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

Experiment 2
Sessions 3,4,5

Clock Adjusting and Monitoring

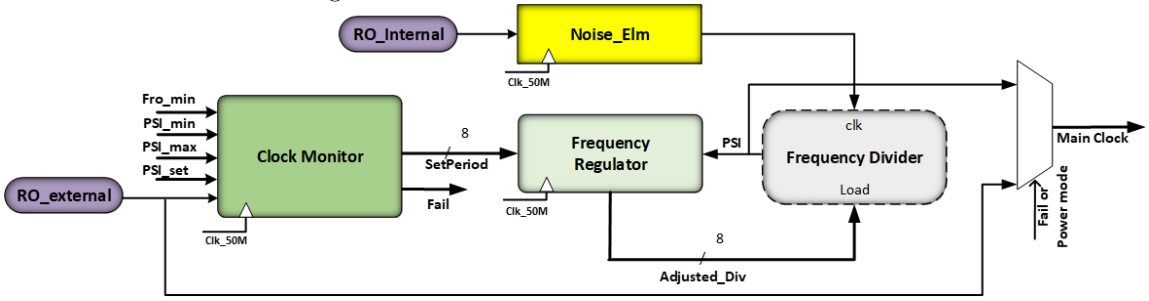
Digital Logic Laboratory
ECE 045
Laboratory Manual

Spring 1400

Contents

© Digital Logic Laboratory, University of Tehran, Spring 1400
Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi

Figure 1: Overall view of the clock source unit



Introduction

Synchronization is a crucial issue in sequential digital circuit design. Many electrical devices devote multiple clock sources to synchronize the 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 source 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 a clock source control unit that is used in microcontrollers. The operating frequency of a microcontroller is the single biggest factor in determining power consumption. The High-Speed Microcontroller family of microcontrollers supports different clock speed management modes that conserve power by slowing the internal clock. By controlling the source of the system clock, the system designer can simultaneously achieve higher performance and decreased power consumption. To provide maximum flexibility, the microcontroller will operate from two clock sources:

- External crystal or ring oscillator
- Internal ring oscillator

The external crystal oscillator is a large consumer of power on any microcontroller, especially during low power operation. The internal ring oscillator, used for quick starts from Stop mode, can also be used to provide an approximately 3 to 4MHz clock source during normal operation. Although an external crystal oscillator is still required at power-up, once the external crystal has stabilized, device operation can be switched to the internal ring oscillator in the power mode.

© Digital Logic Laboratory, University of Tehran, Spring 1400
Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi

Power Management Mode allows the user to run at slower speed to improve power savings by setting the clock divider rate bits and dividing the internal ring oscillator frequency.

Regarding figure 1 there are different components used in a clock source control unit:

- Frequency Regulator unit
- Frequency Divider unit (LAB1)
- Clock Monitor unit
- Noise Eliminator unit

The design selects between two different clock sources that are internal and external clocks and outputs it as the main operating clock for the microcontroller. As mentioned the system can work with a reduced speed in the power mode. In this mode, the internal clock frequency is divided by a user defined reference value and selected as the output. Since the ring oscillator is opposed to thermal variations, its frequency varies between a maximum and minimum value so a frequency regulator module will be used to adjust the frequency. On the other hand, the clock sources are subject to noises. To eliminate this noise, a noise eliminator circuit is used before performing the internal clock adjusting. The frequency divider unit generates a low speed clock. To set the frequency of this clock at a desired value, the clock monitor unit decides on the frequency reference value for the clock speed reduction and division to sit within a specified range. The clock monitor unit also decides if the external clock source is sufficient enough for the microcontroller start-up and non power mode operation. If it fails, then the alternative internal clock source would be chosen. Based on these descriptions, the design of each component is explained in the following sections.

1 Clock Adjusting Unit

1.1 Design and Synthesis

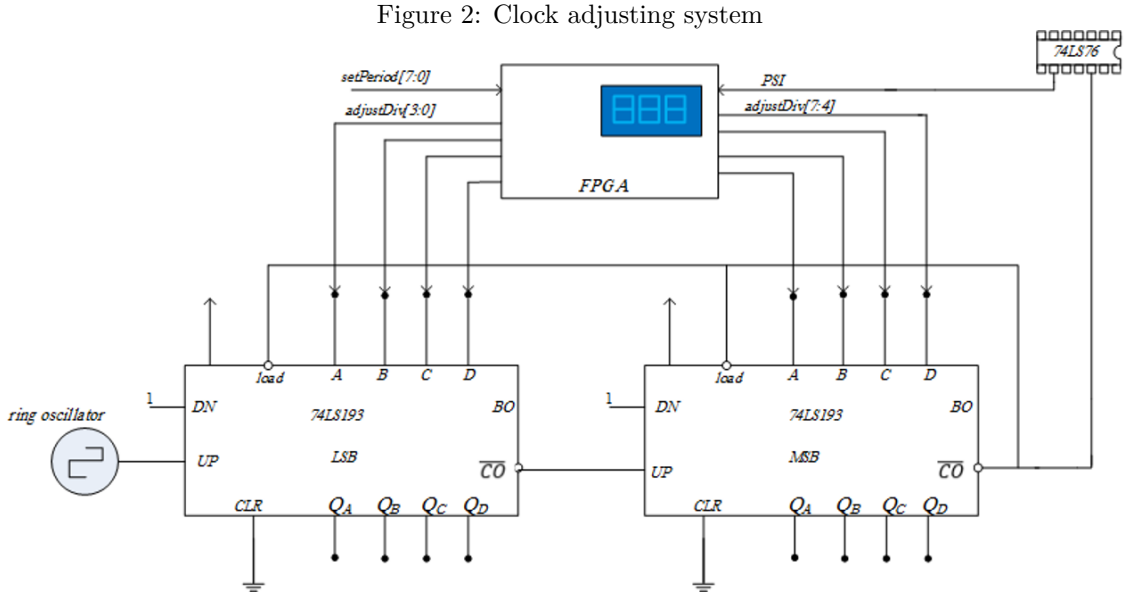
1.1.1 Frequency Regulator

The Clock Adjusting Unit includes the Frequency Regulator and Frequency Divider units. The inputs and outputs of the frequency Regulator 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 Frequency 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 50MHz 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:

© Digital Logic Laboratory, University of Tehran, Spring 1400
Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi

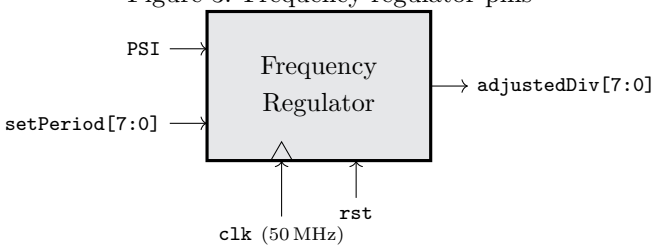
Figure 2: Clock adjusting system




```
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

Figure 3: Frequency regulator pins



© Digital Logic Laboratory, University of Tehran, Spring 1400
Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi

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.

1.1.2 Frequency Divider Unit

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

1. Add the Frequency divider unit of Experiment 1 to the block diagram of section 1 as shown in figure 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.

1.2 Simulation in Modelsim

After synthesizing your design, you should verify the hardware.

1. Provide a test-bench for your 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. Verifying Frequency Regulator: In your testbench, first start with approximately 20 MHz internal 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.

   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.

© Digital Logic Laboratory, University of Tehran, Spring 1400
Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi

Table 1: Scenario with fixed *setperiod* value

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

Table 2: Scenario 1 with different *PSIset* values

Internal FRO	Desired Frequency	Final Parallel Loads	Final frequency	PSIset
20 MHz	400 kHz	205	...	...

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

2 Clock Monitoring unit

2.1 Design and Synthesis

Clock Monitor unit is responsible for two tasks. The first task is to determine the 8-bit input of frequency regulator unit named *Setperiod* that sets the divided clock signal to the desired frequency. This unit is a simple logic block that receives two input values *PSImin* and *PSImax* as parameters. The user also provides a reference regulating value *PIOset* 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 the maximum and minimum values *PSImax* and *PSImin*, 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 *PSImin* or *PSImax* respectively. The second task of this unit is to decide if the external clock is suitable for the correct performance of the system. This is done by setting a flag named *Fail*. All external clock oscillator frequencies below $FROmin/6$ are reported as failure. For this purpose, the monitoring unit will count on every rising edge of the 50MHz clock during one period of the external ring oscillator, and issues the fail flag if the count value exceeds $6 \times FROmin$.When the external clock is failed, the clock source unit will select the internal divided clock as the main clock for the microcontroller.

1. Write a Verilog code based on description above.
2. Synthesize this code as a top-level entity.
3. Create a symbol for this clock monitor module.
4. Add the frequency regulator block of section 1 to this block diagram and set it as the top-level entity.
5. set the necessary inputs and outputs.
6. Synthesize the top-level design.
7. Include all the synthesis results, the Quartus II files in your report.

© Digital Logic Laboratory, University of Tehran, Spring 1400
Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi

2.2 Simulation in Modelsim

In this section you will verify the functionality of the clock monitor design when connected to the Clock Adjusting Unit. Both tasks of this unit that is the failure detection and *Setperiod* determination must be verified in your testbench. To do this you need to make an instance of another ring oscillator representing external clock.

1. Scenario 1: Verifying the *Setperiod* determination: In your testbench, keep the value of internal ring oscillator as the example value in table 2. Provide different values for input *PSIset* in different time periods. These values must include both values within 160 (*PSImin*) and 90 (*PSImax*) 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. Set the parameters *PSImin* and *PSImax* to 160 and 90 respectively for this special case. Keep the frequency of the external clock in 24MHz.

2. Scenario 2: Verifying the failure detection: In your testbench, set the parameters *PSImin* and *PSImax* to 160 and 90 respectively. Set the *PSIset* value in between these two values. Keep the internal Ring oscillator at 20 MHz and examine the failure of the system for at least two different frequencies for the external ring oscillator. Be sure to verify the design for both frequencies greater and less than $6 \times FROmin$.

3 Noise Eliminator unit

3.1 Design and Synthesis

In this experiment, a noise pulse (high or low) is one that lasts only for one clock cycle. In figure 4, input pulse goes from low to high, but included with some high and some low noise pulses. The goal is to clean this up and produce the ideal pulse as shown.

This circuit includes only a Moore FSM with four states. The below `always` statement shows the combinational part of this FSM. Starting with a zero value on input in state *Waitfor1*, we are looking for a 0 to 1 transition. With this transition, the machine goes to the next state, *Noisechk1*. Since the noise value wont last more than one clock cycle, the 1 value on the input will be considered as a noise if it goes to zero on the next clock cycle and it will go back to the first state. Otherwise, the next state would be *Stable1* and the machine stays in this state unless there is a 1 to 0 transition. With an incoming zero, we would got another noising check state called *Noisechk0*. Similarly, if the zero lasts for more than one clock cycle it would be considered as a true 0 and the machine goes to the first state. Otherwise it would go back to the state *Stable1*. Remember that the output value in both noise checking states must remain to its previous value since it must not change if there is a noise on the input. Note that in reality, the actual pulse is delayed by one clock Cycle.

1. Write a Verilog code based on description above.

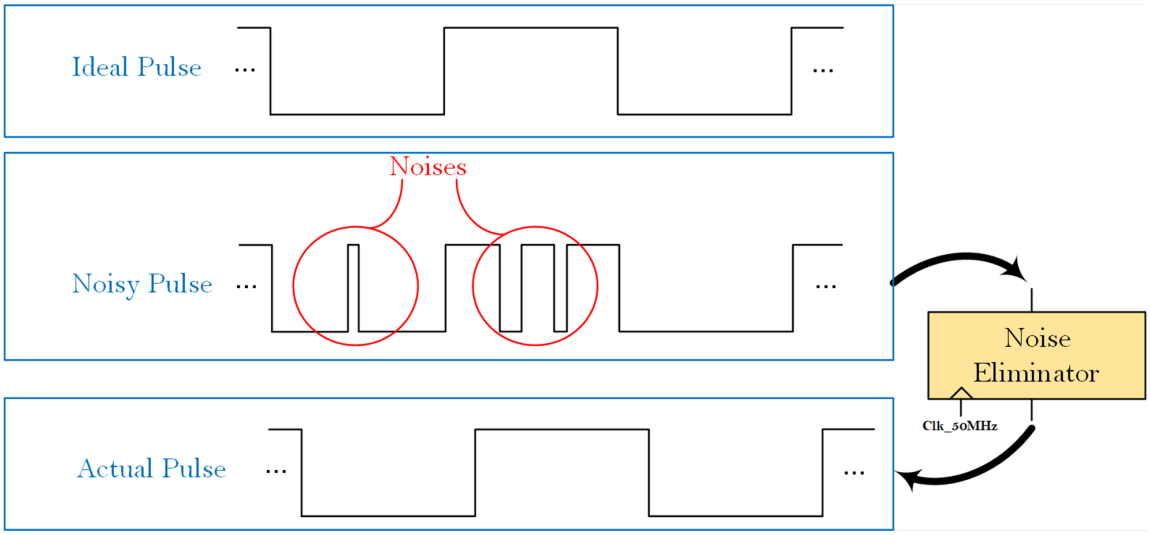
2. Synthesize this code as a top-level entity.

3. Create a symbol for this clock monitor module.

4. Add the design block of section 2 to this block diagram according to figure 1 and set it as the top-level entity.

© Digital Logic Laboratory, University of Tehran, Spring 1400
Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi

Figure 4: Noise Elimination



5. Synthesize the top-level design.
6. Include all the synthesis results, the Quartus II files in your report.

```
always@ (pstate, ...) begin // State Transitions
        case (pstate)
                Waitfor1:   begin ... end
                Noisechk1:  begin ... end
                Stable1:    begin ... end
                Noisechk0:  begin ... end
        endcase
        end
```

3.2 Simulation in Modelsim

1. In your testbench, connect the internal ring oscillator as the input of Noise Eliminator unit.
2. Inject at lease two noise pulses into this input. Be sure that the duration of each of this noise pulses is less than 50MHz clock.
3. Examine if the output clock is working correctly.
4. To verify the functionality of the overall system, you need to add a multiplexer for selecting between two clocks. Set the external clock frequency to a value less than $6 \times FROmin$ and

show which clock source is selected. For this, set the parameters *PSImin* and *PSImax* to 160 and 90 respectively. Set the *PSIset* value in between these two values. Keep the internal Ring oscillator at 20 MHz.

5. Issue the power mode flag and show the selected clock as the main clock source.For this, set the parameters *PSImin* and *PSImax* to 160 and 90 respectively. Set the *PSIset* value in between these two values. Keep the internal Ring oscillator at 20 MHz. Keep the external internal Ring oscillator at 24 MHz.

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 Maryam Rajabalipanah, PHD student of Digital Systems at University of Tehran.

© Digital Logic Laboratory, University of Tehran, Spring 1400
Prepared and developed by Katayoon Basharkhah under supervision of Professor Z. Navabi