Lab 2 Project Report

Ethan Keshishian

CS M152A Lab1

Introduction:

The goal of Lab 2 is to implement 9 tasks that generate various clock waveforms. This is done with Verilog and the Xilinx software, which helps write the code as well as generate the waveforms. The design has a module called clock_gen at the top level, and multiple submodules that help implement each task. This keeps the code for each task separate. There is a submodule for each task except tasks 4, 5, and 6, which are grouped together under one submodule. This is in line with the example design given in the lab document. The names and designs of both module and submodule match the provided design for simplicity. My final implementation handles each task and has waveforms that match the solution waveforms. Additionally, the waveforms start at the same time as the example waveforms.

Module design:

The top module is a module named clock_gen that handles all nine tasks be delegating them to submodules. The top module takes in the clk_in and rst variables which handle sequential logic, and outputs 12 variables. Within clock_gen, there are seven submodules defined to handle the tasks, and each one of these accepts the clk_in, rst, and other output variables it will modify. These seven submodules are implemented next.

1. module clock_div_two(clk_in, rst, clk_div_2, clk_div_4, clk_div_8, clk_div_16)

This is the first module that handles task 1. It uses clk_in and rst as input, and outputs clk_div_2, clk_div_4, clk_div_8, and clk_div_16.

This module uses the clock to create waveforms that are active in frequency slower than the clock. For example, clk_div_2 divides the clock's frequency in half. To implement this, there is a 4-bit counter which is implemented with an always block and an internal register (called a). If rst is enabled, the counter remains at 0. Otherwise, it will increment the register by 1 with each clock cycle. This counter has no upper bound; it will overflow and restart one cycle after reaching 4'b1111. The outputs of this module are wires. The clock output is obtained by assignment to each bit of the internal register used by the counter. clk_div_2 is a[0], clk_div_4 is a[1], clk_div_8 is a[2], and clk_div_16 is a[3].

2. module clock_div_thirty_two(clk_in, rst, clk_div_32)

This is the module that handles task 2. It uses clk_in and rst as input, and outputs clk_div_32.

This module divides the clock by 32. It similarly uses a 4-bit counter which is implemented with an always block and an internal register again called a. However, in this case, the bits cannot be assigned as before. The output is treated as a register, and the value of the register should flip every time a overflows. In order to accomplish this, a is assigned a default value of 0, and there is if statement that flips the value of a any time a is equal to 4'b111.

3. module clock_div_twenty_six(clk_in, rst, clk_div_26)

This is the module that handles task 3. It uses clk_in and rst as input, and outputs clk_div_26.

This module divides the clock by 26. It uses a 4-bit counter with an always block and has an internal register, a. This case is similar to task 2, but the clock has to manually be reset to 0 when it has counted 26 times. In order to accomplish this, the always block that contains the counter has been slightly modified and clk_div_26 is set to 0 at first. If the value of a is equal to 12, then clk_div_26 is inverted, and a is reset to 0. This is verifiable with the waveform.

4. module clock_div_three(clk_in, rst, clk_div_3, clk_pos, clk_neg)

This is the module that handles tasks 4, 5, and 6. It uses clk_in and rst as input, and outputs clk_div_3, clk_pos, and clk_neg.

This module handles the case where the clock is divided by 3. This is challenging, because 3 is an odd number, and this splits the clock's positive and negative edges. To handle it, there are two 4-bit counters, a and b, and they are responsible for handling the positive and negative edges, respectively. The positive edge is implemented with the a counter, similar to before. If the value of a is 2, it resets to 0. If a is equal to 1, clk_pos is 1, otherwise it's 0. This creates a waveform that is active for 1/3 of the time. clk_neg is obtained in an identical always block, except it activates on the negedge instead of posedge, and it uses the b register as its counter. This also creates a waveform that is active for 1/3 of the time. Since it's activated on the negedge instead, it's offset from the clk_pos waveform. Both are active at the same time 1/6 of the time. Finally then, the clk_div_3 wire is obtained by calculating clk_pos | clk_neg. Since they are offset, they are active for 1/3 + 1/3 – 1/6 of the time, which is equal to 1/2 of the time. They create a waveform that divides the clock's frequency by 3.

5. module clock_div_five(clk_in, rst, clk_div_5)

This is the module that handles task 7. It uses clk_in and rst as input, and outputs clk_div_5.

This module handles the case where the clock is divided by 5. This is slightly more complicated than the divide by 3 case. In this case, there is again two 4-bit counters, a and b. There are also internal variables clk_pos and clk_neg initialized to 0. In the a always block which runs on the posedge, the counter is reset when a is equal to 4. If a is equal to 2 or 3, clk_pos is equal to 1. Otherwise, it's equal to 0. This creates a variable that's active 2/5 of the time. Similarly, the second counter is implemented with an always block that activates on the negative edge of clk_in. It also resets when b is equal to 4. If b is equal to 2, clk_neg is 1.

Otherwise, it’s equal to 0. This creates a variable that’s active 1/5 of the time and slightly offset. This offset means both clk_pos and clk_neg are active 1/10 of the time. Finally, the clk_div_5 wire is assigned to clk_pos | clk_neg. Mathematically this means they are active 2/5 + 1/5 – 1/10 of the time, which is equal to 5/10 or 1/2 of the time.

6. module clock_pulse(clk_in, rst, clk_div)

This is the module that handles task 8. It uses clk_in and rst as input, and outputs clk_div as output.

This module handles the case where the clock is divided by 200. To handle this, I use a 7-bit counter to count to 100. The implementation is similar to the others, except register a is larger. If a is equal to 99, then a is reset to 0. If a is equal to 1, an internal variable called flip_bit is equal to 1. Otherwise, flip_bit is equal to 0. Then, in a separate always block that also activates on the positive edge of clk_in, clk_div is being inverted when flip_bit is equal to 1. This give the correct output that matches the waveform.

7. module clock_strobe(clk_in, rst, toggle_counter)

This is the module that handles task 9. It uses clk_in and rst as input, and outputs toggle_counter as output.

This module counts up by 3 every three cycles and subtracts by 5 when the cycle ends. This is handles with a 3-bit counter called a in an always block, which also handles the changes to toggle_bit. If a is equal to 3, then a is reset to 0, and 5 is subtracted from toggle_counter. Otherwise, toggle_counter is incremented by 3. This matches the given waveform.

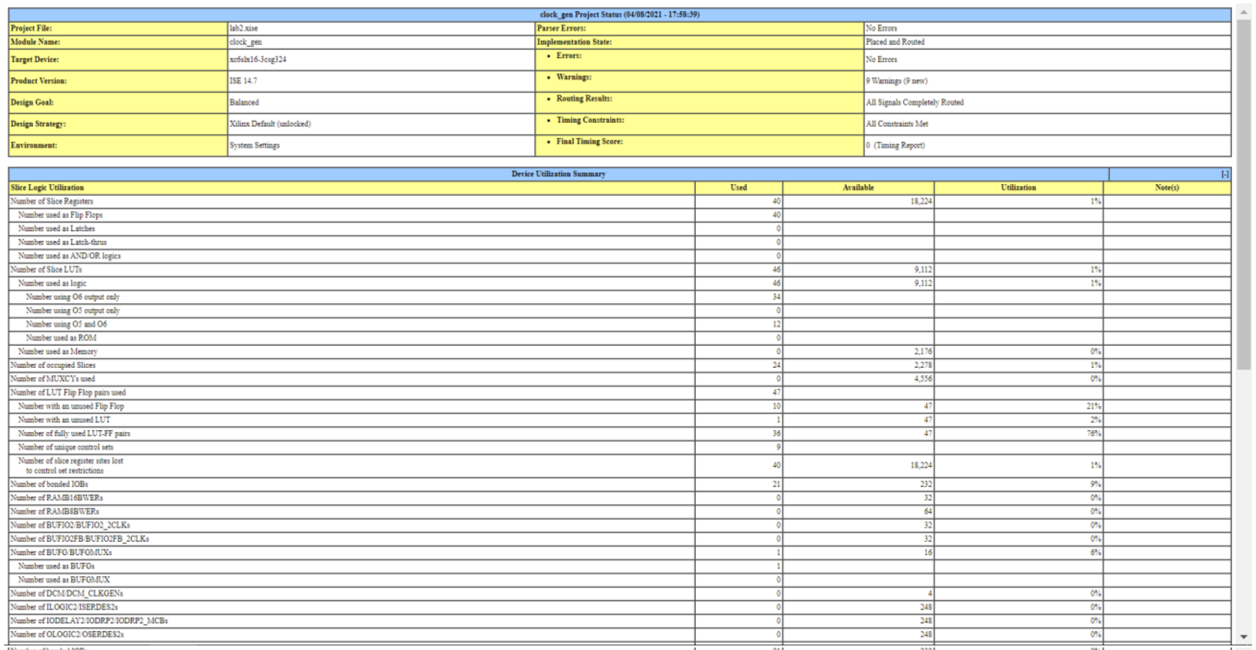
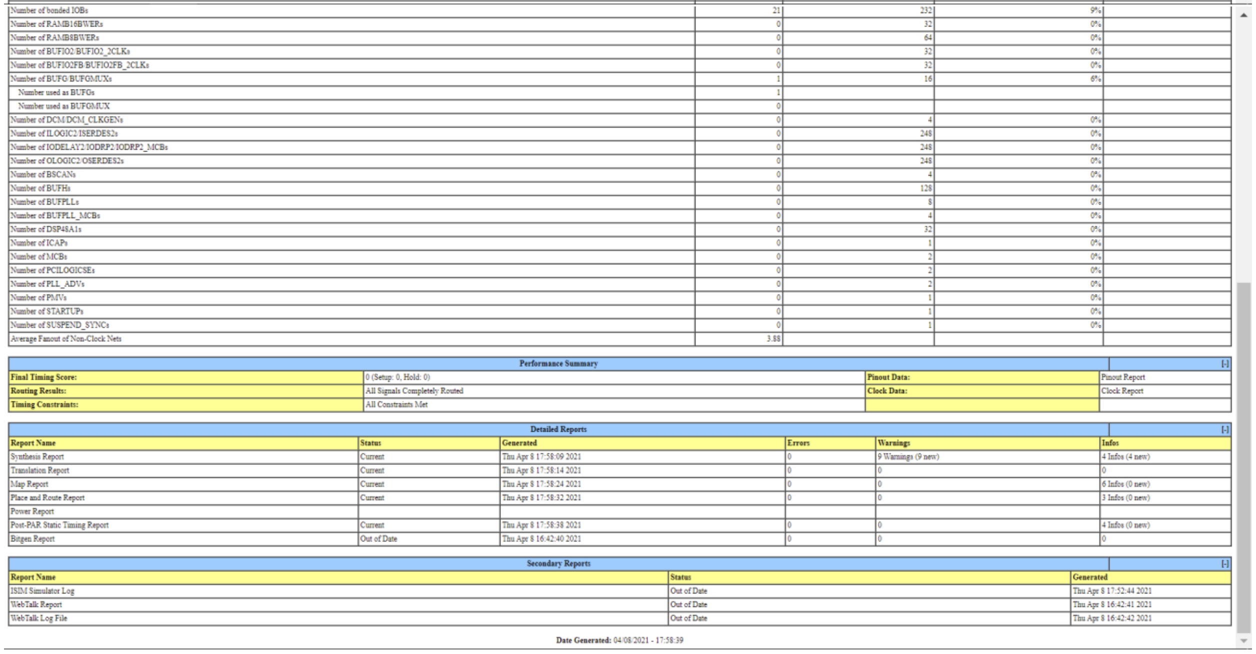
Testbench design:

My testbench tests the module by setting rst to 1, then setting rst to 0 and waiting. This ensures that the output is correct during reset and after. I confirmed that the module was working correctly by examining the waveform output and comparing it to the provided waveforms. My waveforms match the given ones identically, starting at the same point. My waveforms are provided below.

Conclusion:

Overall, my implementation uses seven submodules and one top module to handle each task. It outputs twelve variables, as described above. There were multiple difficulties I ran into with the project. I was confused by the implementation of tasks 4, 5, and 6, thinking the output of task 4 was clk_pos, task 5 was clk_neg, and task 6 was clk_div_3, because of their ordering. I initially implemented this task differently too, directly counting clk_in instead of using the positive edge or the negative edge. Overall, I enjoyed the project, and found myself learning a lot from the challenges I faced.

ISE Design Overview Summary Report:

clock_gen Project Status (04/08/2021 - 17:58:39)			
Project File:	lab2.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:**	9 Warnings (9 new)
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	40	18,224	1%	
Number used as Flip Flops	40			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	46	9,112	1%	
Number used as logic	46	9,112	1%	
Number using O6 output only	34			
Number using O5 output only	0			
Number using O5 and O6	12			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number of occupied Slices	24	2,278	1%	
Number of MUXCYs used	0	4,556	0%	
Number of LUT Flip Flop pairs used	47			
Number with an unused Flip Flop	10	47	21%	
Number with an unused LUT	1	47	2%	
Number of fully used LUT-FF pairs	36	47	76%	
Number of unique control sets	9			
Number of slice register sites lost to control set restrictions	40	18,224	1%	
Number of bonded IOBs	21	232	9%	
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	3.88			

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		

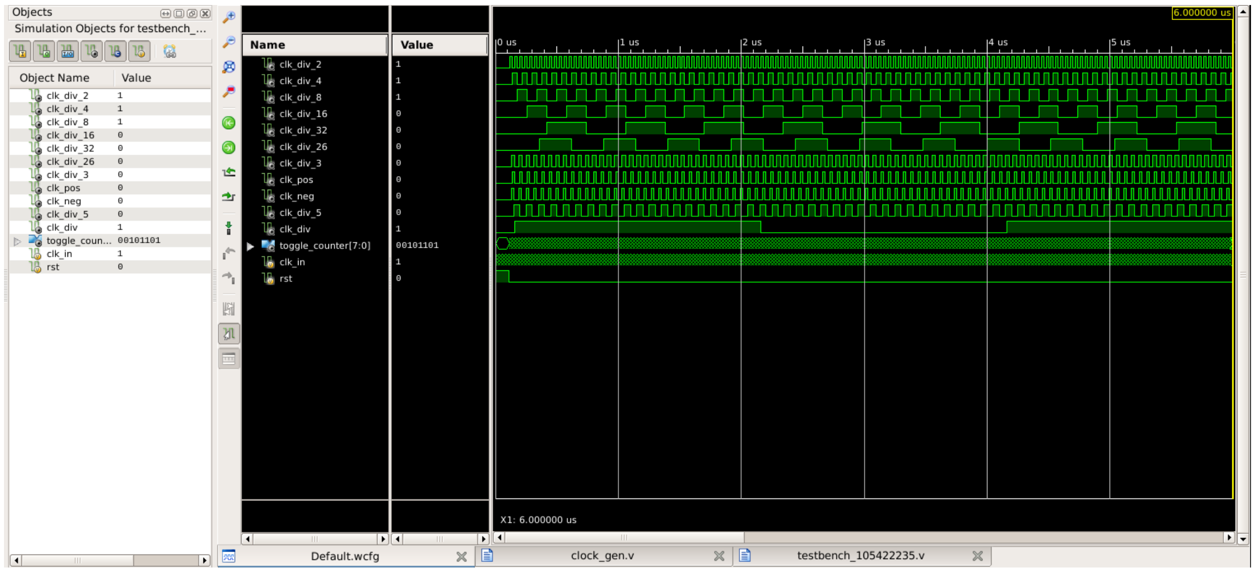
Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Thu Apr 8 17:58:09 2021	0	9 Warnings (9 new)	4 Infos (4 new)
Translation Report	Current	Thu Apr 8 17:58:14 2021	0	0	0
Map Report	Current	Thu Apr 8 17:58:24 2021	0	0	6 Infos (0 new)
Place and Route Report	Current	Thu Apr 8 17:58:32 2021	0	0	3 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Thu Apr 8 17:58:38 2021	0	0	4 Infos (0 new)
Bitgen Report	Out of Date	Thu Apr 8 16:42:40 2021	0	0	0

Secondary Reports		[-]
Report Name	**Status**	**Generated**
ISIM Simulator Log	Out of Date	Thu Apr 8 17:52:44 2021
WebTalk Report	Out of Date	Thu Apr 8 16:42:41 2021
WebTalk Log File	Out of Date	Thu Apr 8 16:42:42 2021

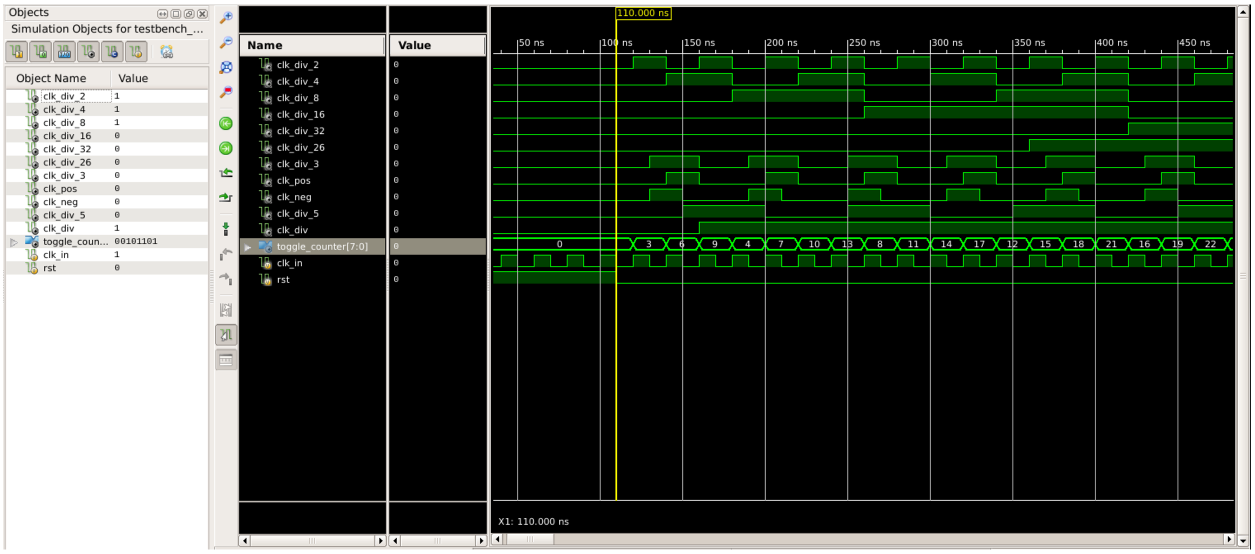
Date Generated: 04/08/2021 - 17:58:39

Simulation Output:

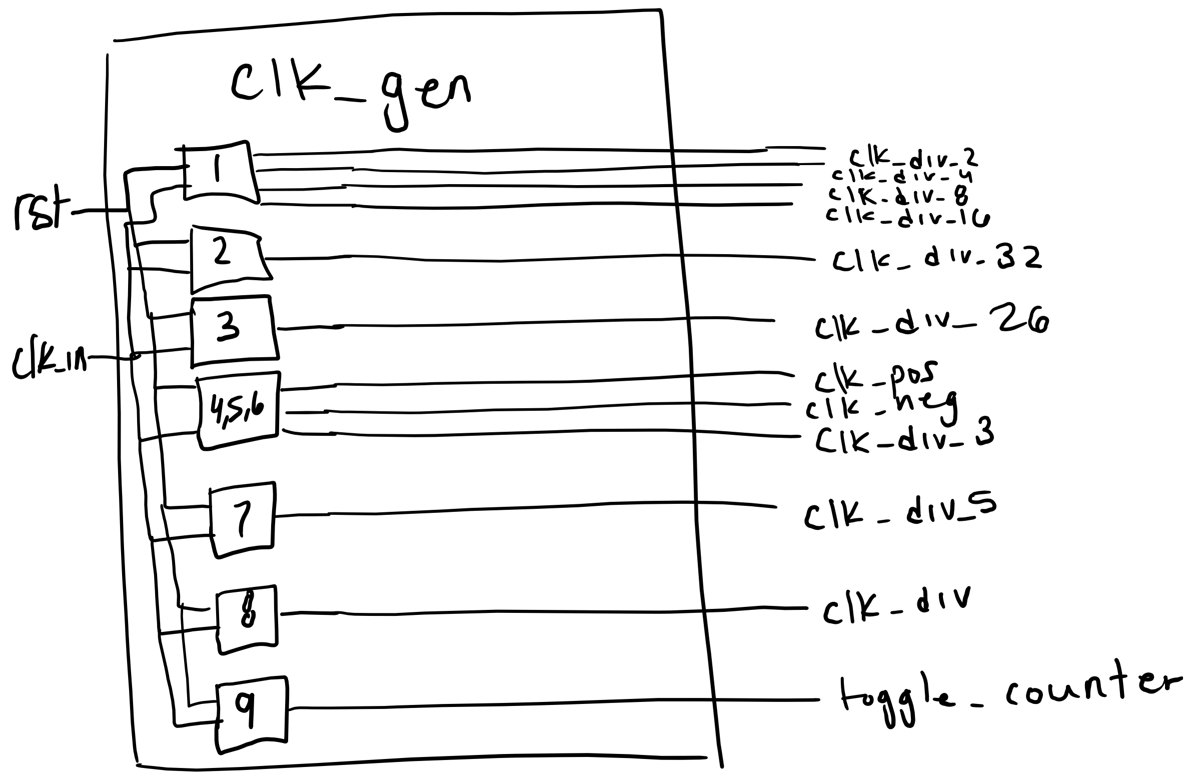
Zoomed out:



Zoomed in:



Hand-drawn Design Schematic:


clk_gen
rst
clk_in
1
2
3
4,5,6
7
8
9
clk_div_2
clk_div_4
clk_div_8
clk_div_16
clk_div_32
clk_div_26
clk_pos
clk_neg
clk_div_3
clk_div_5
clk_div
toggle_counter