CS M152A Project 3

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 as an input, a reset signal 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. Divide by 3 clocks and odd division
4. Single pulse strobe

1. Clock Divider by Power of 2s:

Using the design from lab 1 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 1^{st} bit is 4 times as slow. The 2^{nd} 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.

Using counters can easily generate clocks that have periods that are even integer multiple of the original.

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 16^{th}(0^{th}) 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 or module 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 by modifying 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 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.

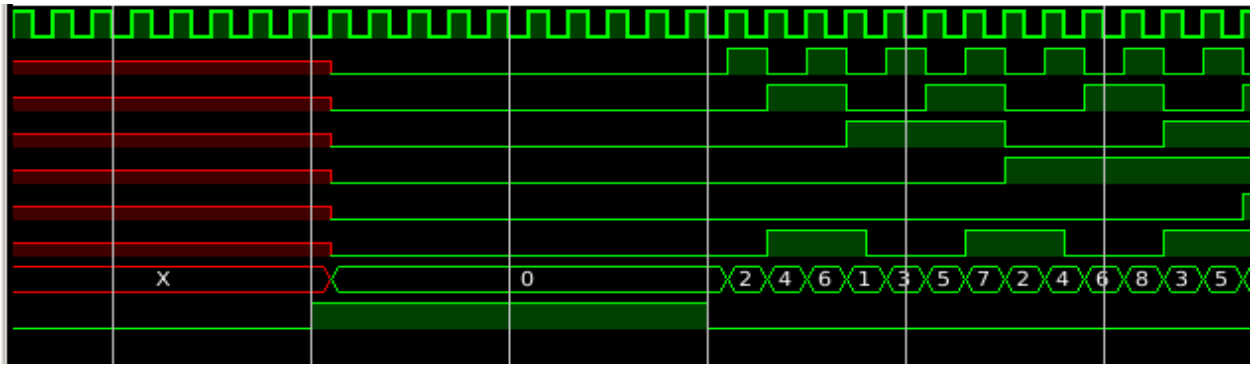
0→2→4→6→1→3→5→7→2→4→6→8→3→5→7→9→4→….. **(9)**

Clock Generator Module:

Design task: Each of your 4 deliverable 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

```
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] glitchy_counter
        );
endmodule
```
(10)

An example output of a successful Clock Gen Module waveform is shown below. This is the **10th waveform** required from the lab, you need the **other 9** from other sections. Make sure you obtain **all 10 waveforms** outlined in the lab. The signals are ordered from top to bottom as the following



Input clock
Divide by 2 clock
Divide by 4 clock
Divide by 8 clock
Divide by 16 clock
Divide by 28 clock
Divide by 5 clock
Glitchy counter
Reset signal

Deliverables

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 Clock Generator Module Section. It is very important as your code is automatically evaluated. 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. Note your final testbench should be instantiating clock_gen.v. Testbenches for the verified tasks do not need to be submitted, but the waveforms are required in your report.
3. **Synthesis & Implementation Report:** attach the full synthesis report and implementation (map) report as txt files. Relevant information should be mentioned in your lab report.
4. **Lab Report** should contain explanations about your module and testbench design. Explain ideas you used to implement different blocks. Also, explain how you test your design. Schematics can be generated from ISE but please explain how your verilog results in the RTL generated. **Please document the waveforms for all verification tasks and design tasks. Additionally, generate a waveform for the final Clock Generator with all 9 ports showing.** Please name your report "UID.pdf" where UID is your UCLA ID.
5. **Video demo:** focusing on the final clock_gen.v module and explaining the concepts utilized from previous sections is sufficient.

Additional clarification:

1. **Follow the syllabus for submission requirements first. ****
2. Only the module name and port name of clock_gen.v needs to be standardized. You can format however you want for verification tasks, design tasks, and submodules.
3. You only need to record the waveform of the **verification task** (no code submission is required). The only code necessary for submission is from the **clock_gen.v** and the associated submodules from the **design tasks** (design and final test bench file).
4. The input clock frequency can be arbitrary but you should probably use the **10ns** period(100Mhz) clock to emulate the original clock frequency of the fpga board. The important aspect is the ratio between the input clock and output clock.
5. You may choose to put submodules in different files but I will only check clock_gen.v and associated testbench. If the submodules do not get associated properly you will be held accountable.
6. **There is a total of 10 waveform captures labeled by (#). You need all (1)-(10)**

```
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] glitchy_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),
	.glitchy_counter (glitchy_counter)
);
endmodule
module clock_div_two(
	clk_in, rst, clk_div_2, clk_div_4, clk_div_8, clk_div_16
);
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, glitchy_counter
);
endmodule
```