LABORATORY MANUAL

Experiment 2
Frequency Regulation

UNIVERSITY OF TEHRAN

School of Electrical and Computer Engineering

Digital Logic Laboratory
ECE 045

Fall 1396

Prepared and developed by Katayoon Basharkhah under the direction of Professor Z.Navabi

EXPERIMENT 2 (Sessions 4,5)

Frequency Regulation

INTRODUCTION

Synchronization is a crucial issue in sequential digital circuit design. Many electrical devices devote multiple internal clocks to synchronize the internal processes. As you got familiar with the clock generation concepts in previous experiment, it is consisted of a reference clock generator like a ring oscillator that is desired to output a stable reference clock by putting an odd number of inverters in a loop chain. However, the thermal variation of the oscillator causes variations in the clock frequency and this ruins the calculations in a digital processor.

The goal of this experiment is to design an adjustable clock generator that can fix the output frequency at the desired value. You will work on the interface between the FPGA and the external breadboard. You will learn how to consider the hardware design to make it a synthesizable one.

By the end of this experiment, you should have learned:

- The concept of adjusting clock frequency
- Hardware implementation
- Writing testbench and simulation

The block diagram of the clock adjustment system is shown in Fig 1. It is consisted of two part. The first one is the circuit used in experiment 1 including a ring oscillator and a counter for frequency division. The second part is a processing element that receives the output of counter and sets its frequency to a specific value via a feedback path. In fact, the processor changes the value that the oscillator is divided by either higher or lower value.

After constructing the clock generator circuit on the breadboard, you are to write a Verilog code for the processing element in FPGA. First you should model systems behavior in Modelsim by writing an accurate hardware model in simulation and after model verification, you should set the interface between the breadboard and FPGA.

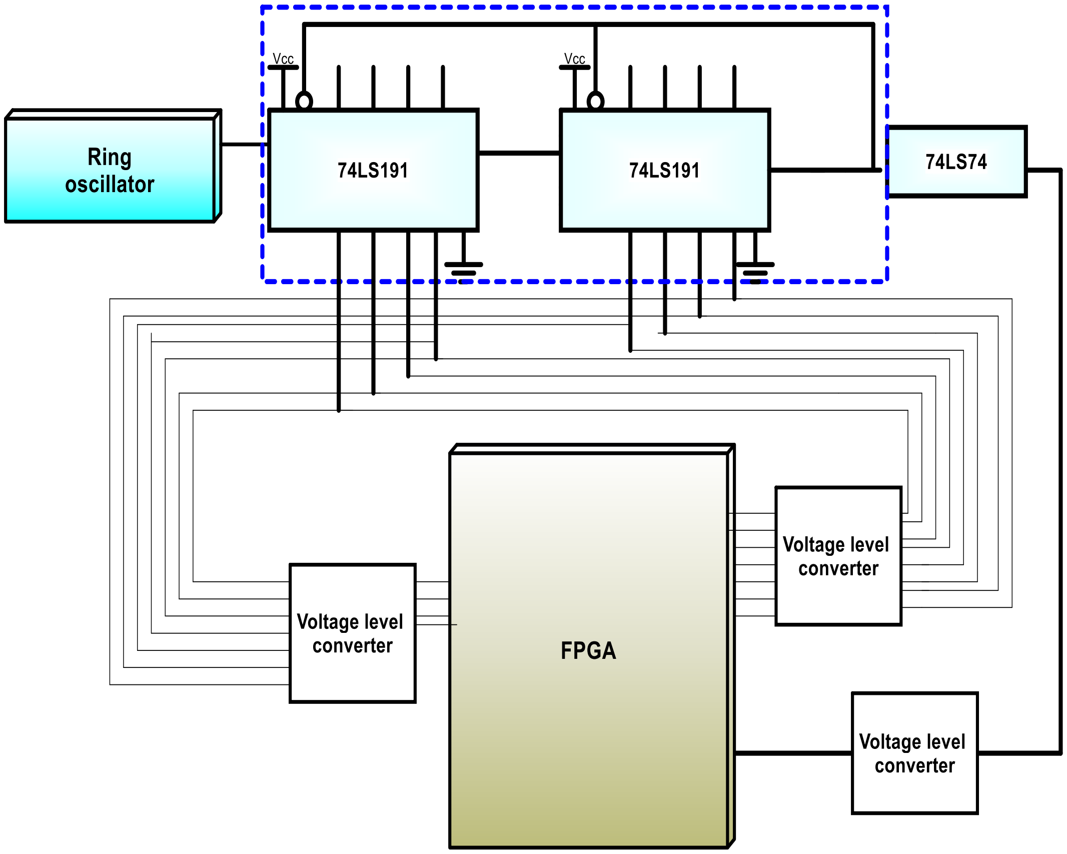

Fig 1- clock adjusting system

PART I
Design description and simulation

The goal of the design is to receive the output of frequency divider and set the output frequency at a fixed value. To do this a design like Fig 2 is needed. The processor should calculate the input frequency and compare it with a reference signal. If the reference and the input's frequencies are not the same then it should change the division value. So the load inputs of the counter should be fed to the FPGA and must be increased or decreased whether the frequency is higher or lower than the reference value.
DE1 board works with a 50MHz clock frequency that is definitely much higher than the input frequency. When the input comes in, FPGA starts to count its duration by this clock at the input rising edge and stops counting by the falling edge.

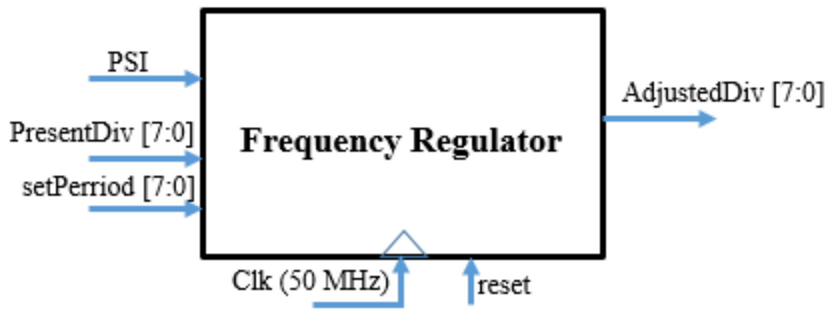

Fig 2: Frequency regulator pins

The inputs and outputs of the design are shown in Fig 2. The 8 bit load inputs of the frequency divider are the present value of the counter named as “PresentDiv” and the adjusted value for division after processing is called “AdjustedDiv”. The reference value called setPeriod is also another input that shows the ratio of the FPGA clock and the input signal. The Timing diagram of the design is shown in Fig4. To measure the input frequency,a counter starts to count the signal duration when the input signal gets “1”. When the input goes to “0”, the counter stops counting and the last value of the count will be stored in a register. At the rise edge of the input signal, the count value will be reset. Use an always statement like below to determine the duration value. You can use a case statement to set the duration value at 4 states described above.

```
always @(posedge clk, posedge rst) begin : decide_when_to_count_and_count
    if (rst)   ….;
    else begin
      case (input signal transition)
      // zero to one ---
     // steady at one---
     // one to zero---
    // steady at zero---
      endcase
    end
end
```

The comparison must be performed when the input is at “0”. Frequency adjustment will be performed by an increment or decrement signal. When the duration is more than the reference value, the present value of the load inputs should be increased and if its value is less than the reference signal then the load inputs should be decreased. Hence an Increment and decrement signal should be set to “1” after the comparison.
Again use an always statement to change the increment and decrement value at the proper time.

```
always @(input and count transition) begin : Comparison

    if (// one to zero//) begin
      ----(comparison)
      ---(set the Inc and Dec value)
    end
end
```

When the comparison is completed, by falling edge of the input signal and based on the increment or decrement value, the present value of the load inputs will be changed and registered to the adjusted value. The third always block is dedicated to this performance.

```
always @(posedge clk, posedge rst) begin : Increment_decrement
      if (// one to zeo//) begin
        ---(Increment or decrement)
    end
end
```

1- Write a Verilog code based on this instructions.
2- Provide a testbench for you design and verify your design. Note that the testbench is a high level abstraction model of your design. Therefore, you should accurately model the real hardware inside your testbench.
3- Include all the simulation results, the Quartus project and the simulation waveforms in your report.

PART II

Hardware implementation

After verifying your design, you should implement the hardware. Construct the circuit of Fig 1. Since the division range of an 8-bit counter is wide, it is better to limit this range for observing the frequency regulation. To do this, you can just change the least significant counter and set other four most significant bits to a fixed number.

Consider two frequencies 1.25 MHz and 200 KHz as an example. The parallel load for each of them is shown in the table. The four most significant bits can be fixed on a breadboard with a dip switch. The connections for the load inputs are shown in Fig 3.

RO frequency	Divider frequency (50% duty cycle)	Division	Parallel load	reference
25 MHz	1.25 MHz	10	246 (11110110)	20
25 MHz	200 KHz	63	193(11001110)	125

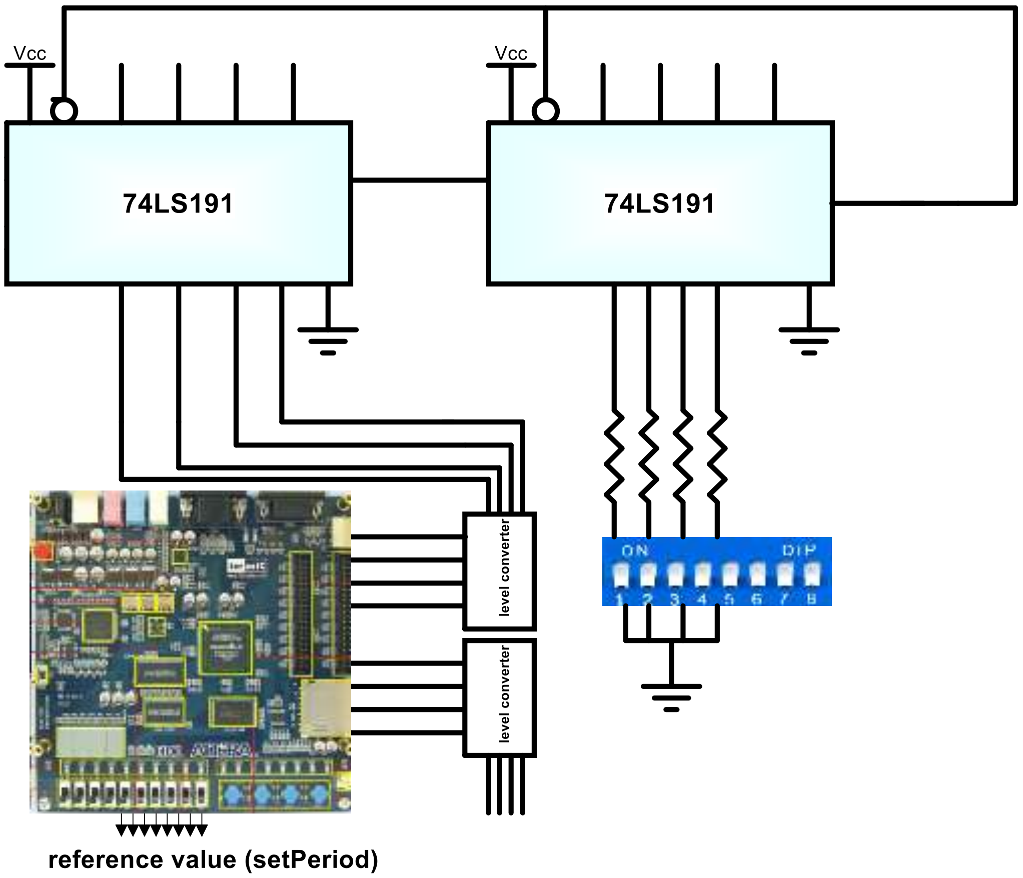

Fig 3: Connections of the design

1- Change the port declaration of your Verilog code in order to meet the connection layout of Fig 3.
2- Connect the four least significant load inputs to the FPGA pins. Do not forget to use level converters.
3- Put the reference signal assignment on the SW[8:0] of the DE1 board and assign SW[9] to the reset of your design.
4- Compile your code after doing the pin assignments.
5- Program the design on the DE1 board.
6- Observe the frequency of the divided signal and its variation (if there is any). Determine the frequency range and make a table like the manual and rite you on frequency range, reference value,, division factor and parallel load.

Hint: The values here are not necessarily the same as yours. You should determine a frequency range based on your own ring oscillator output.

7- Record the results and include all documents like images, files, figures and tables in your report.

PRELAB ASSIGNMENT

Before coming to the lab, answer these questions. The pre-lab needs to be handed in at the start of the lab.

1- Make a preliminary design of the experiment 1. You should declare all the inputs and outputs. Remember to use proper names for signal declaration.
2- Complete the always blocks; “decide_when_to_count_and_count”, “comparison”, “Increment_decrement”.
3- Write a testbench for the design.