```
always@(negedge PSI) begin: comparison
  inc<=1'b0;
  dec<=1'b0;
  if({previous_PSI,PSI}==2'b10)
    if(duration>{1'b0,setPeriod})
      dec<=1'b1;
    else if(duration<setPeriod)
      inc<=1'b1;
end

```

Increment_decrement

```
always@(posedge clk,posedge rst) begin: increment_decrement
  if(rst)
    adjustedDiv<=8'b01111111;
  else begin
    if({previous_PSI,PSI}==2'b10)
      if(inc==1'b1) begin
        adjustedDiv<=adjustedDiv+1;

      end
      else if(dec==1'b1) begin
        adjustedDiv<=adjustedDiv-1;

      end
  end


end

```

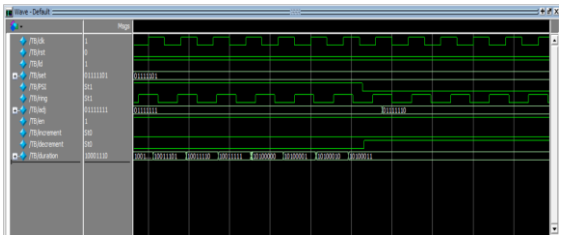


Frequency regulator

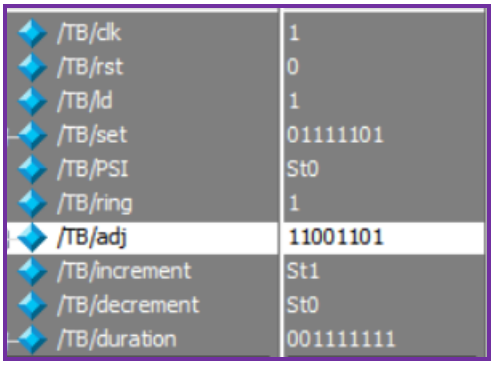


Schematic design

II. Design Simulation in Modelsim

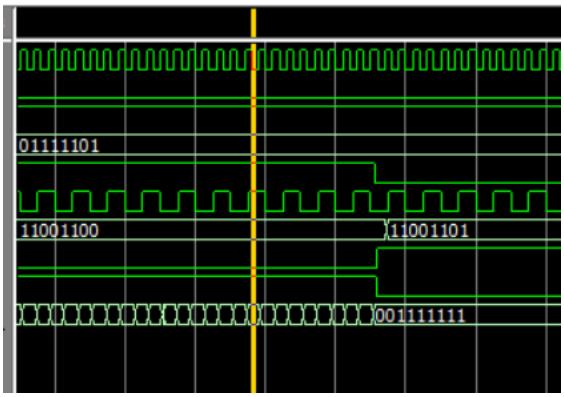


Test bench for 20MHz clock



Output signals

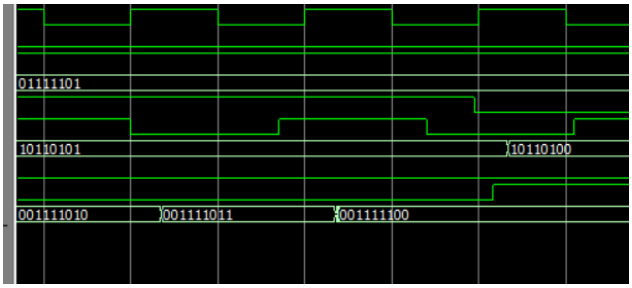
Zoom in:



As we can see the set Period is 01111101(125) and the duration value is 01111111(127) so the duration is higher than the set period and as you can see the increment signal gets one and our initial value that once was 11001100(204) is now 11001101(205).

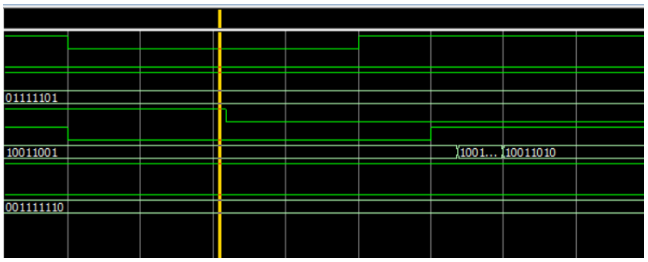
Ring Oscillator	*Desired Frequency*	*Final Parallel Loads*	*Initial Parallel Loads*	*Set period*	*Invert Delay*
20 MHz	400 kHz	205	127	125	8.4
30 MHz	400 kHz	180	127	125	5.6
40 MHz	400 kHz	155	127	122	4.17

Zoom in:



the set Period is 01111101(125) and the duration value is 01111100(124) so the duration is lower than the set period and as you can see the decrement signal gets one and our initial value that once was 10110101(181) is now 11001101(180).

Zoom in:



the set Period is 01111101(125) and the duration value is 01111110(126) so the duration is higher than the set period and as you can see the increment signal gets one and our initial value that once was 10011000(152) is now 10011010(154).

. Calculation ∞

$$\frac{Ring\ Oscillator\ Frequency}{modulo} = 400 * 10^3$$

$$255 - modilo = initial$$

Frequency		
20 MHz	$\frac{20*10^6}{400*10^3} = 50$	255-50=205
30MHz	$\frac{30*10^6}{400*10^3} = 75$	255-75=180
40MHz	$\frac{40*10^6}{400*10^3} = 100$	255-100 = 155

III. CONCLUSION

the conclusion after this lab was one easy way to reduce the effects of Asynchronizations between the clocks by simply insert an FPGA module to change the initial value.