

CS M152A Project 2

Clock Design Methodology

In this lab, you will learn how to use the Xilinx ISE program to design and test various clock waveforms

Introduction

For this lab, you will use the Xilinx ISE software to design and test various clock waveforms on a digital system. The Lab will go through the basic concept behind clocking a system and the techniques to generate them from a system clock,

This lab will be based on simulation only; no FPGA use will be involved. You are going to implement your design in Verilog HDL. At the end of the lab, you are expected to present a design project with source code and test bench, and the design will be focusing on comparing the different waveforms generated by your design.

Overview

The modules to design will take the system clock and a reset signal as the input and output the derived clock. There will be multiple modules in the design with one top module instantiating each of the designs for direct comparison. The top module and submodule are outlined below:

clock_gen.v Description	
Divide by 2^n Clock	The submodule exploring clock division by power of 2
Even Division Clock	The submodule exploring even clock division
Odd Division Clock	The submodule exploring odd clock division
Glitchy counter	The submodule exploring pulse/strobe/flag

Background

Clocking in digital systems allows high testability. Designers can run through waveform simulation or on physical devices by stepping one clock signal and check for expected behavior. Clocks are especially important with the popularity of synchronous data transmission, allowing signals to be communicated between devices. Quick examples are serial communication links such as SPI, I2C, RS232, UART, USB, PCIe, Ethernet, etc. Additionally, clocks are often used as the basis of timer systems used in numerous embedded devices such as traffic light, monitor screens, digital stopwatches, phones, etc.

The lab will focus on creating clocks and pulses of different frequency and duty cycles.

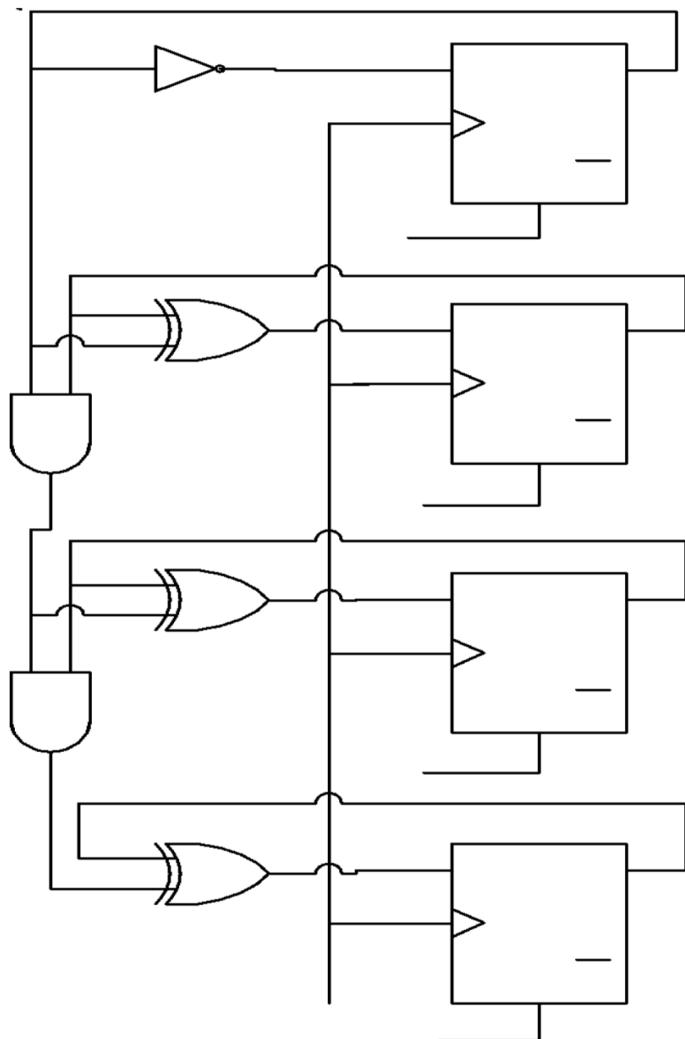
Modules to be explored:

1. Divide by power of 2 clocks
2. Even division clock using counters
3. Odd division clock using counters
4. Single-pulse strobe

0. Counters: Introduction

In the last lab, you learned the FPGA design workflow with a combinational logic design. Now we move to the sequential circuits. In this part, you will design and perform simulations, and explain the real implementations, of several small projects revolving around counters.

4 bit counter Schematic



With the knowledge you acquired from M51A, you should be able to easily recreate the above schematic for a 4-bit counter through the use of K-maps. Xilinx ISE does provide an interface for schematics creation, however, schematic-based design methods are no longer in use in the real world. Our job here is to translate the schematics into Verilog code.

4 bit counter: Verilog Implementation

```
// Example Verilog code for the counter
reg [3:0] a;
always @ (posedge clk)
if (rst)
    a <= 4'b0000;
else
    a <= a + 1'b1;
```

Since we are translating the schematics, each gate should be mapped to an operator, and each flip-flop should be mapped to one line in the edge-sensitive always block. Remember to use non-blocking assignment `<=` in edge-sensitive always blocks. When synthesized, the code will produce the same circuit as the schematic. Create a new module with an **input clk, rst**, and **output a**. Write a testbench for the above code snippet and test it in simulation. Set the input clock frequency to 100 Mhz in the testbench.

1. Clock Divider by Power of 2s:

Verify that the 4-bit counter works. Open up the waveform window and analyze the counter signals. You will notice that the least significant bit (LSB) of the counter is a direct division of 2. The 1st bit is 4 times as slow. The 2nd bit is 8 times as slow. From this you can see that a clock divider can be obtained by extracting the proper bit from the counter.

Design Task: Assign 4 1-bit wires to each of the bits from the 4-bit counter (1)

2. Even Division Clock Using Counters:

In continuation of the 4-bit counter design, generate the divide by 32 clocks by flipping the output clock on every counter overflow. (2)

In this design, the counter counts from 0 to 15 in decimal. On the 16th(0th) edge the output will flip. The total period of 1 clock pulse will be 32 positive edges or 32 times slower.

Design Task: Generate a clock that is 28 times smaller by modifying when the counter resets to 0. (3)

3. Odd Division Clock Using Counters:

Generate a 33% duty cycle clock using if statement and counters and **verify the waveform.** (4)

Duplicate the design in another always block that triggers on the falling edge instead. **View the two-waveform side by side.** (5)

What happens if you assign a wire that takes the logical or of the two 33% clocks. (6)

Design Task: Generate a 50% duty cycle divide-by-5 clock. (7)

4. Pulse/Strobes:

Create a divide-by-100 clock with only 1% duty cycle using the counter methods previously introduced in parts 2 and 3. Create a second always block that runs on the system clock (100Mhz) and switch the output clock every time the divide-by-100 pulse is active with an if statement. **Verify that the output clock is 50% duty cycle divide by 200 clock running at 500Khz.** (8)

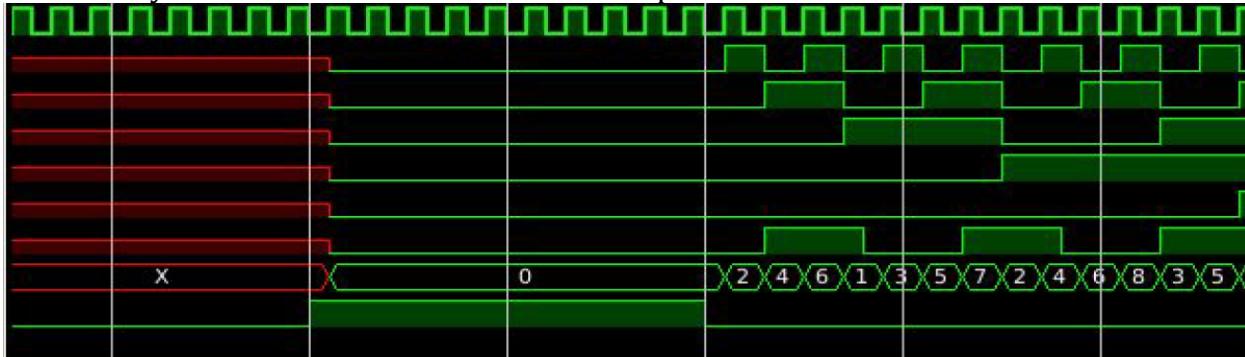
Design task: Use the master clock and a divide-by-4 **strobe** to generate an 8-bit counter that counts up by 2 on every positive edge of the master clock, but subtracts by 5 on every strobe. $0 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 8 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow 4 \rightarrow \dots$ (9)

Clock Generator Module:

Design task: Each of your 4 deliverable design tasks will be combined under the top module design below. You are allowed to name your submodules and ports however you like but the top module should have the following input and output. An example top module is attached on the last page. Your module

```
clock_gen(  
    input clk_in,  
    input rst,  
    output clk_div_2,  
    output clk_div_4,  
    output clk_div_8,  
    output clk_div_16,  
    output clk_div_28,  
    output clk_div_5,  
    output [7:0] toggle_counter  
)  
Endmodule
```

Make sure you obtain all 9 waveforms for the top level module as shown below



An example output of a successful Clock Gen Module waveform is shown above. The signals are ordered from top to bottom as the following

1. Input clock
2. Divide by 2 clock
3. Divide by 4 clock
4. Divide by 8 clock
5. Divide by 16 clock
6. Divide by 28 clock
7. Divide by 5 clock
8. Glitchy counter
9. Reset signal

Deliverables

The project consists of verification tasks **1-9** and 4 design tasks. You need to implement all the 9 verification tasks and add waveforms and brief explanations in your report. However, the final Verilog code that you submit should be the `clock_gen` module as explained above. When you finish, the following should be submitted for this lab:

1. **Verilog source code** for the “`clock_gen`” module. The file should be named exactly as “`clock_gen.v`” and the module and port names should exactly match names defined in the Clock Generator Module Section. Also, note that this code should be completely synthesizable. There is no restriction on the naming of the submodules but make sure to place all the submodules in the `clock_gen.v` file. An example is outlined on the last page.
2. **Verilog testbench** you used to evaluate your design. Note that your testbench is graded based on the correctness of the waveforms generated in your report. Please name the file “`testbench_UID.v`” where UID is your UCLA ID.
3. **Lab Report** should be consisting of explanations about your module and testbench design. Explain ideas you used to implement different blocks. Also, explain how you test your design. **Please document the waveforms for all verify tasks and design**

tasks. Generate a waveform for the final Clock Generator showing all the 9 ports. Please name your report “UID.pdf” where UID is your UCLA ID.

4. **Video Demo:** a 10 minute video showing your screen and focusing on the final clock_gen.v module and explaining the concepts utilized from previous sections.

```
module clock_gen(
    input clk_in,
    input rst,
    output clk_div_2,
    output clk_div_4,
    output clk_div_8,
    output clk_div_16,
    output clk_div_28,
    output clk_div_5,
    output [7:0] toggle_counter
);
clock_div_two task_one(
    .clk_in    (clk_in),
    .rst      (rst),
    .clk_div_2(clk_div_2),
    .clk_div_4(clk_div_4),
    .clk_div_8(clk_div_8),
    .clk_div_16(clk_div_16)
);
clock_div_twenty_eight task_two(
    .clk_in    (clk_in),
    .rst      (rst),
    .clk_div_28(clk_div_28),
);
clock_div_five task_three(
    .clk_in    (clk_in),
    .rst      (rst),
    .clk_div_5(clk_div_5)
);
clock_strobe task_four(
    .clk_in    (clk_in),
    .rst      (rst),
    .toggle_counter (toggle_counter),
);
endmodule

module clock_div_two(
    clk_in, rst, clk_div_2, clk_div_4, clk_div8, clk_div16
);
endmodule
module clock_div_twenty_eight(
    clk_in, rst, clk_div_28
);
endmodule
module clock_div_five (
    clk_in, rst, clk_div_5
);
endmodule
module clock_strobe (
    clk_in, rst, toggle_counter
);
endmodule
```