**University of Tehran**
College of Engineering
School of Electrical & Computer Engineering

Experiment 2
Sessions 3,4

# Frequency Regulation

Digital Logic Laboratory
ECE 045
Laboratory Manual

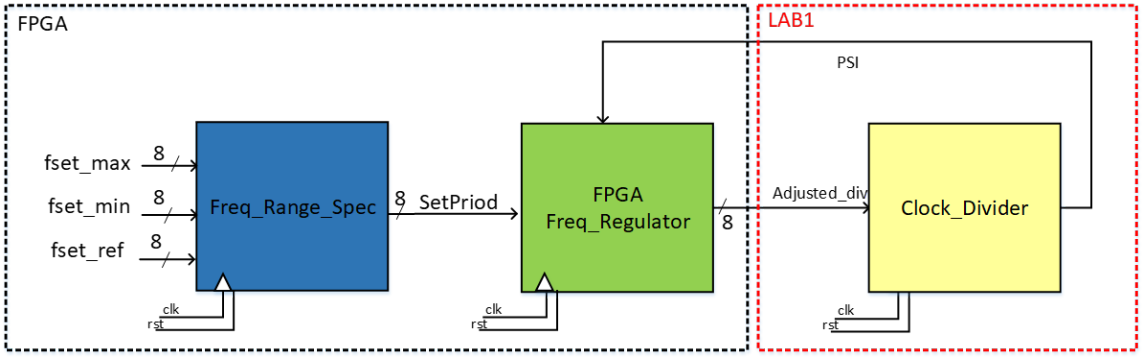
Fall 1399

# Contents




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

Figure 1: Overall view of the adjusting system




# 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 the 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 a desired value. You will learn how to consider the hardware design to make it a synthesizable one.

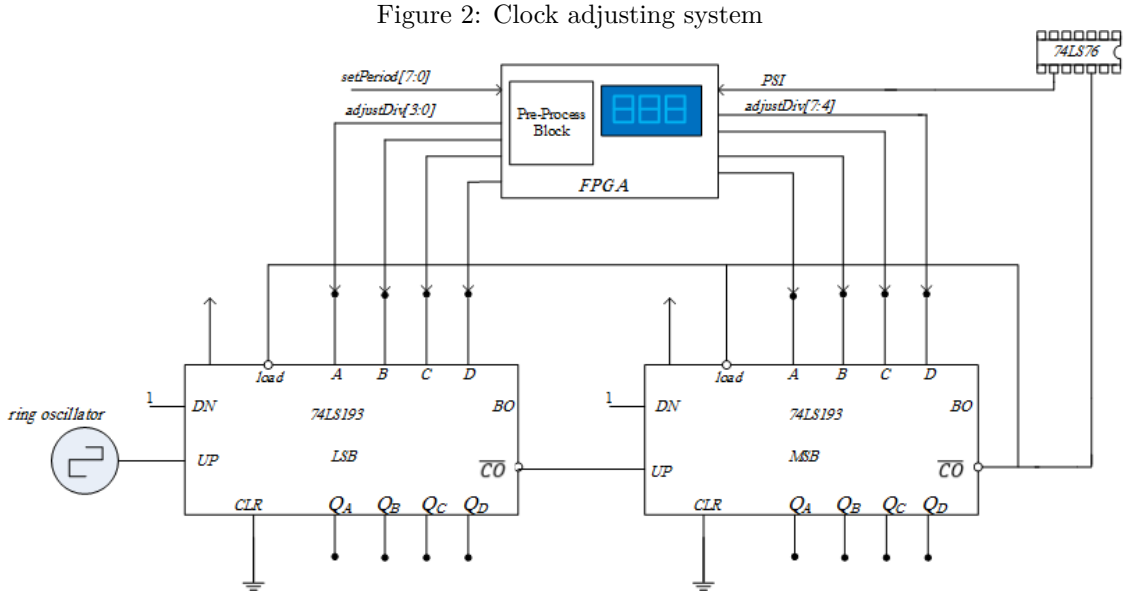
By the end of this experiment, you should have learned:

- The concept of adjusting clock frequency
- Hardware design
- Writing test-bench and simulation

figure 1 shows the overall view of the adjustment system. The frequency divider unit generates a clock. To set the frequency of this clock at a desired value, a processing unit called ”Freq-Regulator” checks the generated clock and regulates it if there is a mismatch between the generated and desired frequency. The processing unit receives a reference value, ”Setperiod” representing the desired frequency. Since this reference value can sit within a specified range, a pre-processing unit called ”freq-range-Spec” is used to check the user’s reference value. This value should be within a maximum and minimum values. Based on this description, this experiment include three parts as follows:

- Frequency range specification unit (FPGA)
- Frequency regulator unit (FPGA)
- Clock divider unit (LAB1)

Figure 2: Clock adjusting system



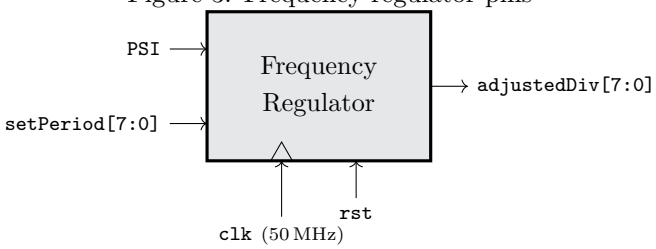

# 1 Design Description and Synthesis

## 1.1 Pre-Processing unit

The pre-processing unit called ”Freq-Range-Spec” is a simple logic block that receives two input values ”fset-max” and ”fset-min” as parameters. The user also provides a reference regulating value ”fset-ref” as the input of this block. The reference value is compared with maximum and minimum boundaries and a decision is made on the final frequency setting value ”Setperiod”. If this value is within a maximum and minimum value ”fset-max” and ”fset-min”, then it can be used as ”setperiod” for regulating the clock. Otherwise regarding being less than the minimum or more than the maximum value, it will be set to ”fset-min” or ”fset-max” respectively.

1. Write a Verilog code for this part and include it as a separate module in your design.
2. Draw the hardware of this module (on the paper) using RTL components.

Figure 3: Frequency regulator pins



## 1.2 Main Processing unit

The main processing unit is named "Frequency-Regulator". The inputs and outputs of this block are shown in figure 3. `PSI`, the divided clock after passing flip-flop, is the main input of FPGA and frequency regulation will be performed based on it. The regulation is performed by changing the value of the counter loads in clock divider. The adjusted value for division (counter load values) after processing is called `adjustedDiv`. A reference value called `setPeriod` is also another input that represents the desired output frequency in terms of number of clock cycles in one time duration. Cyclone IV board works with a 50 MHz clock frequency that is definitely much higher than the input frequency. When the input comes in, FPGA starts to count the input time duration by this clock at the input rising edge and stops counting by the falling edge.

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 counter will be stored in a register. At the rise edge of the input signal, the counter 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 falls to 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 flag value
    end
end
```

When the comparison is completed, by the 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 zero //) begin
        // increment or decrement
    end
end
```

1. Write a Verilog code based on description above.
2. Synthesize this code as a top-level entity.
3. Create a symbol for this frequency regulator module.
4. Add the pre-processing block of section 1.1 to this block diagram and set it as the top-level entity.
5. set the necessary inputs and outputs. Note that the Frequency Regulator module needs a 50 MHz clock signal that is separate from the divider clock.
6. Synthesize the top-level design.
7. Include all the synthesis results, the Quartus II files in your report.

### 1.3 Clock divider unit

For the clock division unit you will use the circuit of Experiment1, including two counters and a Flip-Flop.

1. Add the clock division unit of Experiment 1 to the block diagram of section 1.2.
2. Make the necessary connections.
3. Synthesize this block as a top-level entity.
4. Include all the synthesis results, the Quartus II files in your report.

## 2 Design Simulation in Modelsim

After synthesizing your design, you should verify the hardware.

1. Provide a test-bench for you design and verify your design. You should accurately model the real hardware inside your test-bench. To do this, like experiment one, use the ring oscillator with a frequency smaller than 50 MHz (near 20 MHz).
2. Scenario 1 Verifying Frequency Regulator: In your testbench, first start with approximately 20 MHz ring oscillator frequency. Then after proper time periods, change this frequency in small range near 20 MHz and verify the regulator performance. As shown in table 1, the required Setperiod for achieving this frequency is 125. Set the parameters ”setf-max” and ”setf-min” to 160 and 90 respectively for this special case.

   To report the results, fill the table 1 like the example. Keep the desired frequency at 400 KHz and change the ring oscillator frequency at different time periods and write the corresponding achieved parallel loads in the table. Like the example explained above, set the parameters ”setf-max” and ”setf-min” to near 30 percent upper and lower than the value of Setperiod.

Table 1: Scenario 1 with fixed "fset-ref" value

| Ring Oscillator Frequency | Desired Frequency | Final Parallel Loads | Initial Parallel Loads | Setperiod |
|---|---|---|---|---|
| 20 MHz | 400 kHz | 205 | 127 | 125 |

Table 2: Scenario 2 with different "fset-ref" values

| Ring Oscillator Frequency | Desired Frequency | Final Parallel Loads | Final frequency | fset-ref |
|---|---|---|---|---|
| 20 MHz | 400 kHz | 205 | ... | ... |

3. Scenario 2 Verifying the Preprocessor: In your testbench, keep the value of ring oscillator and desired frequency as the example values in table 2. Provide different values for input "fset-ref" in different time periods. These values must include both values within 160 and 90 and values outside this range. Verify the regulator performance in each case. report the final frequency and the corresponding parallel loads and fill the table.

4. In your report, include the Modelsim waveforms for these signals: duration, increment, decrement, setperiod, Ring oscillator clock, 50 MHz clock, PSI, and adjusteddiv.

## Acknowledgment

This lab manual was prepared and developed by Katayoon Basharkhah, PHD student of Digital Systems at University of Tehran, under the supervision of professor Zain Navabi.

This manual has been revised and edited by Zahra Jahanpeima, PHD student of Digital Systems at University of Tehran and Hadi Safari, undergraduate student of Computer Engineering at University of Tehran.