CS M152A Lab 2: Clock Design Methodology

Arnold Pfahnl

TA: Mohit Garg, Winter 2021

Contents

1 Introduction

In this lab, we take a system clock and output various derived clocks. In general, the groupings of derived clocks in this lab include divide-by-2^n clocks, even division clocks, odd division clocks, and glitchy counter clocks.

In the following sections, nine tasks are explored, with four of these tasks being implemented as submodules of the *clock_gen* module. For tasks that are implemented in the *clock_gen* module, the overall waveform will be in section *Clock Generator Module*.

2 Clock Divider by Powers of Two (Task 1)

A basic 4-bit counter can represent division-by-2/4/8/16 clocks. To implement this design, I created a 4-bit counter that, given the positive edge of an input clock, resets to 0 on a reset signal and increments by 1 otherwise. Bit 0 (least significant bit) is assigned to the divide-by-2 clock, bit 1 is assigned to the divide-by-4 clock, bit 2 is assigned to the divide-by-8 clock, and bit 3 (most significant bit) is assigned to the divide-by-16 clock. These assignments were made since the least significant bit is a direct division of 2, bit 1 is 4 times as slow, bit 2 is 8 times as slow, and bit 3 is 16 times as slow.

Since these clocks form one of the four deliverable design tasks, their waveforms are under the Clock Generator Module section. These clocks are implemented as submodule *clock_div_two* for the *clock_gen* module and produce the *clk_div_2*, *clk_div_4*, *clk_div_8*, and *clk_div_16* waveforms.

3 Even Division Clocks Using Counters

3.1 Divide-by-32 clock (Task 2)

In the divide-by-32 clock, the clock flips on every 16th edge of the input clock. In my implementation utilizing a 4-bit counter, if the counter equals $4`b1111$, the clock flips and the counter is reset to 0.

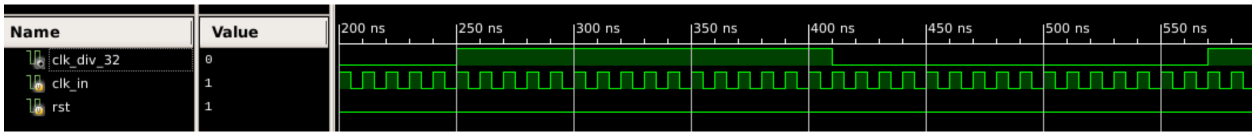


Figure 1: *clk_div_32* waveform for a divide-by-32 clock. *clk_in* and *rst* are the input clock and reset, respectively.

3.2 Divide-by-28 clock (Task 3)

My implementation for the divide-by-28 clock is very similar to the divide-by-32 clock. The only difference is that the clock flips (and counter reset) when the counter equals $4'b1101$.

Since this clock is one of the four deliverable design tasks, the waveform for this clock is under the Clock Generator Module section. This clock is implemented as submodule *clock_div_twenty_eight* for the *clock_gen* module and produces the *clk_div_28* waveform.

4 Odd-Division Clocks Using Counters

4.1 33% duty cycle clocks (Tasks 4-6)

Basing a clock solely off of when a counter's bits change is not sufficient for odd-division clocks. In this section, 33% duty cycle clocks are combined to create a 50% duty cycle divide-by-3 clock.

Task 4: The design first begins with a 33% duty cycle clock that triggers on the positive edge of the input clock (*odd_pos* in the waveform below). To implement this, I used a 2-bit counter that resets to 0 when the counter reaches $2'd2$ and increments by 1 otherwise. *odd_pos* is set to bit 1 of the counter. This is because the counter will be in one of three states: $2'b00$, $2'b01$, $2'b10$, and, thus, approximately 33% of the time bit 1 is set.

Task 5: For the second part of the design, a 33% duty cycle clock triggers on the negative edge of the input clock (*odd_neg* in the waveform below). Since the positive edge clock is already implemented, *odd_neg* is implemented such that on the negative input clock edge, *odd_neg* is set to the value of *odd_pos*.

Task 6: Finally, combining the positive- and negative-edge 33% duty cycle clocks results in a 50% duty cycle divide-by-3 clock (*odd_or* in the waveform below). As the name suggests, this clock is created by OR-ing the positive- and negative-edge 33% duty cycle clocks together.

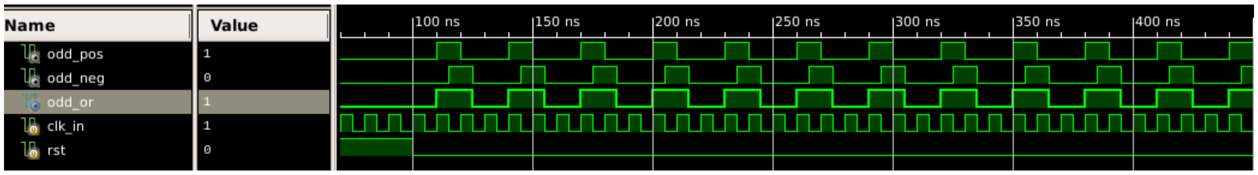


Figure 2: *odd_pos*, *odd_neg*, and *odd_or* waveforms corresponding to a positive-edge 33% duty cycle clock, negative-edge 33% duty cycle clock, and a clock composed of the previous two OR-ed together (divide-by-3 clock). *clk_in* and *rst* are the input clock and reset, respectively.

4.2 50% duty cycle divide-by-5 clock (Task 7)

Creating a 50% duty cycle divide-by-5 clock (*clk_div_*5), is nearly identical to the steps for creating a 50% duty cycle divide-by-3 clock in the previous section. Instead of a 2-bit counter, a 3-bit counter is used, and the counter is reset when it reaches $3'd4$.

The positive-edge clock contribution to *clk_div_*5 is set to bit 1 of the counter. Since the counter will be in one of five states: $3'b000$, $3'b001$, $3'b010$, $3'b011$, $3'b100$, the clock will be active for 2 of these 5 states, or 2 input cycles out of 5. The clock being active is equivalent to half a cycle, and since we want a divide-by-5 clock, our clock needs to be active for 2.5 input cycles.

The negative-edge clock contributes the extra half cycle. The negative-edge clock is active for the same states, but offset by half of the input clock cycle, providing the additional half cycle when OR-ed with the positive-edge clock.

Since this clock is one of the four deliverable design tasks, the waveform for this clock is under the Clock Generator Module section. This clock is implemented as submodule *clock_div_five* for the *clock_gen* module and produces the *clk_div_*5 waveform.

5 Pulse/Strobes

5.1 Divide-by-100 and divide-by-200 clocks (Task 8)

To create a divide-by-100 clock with 1% duty cycle (*clk_div_*100 in the waveforms below), I used an 8-bit counter that starts counting from $8'd29$ and resets when it reaches $8'd128$. I specifically chose $8'd128$ since the most significant bit (the eighth bit) will only be set for one input clock cycle. $8'd29$ was chosen as the reset value to make the clock cycle 100 times that of the input clock cycle. As confirmation, the period of one cycle (the difference between the two blue markers of the zoomed-in waveform in Figure 4) is 1000 ns, and since each input clock cycle is 10 ns long, the divide-by-100 clock cycle is, in fact, 100 times longer than the input clock cycle.

Next is the divide-by-200 clock with 50% duty cycle (*clk_div_*200 in the waveforms below). On every positive edge of the input clock, I checked to see if *clk_div_*100 was set, and if so the divide-by-200 clock's output flips. In the zoomed-out waveform in Figure 3, we confirm that the cycle length is 2000 ns, or 200 times longer than the input clock cycle.

Zoomed-out view:

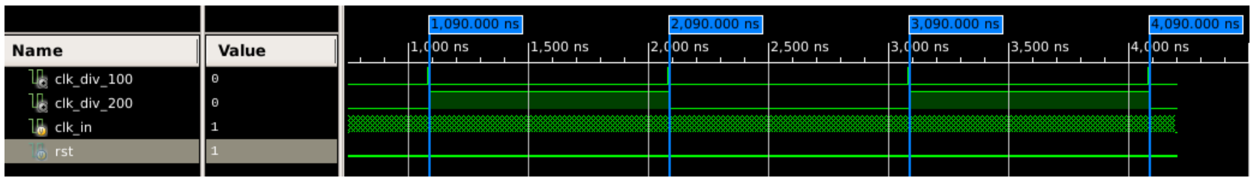


Figure 3: *clk_div_*100 and *clk_div_*200 zoomed-out waveforms. *clk_in* and *rst* are the input clock and reset, respectively.

Zoomed-in view:

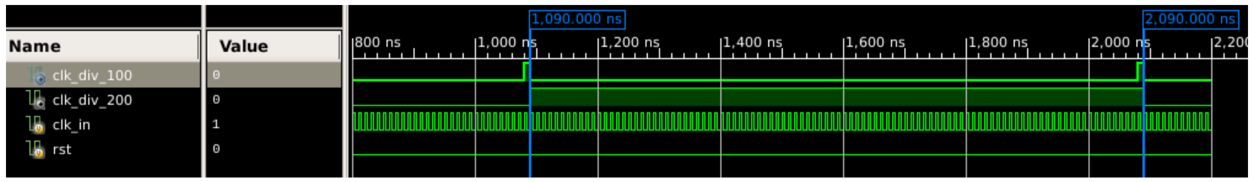


Figure 4: *clk_div_*100 and *clk_div_*200 zoomed-in waveforms. *clk_in* and *rst* are the input clock and reset, respectively.

5.2 Glitchy counter (Task 9)

The glitchy counter (*toggle_counter*[7 : 0]) uses a divide-by-4 strobe and input clock to generate an 8-bit counter that increments by 2 on positive edges, and decrements by 5 on every strobe. The following is the beginning of the pattern:

$$0 \to 2 \to 4 \to 6 \to 1 \to 3 \cdots$$

I confirmed the output of *toggle_counter* to be the same as the expected pattern by viewing the waveform of *toggle_counter* in Figure 5.

Since this clock is one of the four deliverable design tasks, the waveform for this clock is under the Clock Generator Module section. This clock is implemented as submodule *clock_strobe* for the *clock_gen* module and produces the *toggle_counter*[7 : 0] waveform.

6 Clock Generator Module

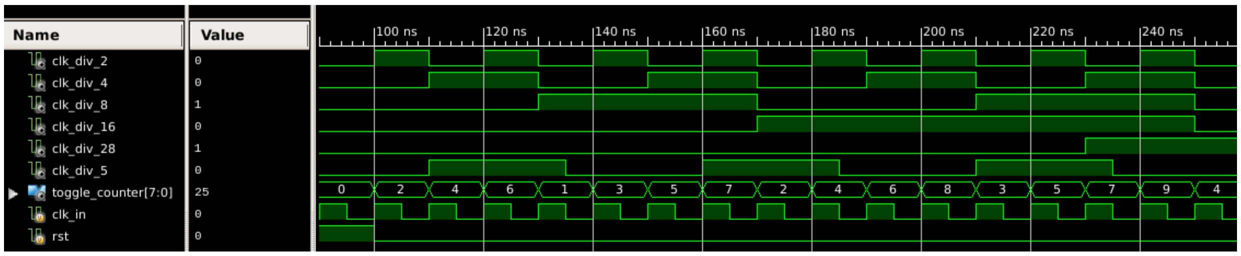
The clock generator module, *clock_gen*, implements four clock submodules producing a total of seven clock signals from one input clock signal *clk_in* and one reset signal *rst*.

The specific clocks produced are a divide-by-2 clock *clk_div*_2, a divide-by-4 clock *clk_div*_4, a divide-by-8 clock *clk_div*_8, and a divide-by-16 clock *clk_div*_16 all from the Clock Divider by Powers of Two (Task 1) section and produced by the *clock_div_two* submodule, a divide-by-28 clock *clk_div*_28 from the Divide-by-28 clock (Task 3) section produced by the *clock_div_twenty_eight* submodule, a divide-by-5 clock *clk_div*_5 from the 50% duty cycle divide-by-5 clock (Task 7) section produced by the *clock_div_five* submodule, and a glitchy counter *toggle_counter*[7 : 0] from the Glitchy counter (Task 9) section produced by the *clock_strobe* submodule.

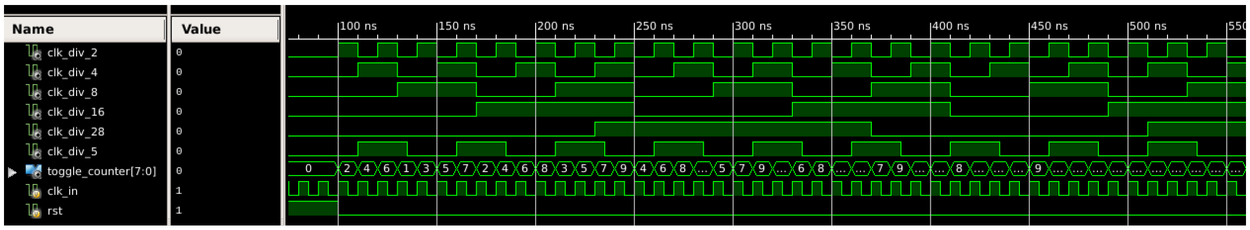
Figure 5 contains the waveforms produced by the *clock_gen* module clocks, and Figure 6 contains a high-level schematic of the *clock_gen* module, its submodules, and its input and output ports.

6.1 Simulation Waveforms

Figure 5 contains the waveforms for the clocks generated by the *clock_gen* module. Figure 5b, specifically provides a zoomed-out view that allows for a better look at longer waveform cycles such as *clk_div*_28 and *clk_div*_16.



(a) *clock_gen* waveforms zoomed in.



(b) *clock_gen* waveforms zoomed out.

Figure 5: *clock_gen* waveforms.

From top to bottom:

1. Divide-by-2 clock *clk_div_2* from section 2
2. Divide-by-4 clock *clk_div_4* from section 2
3. Divide-by-8 clock *clk_div_8* from section 2
4. Divide-by-16 clock *clk_div_16* from section 2
5. Divide-by-28 clock *clk_div_28* from section 3.2
6. Divide-by-5 clock *clk_div_5* from section 4.2
7. Glitchy counter *toggle_counter*[7 : 0] from section 5.2
8. Input 100 MHz clock *clk_in*
9. Reset signal *rst*

6.2 Clock Generator Module Schematic

The module schematic is depicted in Figure 6. All submodules share the same input clock and input reset signal.

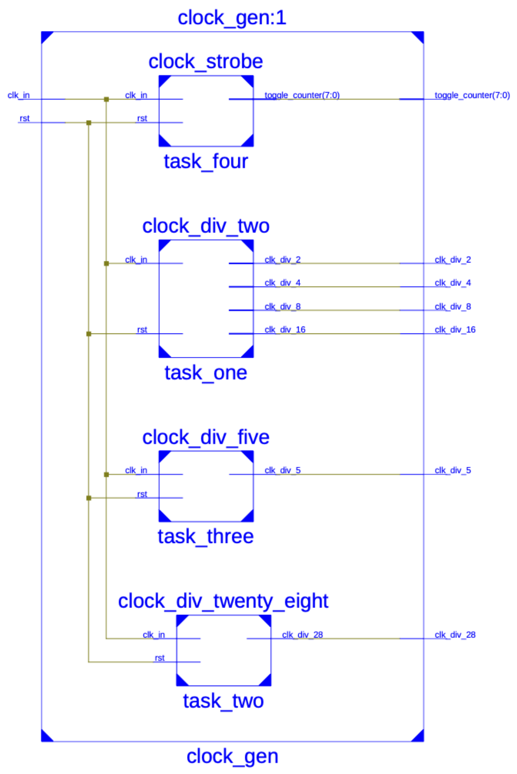


Figure 6: *clock_gen* schematic.

6.3 ISE Design Summary

clock_gen Project Status (01/21/2021 - 22:39:16)			
Project File:	clock_gen.xise	**Parser Errors:**	No Errors
Module Name:	clock_gen	**Implementation State:**	Placed and Routed
Target Device:	xc6slx16-3csg324	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	No Warnings
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Device Utilization Summary [-]				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	19	18,224	1%	
Number used as Flip Flops	19			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	15	9,112	1%	
Number used as logic	15	9,112	1%	
Number using O6 output only	9			
Number using O5 output only	0			
Number using O5 and O6	6			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number of occupied Slices	8	2,278	1%	
Number of MUXCYs used	0	4,556	0%	
Number of LUT Flip Flop pairs used	16			
Number with an unused Flip Flop	2	16	12%	
Number with an unused LUT	1	16	6%	
Number of fully used LUT-FF pairs	13	16	81%	
Number of unique control sets	3			
Number of slice register sites lost to control set restrictions	13	18,224	1%	
Number of bonded IOBs	16	232	6%	
Number of RAMB16BWERs	0	32	0%	
Number of RAMB8BWERs	0	64	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			

Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	4	0%	
Number of ILOGIC2/ISERDES2s	0	248	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	248	0%	
Number of OLOGIC2/OSERDES2s	0	248	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	128	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	32	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	2	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	4.39			

Performance Summary			[-]
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Thu Jan 21 22:38:42 2021	0	0	4 Infos (4 new)
Translation Report	Current	Thu Jan 21 22:38:54 2021	0	0	0
Map Report	Current	Thu Jan 21 22:39:02 2021	0	0	6 Infos (0 new)
Place and Route Report	Current	Thu Jan 21 22:39:08 2021	0	0	3 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Thu Jan 21 22:39:13 2021	0	0	4 Infos (0 new)
Bitgen Report					

Secondary Reports		[-]
Report Name	**Status**	**Generated**
ISIM Simulator Log	Out of Date	Thu Jan 21 22:38:31 2021

Date Generated: 01/21/2021 - 22:39:16

7 Conclusion

In this lab I was able to successfully implement divide-by-2^n clocks, even division clocks, odd-division clocks, pulses, strobes, and a glitchy counter.

The biggest difficulty I had with this lab was understanding the odd-division clocks. However, after doing my own research on the subject, and drawing out the clock signals by hand, I was able to understand how and why they worked, as well as how to implement them.

Overall, this lab was a great introduction to clocking and implementing sequential circuits. I don't have any suggestions for improvement at this time.