CS M152A - Lab 2, Clock Design Methodology

Name: Sum Li

UID: 505146702

Due Date: November 8, 2020

TA: Mohit Garg

1) Introduction and Requirement

For this lab, the goal is to the Xilinx ISE program to design and test different sequential circuits for various clock waveforms on a digital system. All of the implemented submodules in this lab relate to the concept of clocking a system and the methods to generate a specific type of clocking from a system clock. After completing the lab, I have compared different clock waveforms diagrams that are generated by the circuit. Each of the submodules is given a system clock and a reset signal as given input. The output of the submodule is the derived clock based on specific design requirements.

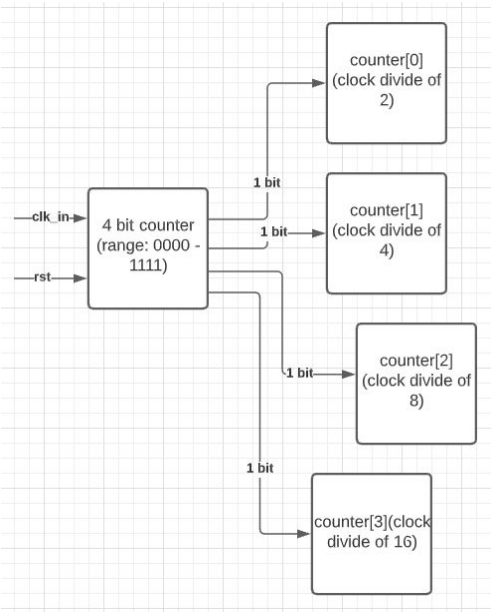
There are a total of 4 main design tasks and 5 observation tasks in this lab. The four main design tasks: (1) divide by power of 2 clocks, (2) even division clock using counters, (3) odd division clock using, (4) single pulse strobe. All the implementations of the submodules are based on modification of the verilog implementation of the 4 bit counter. For each of the submodules, I used the corresponding operators, edge sensitive always block and non block assignment operator (<=) during the translation of the schematics for gates and flip flops.

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

Figure 1: Verilog implementation of a 4 bit counter

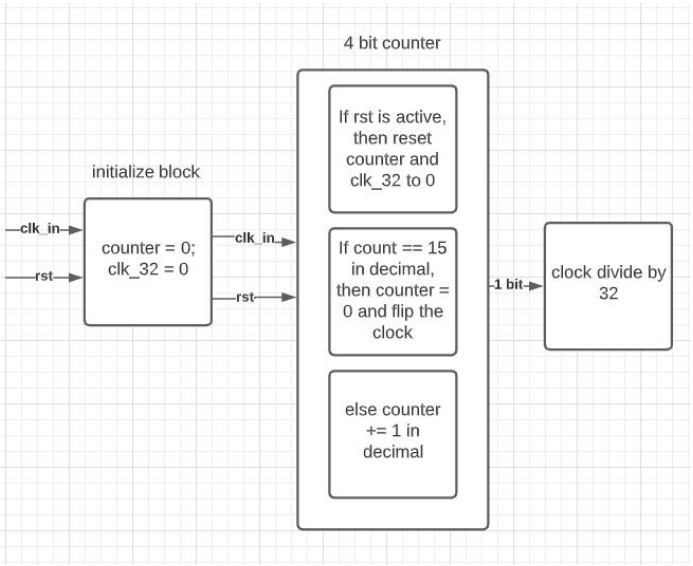
2) Design Description

1. Clock Divider by Power of 2s:

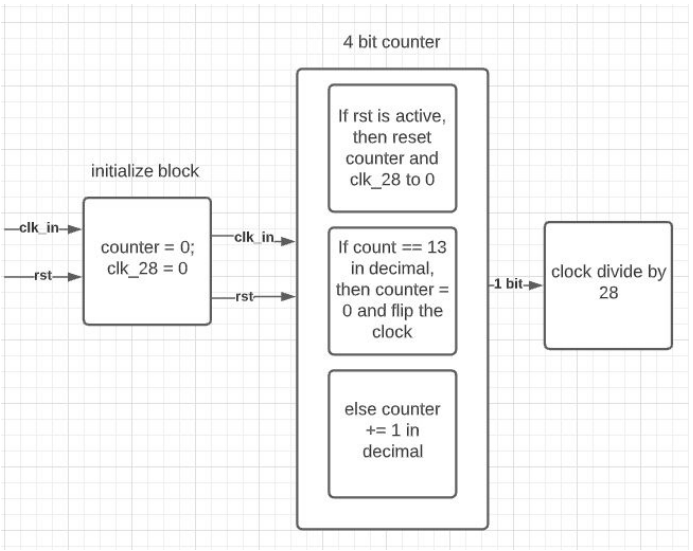


The first design task **(Task 1)** of the lab is to verify the 4 bit counter which is assigning 1 bit wires to each of the bits from the 4 bit counter. If I compare the 4 bit counter with the global master clock, I am able to obtain a clock divider by extracting each of the bits from the 4 bit counter. The least significant bit (LSB, first bit) of the counter is a direct division of 2. The second bit of the counter is a direct division of 4. The third bit of the counter is a direct division of 8. The fourth bit of the counter is a direct division of 16. I created 4 output wires to extract the values of the 4 corresponding bits from the counter.

2. Even Division Clock Using Counters:



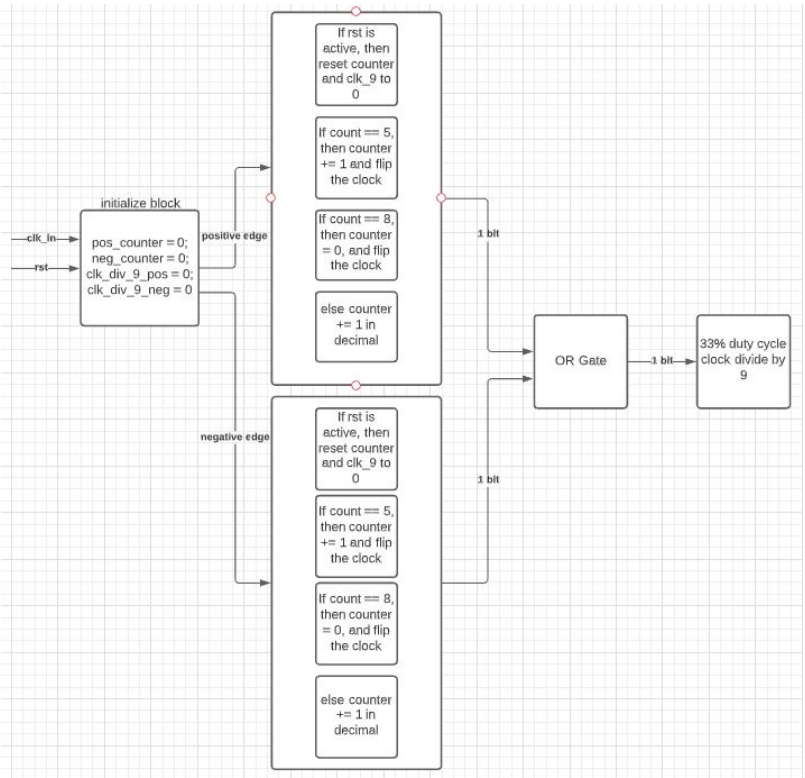
The first observation task **(Task 2)** of the lab is to generate the divide by 32 clocks by flipping the output clock on every counter overflow in the 4 bit counter design. The range of the counter is from 0 to 15 in decimal which is 0000 to 1111 in binary representation of 4 bits. When the counter reach 15 in decimal, I flipped the clock value immediately during the half cycle to generate a clock period that is 32 times larger of the master clock: 16 * 2 = 32. In the normal cases, the counter just increments by 1 in decimal.



The second design task **(Task 3)** of the lab is to generate the divide by 28 clock by modifying the counter resets to 0 in the 4 bit counter design. The range of the counter is from 0 to 15 decimal which is 0000 to 1111 in binary representation of 4 bits. When the counter reached

13 in decimal, I flipped the clock value immediately during the half cycle to generate a clock period that is 28 times larger of the master clock: 14 * 2 = 28. In the normal cases, the counter just increments by 1 in decimal.

3. Odd Division Clock Using Counters:

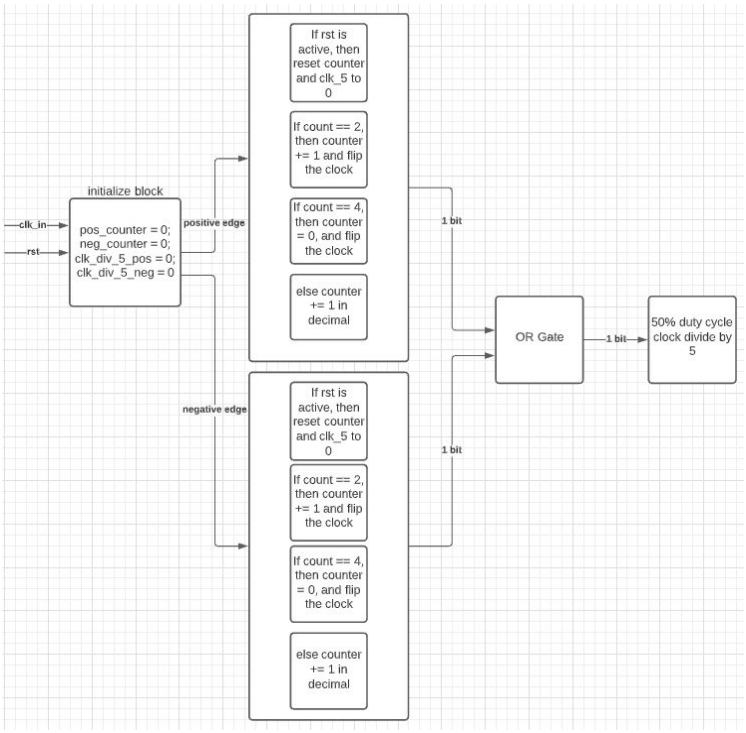


The second observation task **(Task 4)** of the lab is to generate a 33% duty cycle with the use of if else statements and the 4 bit counter with a positive edge. I implemented a divide by 9 clock with an always block that triggers on the raising edge. It is because the 33% cycle refers to 3/9 = 0.33333333… = 33% of the entire clock period is gonna be high while 6/9 = 66.666666% of the entire clock period is gonna be low. The initial value of the output clock is 0 and is gonna be low first then high later. When the 4 bit counter reaches 5 in decimal, the output clock is flipped to 1. When the 4 bit counter reaches 8 in decimal, the output clock is flipped to generate a clock period that is 9 times larger than the master clock. In the normal case, the 4 bit counter just increments by 1 in decimal when there is a rising edge which is the flip of the master clock.

The third observation task **(Task 5)** of the lab is to generate a 33% duty cycle with the use of if else statements and the 4 bit counter with a negative edge. I implemented a divide by 9 clock with an always block that triggers on the falling edge. It is because the 33% cycle refers to

3/9 = 0.33333333… = 33% of the entire clock period is gonna be high while 6/9 = 66.666666% of the entire clock period is gonna be low. The initial value of the output clock is 0 and is gonna be low first then high later. When the 4 bit counter reaches 5 in decimal, the output clock is flipped to 1. When the 4 bit counter reaches 8 in decimal, the output clock is flipped to generate a clock period that is 9 times larger than the master clock. In the normal case, the 4 bit counter just increments by 1 in decimal when there is a falling edge which is the flip of the master clock.

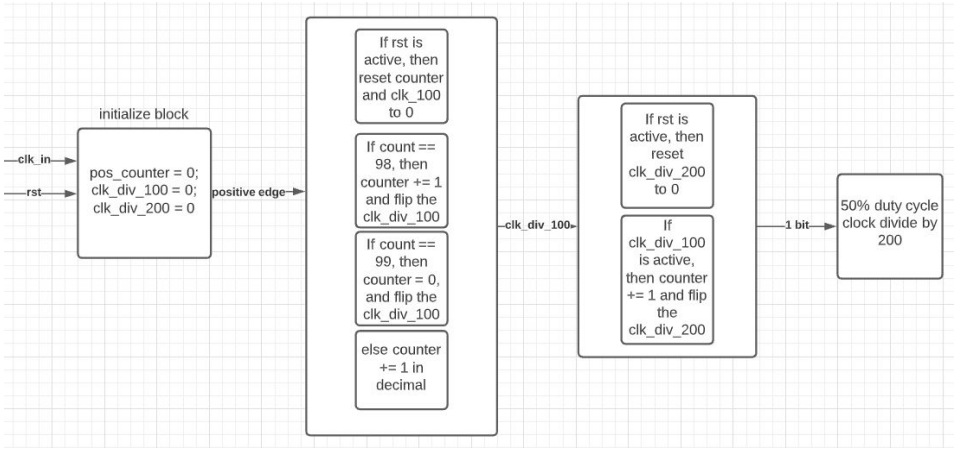
The fourth observation task **(Task 6)** of the lab is to assign a wire that takes the logical OR of the two 33% duty cycle clocks that triggers on the raising or falling of the edge of the master clock. I created a temporary register to store the logical OR of the two temporary values I had for the two 33% duty cycle clocks. Then, I assigned the final value to the final variable that stores the logical OR of the two clocks.



The third design task **(Task 7)** of the lab is to generate a 50% duty cycle divide by 5 clock with the use of if else statements and the 4 bit counter. I needed to flip the clock during the half cycle which is 2.5, but the 4 bit counter is incremented by 1 in decimal under the raising edge of the master clock. Therefore, it is not possible to flip the clock according to that. However, I used the same design idea from Task 4 to 6 to implement two 2/3 duty cycle divide by 5 clocks and then find the logical OR of the two clocks to get the 50% duty cycle divide by 5

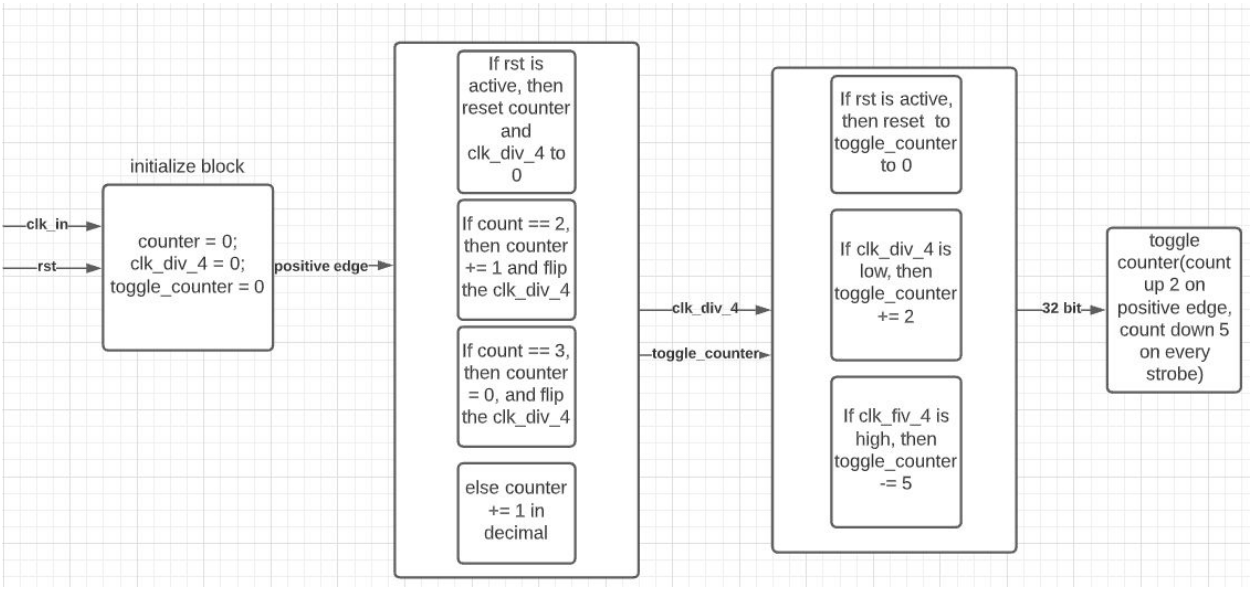
clock. One of the divide by 5 clock is gonna trigger on the rising edge while the other divide by 5 clock is gonna trigger on the falling edge. A 50% duty cycle refers to half the entire clock period is gonna be high while the other half the entire clock period is gonna be low. The initial value of the output clock is 0 and is gonna be low first then high later. When the 4 bit counter reaches 2 in decimal, the output clock is flipped to 1. When the 4 bit counter reaches 4 in decimal, the output clock is flipped to generate a clock period that is 5 times larger than the master clock. In the normal case of a positive edge clock, the 4 bit counter just increments by 1 in decimal when there is a rising edge which is the flip of the master clock. In the normal case of a negative edge clock, the 4 bit counter just increments by 1 in decimal when there is a falling edge which is the flip of the master clock. At the end, I took the logical OR of the two divide by 5 clocks because when the two clocks overlap on high, it divides a cycle into half.

4. **Pulse/Strobes:**



The fifth observation task **(Task 8)** of the lab is to generate a 1% duty cycle divide by 100 clock with the use of if else statements and a 7 bit counter with positive edge. The range of the counter is from 0 to 99 in decimal which is 000 000 0 to 110 001 1 in binary representation of 7 bits. The initial value of the output clock is 0 and is gonna be low first then high later. When the 7 bit counter reaches 98 in decimal, the output clock is flipped to 1 for the 1% duty cycle. When the 7 bit counter reaches 99 in decimal, the output clock is flipped to generate the corresponding clock period. In the normal case, the 7 bit counter just increments by 1 in decimal when there is a rising edge which is the flip of the master clock. Then, I needed to use the 1% duty cycle divide by 100 clock to implement a 50% duty cycle divide by 200 clock by creating a second always block that runs on the system clock and flip the divide by 200 clock when the

divide by 100 clock pulse is active. Therefore, I used an if else statement in the second always block to check whether the divide by 100 clock has a rising edge or not. If the divide by 100 clock had an rising edge, then I flip the divide by 200 clock. It is because divide by 200 clock is 200 times larger than the master clock which is also 2 * divide by 100 clock.



The fourth design task **(Task 9)** of the lab is to implement a divide by 4 strobe which is an 8 bit counter that counts up by 2 for every positive edge of the master clock, but subtracts by 5 on every strobe. I used the same design idea from task 8 and I first implemented a 25% duty cycle divide by 4 clock and then implemented a second always block to check if there is a rising edge of the divide by 4 clock to increment 2 or decrement 5 for the 8 bit counter. For the implementation of the divide by 4 clock, I used if else statements in the first always block and a 4 bit counter. The initial value of the output clock is 0 and is gonna be low first then high later. When the 4 bit counter reaches 2 in decimal, the output clock is flipped to 1 for the 25% duty cycle. When the 4 bit counter reaches 3 in decimal, the output clock is flipped to generate the corresponding clock period. In the normal case, the 4 bit counter just increments by 1 in decimal when there is a rising edge which is the flip of the master clock. For the second always block, I checked if there is a falling edge of the divide by 4 clock, the 32 bit toggle counter will be incremented by 2 in decimal. If there is a rising edge of the divide by 4 clock, the 32 bit toggle counter will be decremented by 5 in decimal. The combination of the first and second always block creates the 8 bit counter.

3) Simulation Documentation

For the testbench file, the input clock frequency is 100Mhz which is 10 ns for the clock period of the master clock for all 9 tasks in this lab. I first defined the given inputs which are the input clock and the reset signal. Then, I defined the given outputs which are clk_div_2, clk_div_4, clk_div_8, clk_div_16, clk_div_28, clk_div_5, toggle_counter. After that, I created the corresponding unit under testing. I have created an initial block and it takes 50 ns for the clk_in and reset signal to be reset. After that, I set the reset signal to high which is 1. Then, I waited for another 30 ns. Then, I put the reset signal to low which is 0. I have also created separate testbench files for all the observation tasks and the set up is the same as described above except the given output variables to store the final value of the clock the task specified.

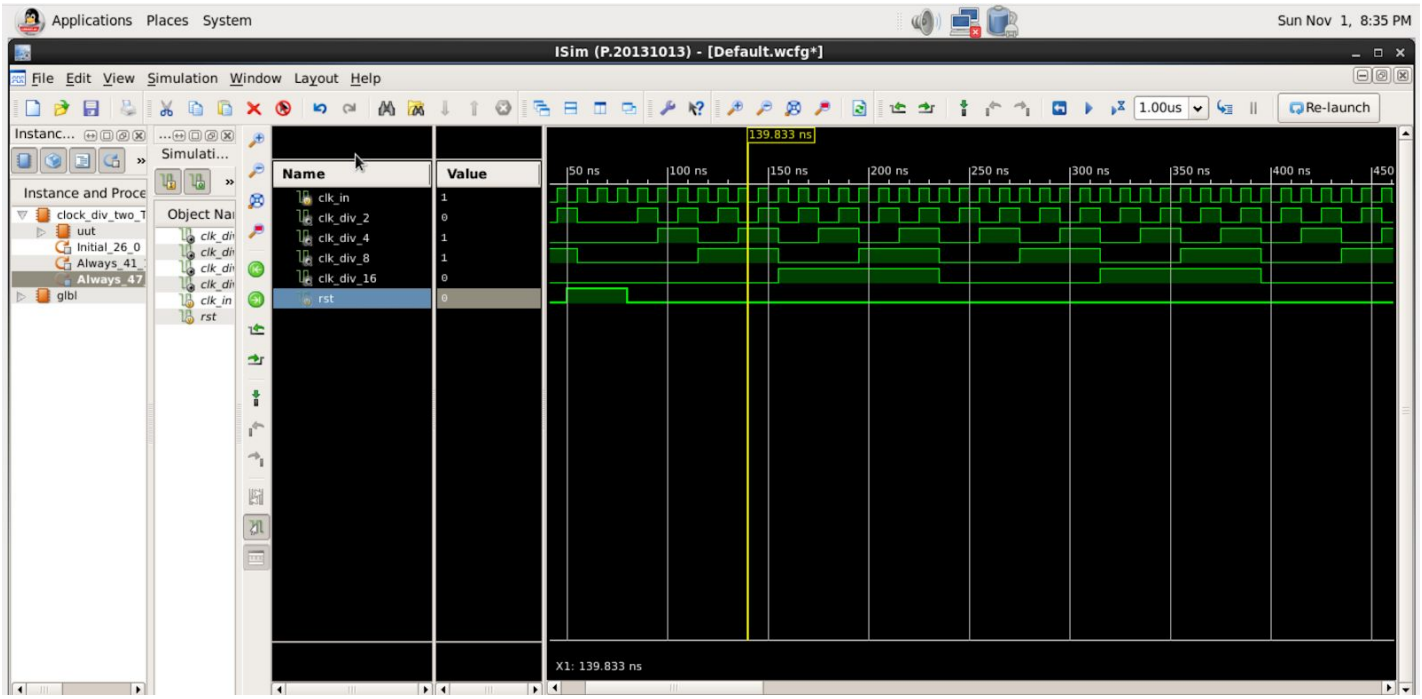


Figure: Task 1 - Clock Divider by Power of 2s

According to the waveform diagram, the clock period of the master clock (clk_in) is 10 ns. The clock period of the divide by 2 clock is 20 ns. The clock period of the divide by 4 clock is 40 ns. The clock period of the divide by 8 clock is 80 ns. The clock period of the divide by 16 clock is 160 ns.

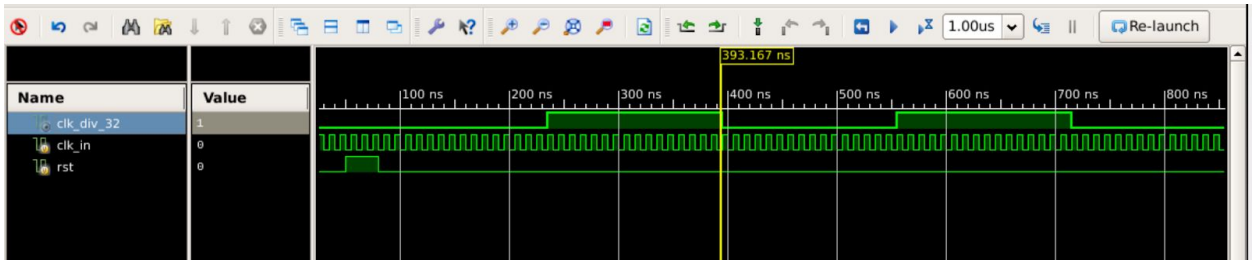


Figure: Task 2 - Generate the divide by 32 clocks from a 4 bit counter design

According to the waveform diagram, the clock per the clock period of the master clock (clk_in) is 10 ns. The clock period of the divide by 32 clock (clk_div_32) is 320 ns which can be calculated as 713 ns - 393 ns = 320 ns.

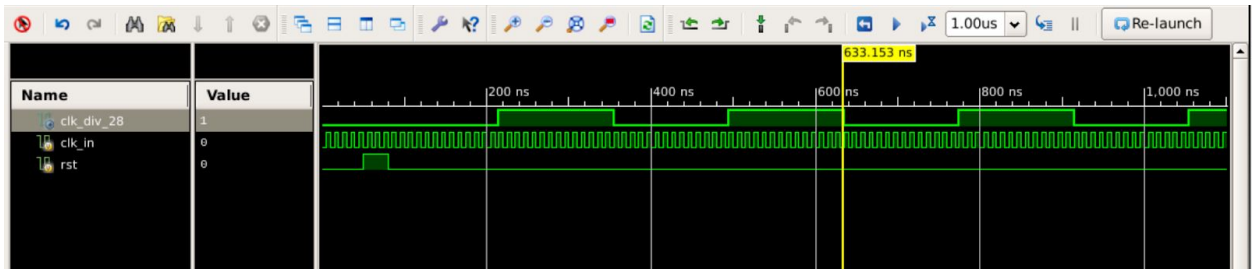


Figure: Task 3 - Generate the divide by 28 clocks from a 4 bit counter design

According to the waveform diagram, the clock per the clock period of the master clock (clk_in) is 10 ns. The clock period of the divide by 28 clock (clk_div_28) is 280 ns which can be calculated as 913 ns - 633 ns = 280 ns.

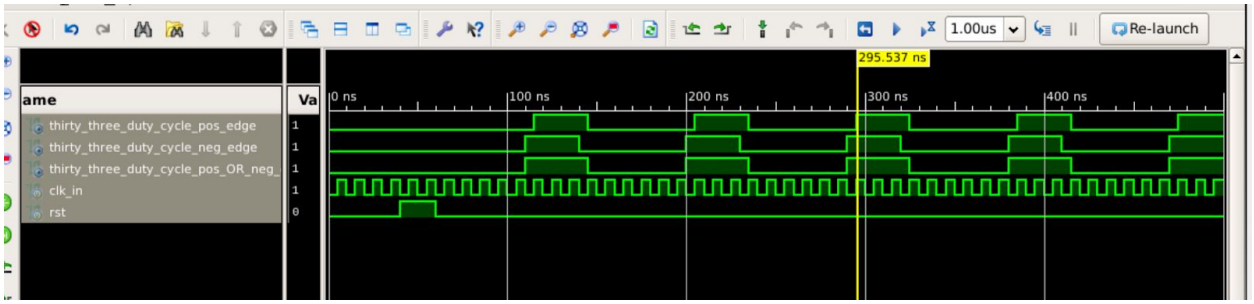


Figure: Task 4 - 33% duty cycle divide by 9 clock that trigger on raising edge from a 4 bit counter design

According to the waveform diagram, the clock per the clock period of the master clock (clk_in) is 10 ns. The clock period of the divide by 9 clock (clk_div_9) on the rising edge is 90 ns which can be calculated as 295 ns - 205 ns = 90 ns. The clock period of a high signal is shown as:

235 ns - 205 ns = 30 ns. The 33% duty cycle can be calculated as (30 ns / 90 ns) * 100% = 33%.

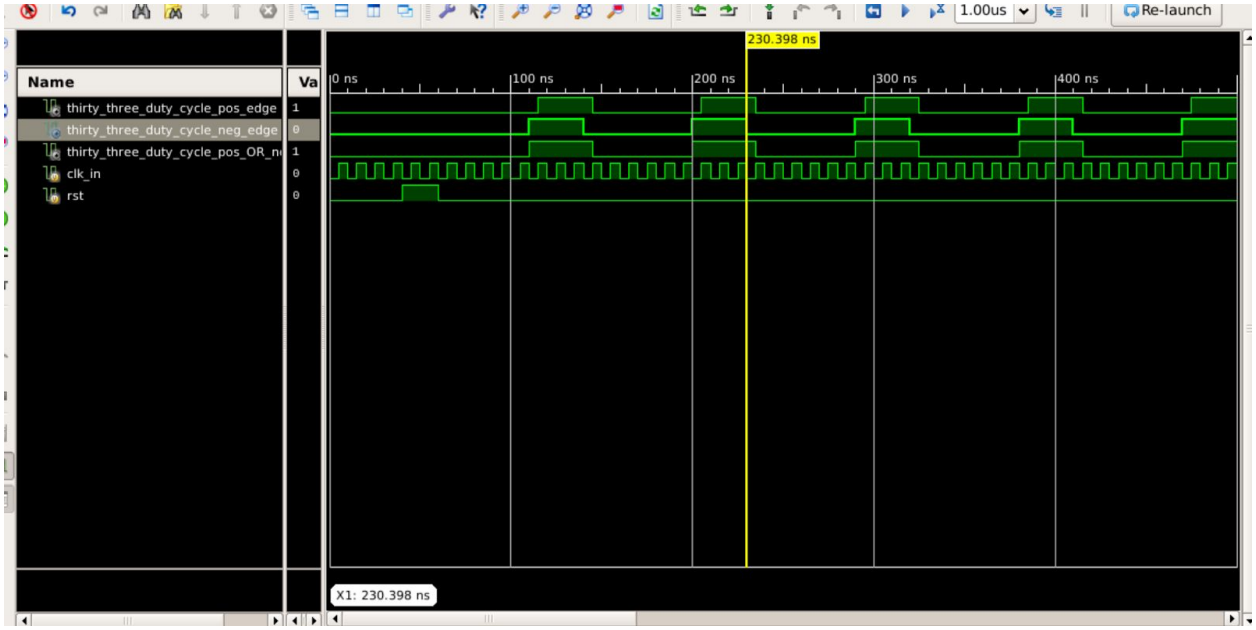


Figure: Task 5 - 33% duty cycle divide by 9 clock that trigger on falling edge from a 4 bit counter design

According to the waveform diagram, the clock per the clock period of the master clock (clk_in) is 10 ns. The clock period of the divide by 9 clock (clk_div_9) on the rising edge is 90 ns which can be calculated as 290 ns - 200 ns = 90 ns. The clock period of a high signal is shown as: 230 ns - 200 ns = 30 ns. The 33% duty cycle can be calculated as (30 ns / 90 ns) * 100% = 33%.

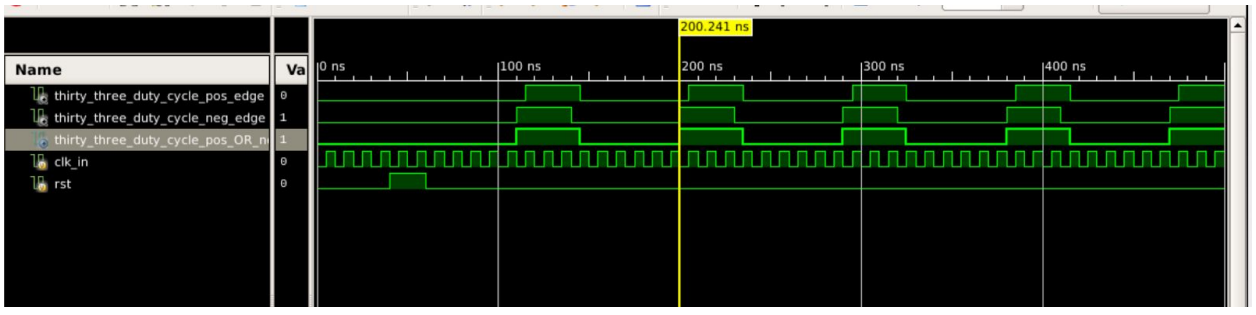


Figure Task 6 - Logical OR of the two 33% duty cycle clocks (raising and falling edge)

The clock period of a high signal that is overlap by the two 33% duty cycle clocks is: 235 ns - 200 ns = 35 ns. Therefore, the overall duty cycle can be calculated as (35 ns / 90 ns) * 100 % = 38%.

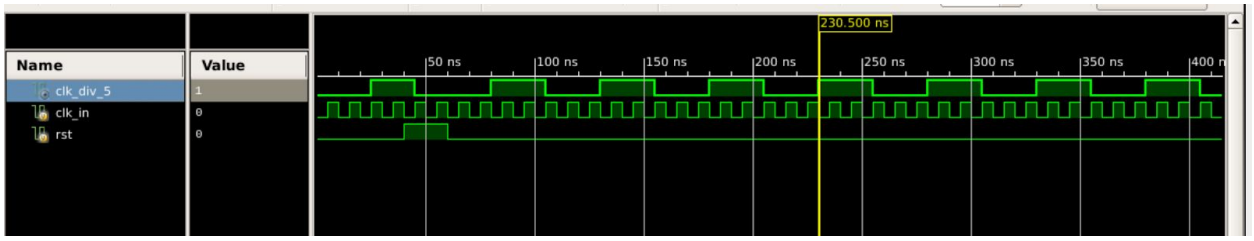


Figure Task 7 - 50% duty cycle divide by 5 clock

According to the waveform diagram, the clock per the clock period of the master clock (clk_in) is 10 ns. The clock period of the divide by 5 clock (clk_div_5) on the rising edge is 50 ns which can be calculated as 305 ns - 255 ns = 50 ns. The clock period of a high signal is shown as: 255 ns - 230 ns = 25 ns. The 50% duty cycle can be calculated as (25 ns / 50 ns) * 100% = 50%.

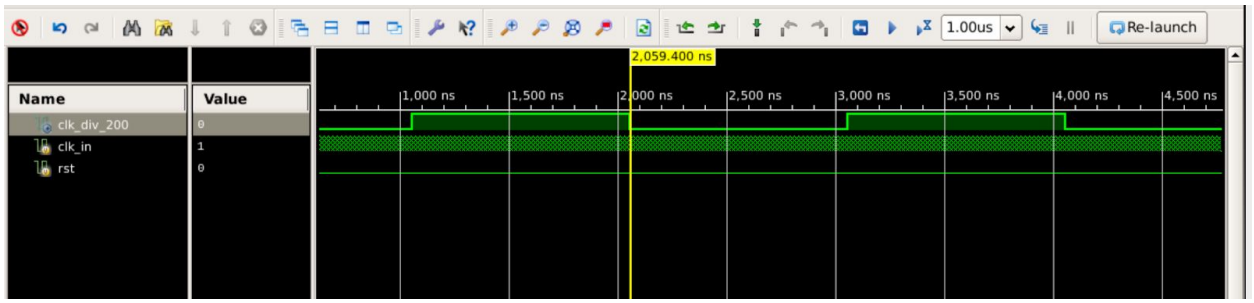


Figure: Task 8 - 50% duty cycle divide by 200 clock

According to the waveform diagram, the clock per the clock period of the master clock (clk_in) is 10 ns. The clock period of the divide by 200 clock (clk_div_200) on the rising edge is 2000 ns which can be calculated as 4059 ns - 2059 ns = 2000 ns. The clock period of a high signal is shown as: 4059 ns - 3059 ns = 1000 ns. The 50% duty cycle can be calculated as (1000 ns / 2000 ns) * 100% = 50%.

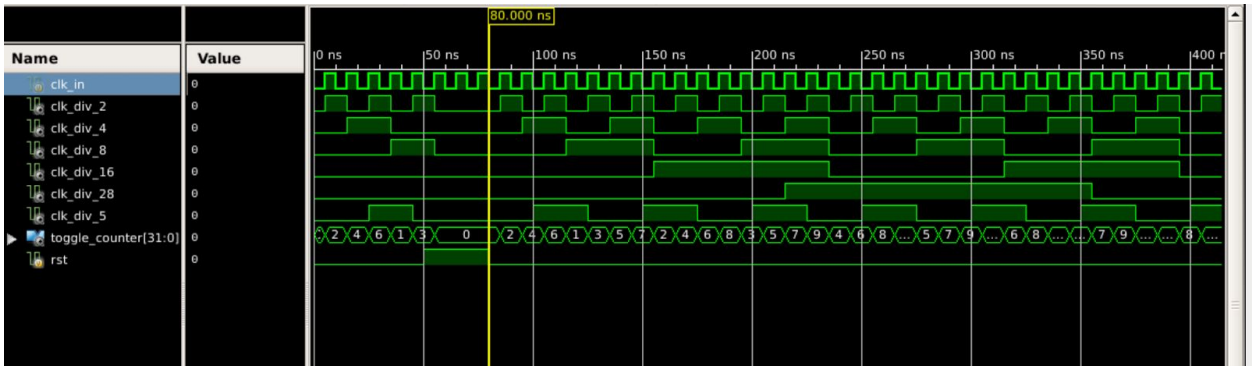


Figure: Task 9 - 8 bit counter that counts up by 2 on every positive edge of the master clock, but subtracts by 5 on every strobe

According to the waveform diagram, the 8 bit counter matches the expected sequence from the instruction which is incremented by 2 every cycle of the master clock. But, the counter also decrements by 5 every 4 cycles of the master clock.

ISE Design Overview Summary Report

clock_gen Project Status (11/04/2020 - 20:17:39)			
Project File:	clock_design.xise	**Parser Errors:**	No Errors
Module Name:	clock_gen	**Implementation State:**	Programming File Generated
Target Device:	xc6slx16-3csg324	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	5 Warnings (0 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	63	18,224	1%	
Number used as Flip Flops	63			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	51	9,112	1%	
Number used as logic	49	9,112	1%	
Number using O6 output only	37			
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 used as ROM	0			
Number used as Memory	0	2,176	0%	
Number used exclusively as route-thrus	2			
Number with same-slice register load	0			
Number with same-slice carry load	2			
Number with other load	0			
Number of occupied Slices	21	2,278	1%	
Number of MUXCYs used	32	4,556	1%	
Number of LUT Flip Flop pairs used	52			
Number with an unused Flip Flop	0	52	0%	
Number with an unused LUT	1	52	1%	
Number of fully used LUT-FF pairs	51	52	98%	
Number of unique control sets	4			
Number of slice register sites lost to control set restrictions	17	18,224	1%	
Number of bonded IOBs	44	232	18%	
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 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	3.63			

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	Wed Nov 4 20:16:54 2020	0	5 Warnings (0 new)	6 Infos (0 new)
Translation Report	Current	Wed Nov 4 20:17:00 2020	0	0	0
Map Report	Current	Wed Nov 4 20:17:13 2020	0	0	6 Infos (0 new)
Place and Route Report	Current	Wed Nov 4 20:17:21 2020	0	0	3 Infos (0 new)

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	3.63			

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	Wed Nov 4 20:16:54 2020	0	5 Warnings (0 new)	6 Infos (0 new)
Translation Report	Current	Wed Nov 4 20:17:00 2020	0	0	0
Map Report	Current	Wed Nov 4 20:17:13 2020	0	0	6 Infos (0 new)
Place and Route Report	Current	Wed Nov 4 20:17:21 2020	0	0	3 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Wed Nov 4 20:17:27 2020	0	0	4 Infos (0 new)
Bitgen Report	Current	Wed Nov 4 20:17:36 2020	0	0	0

Secondary Reports		[-]
Report Name	**Status**	**Generated**
ISIM Simulator Log	Out of Date	Wed Nov 4 19:36:05 2020
WebTalk Report	Current	Wed Nov 4 20:17:38 2020
WebTalk Log File	Current	Wed Nov 4 20:17:38 2020

Date Generated: 11/04/2020 - 20:20:14

4) Conclusion

After I have finished this lab, I understand how to use Verilog and Xilinx ISE programs to design and test several sequential circuit submodules to compare various clock waveforms on a digital system. Through the lab, I had an opportunity to understand the concepts behind clocking a system and the techniques to generate various signal diagrams from a system clock. One of the difficulties that I have encountered is to figure out the relationship between the duty cycle and the clock period in order to create a specific type of clock for the circuit to use. Second, it takes me a while to figure out when during a cycle to flip the clock in order to divide into the clock period that I want.