

## CS M152A Lab 2: Clock Methodology

---

Ashley Zhu

UID: 505308582

Section: Lab 7

TA: Uneeb Rathore

QTR: Spring 2021

### Overview

In digital systems, **clock signals** are signals that oscillate periodically between a high and low (positive and negative) state in order to communicate to the system when to execute particular programmed functions. Unlike the enable bit that exists within some systems, the clock states should not be altered manually by the user and should follow a specific pattern. Setting a specific period of time for a clock is all the user needs to do in order to configure a "system clock". If a user hopes to alter this period depending on certain inputs, they can utilize **clock division** to do so. Clock division (in other words frequency division) includes reducing the frequency by which a clock is in its high state and in effect, increasing the period of the wave by the same factor. This is useful in both analog and digital contexts, as clocking allows a designer to run through all the phases of a program in a synchronous manner. Synchronous data transmission is frequently used in serial communication links like UART, USB, and Ethernet, as well as in embedded devices like traffic lights, monitor screens, stopwatches, and phones. In Verilog, we can recreate these physical systems with **counters** as substitutes for **flip-flops**. By taking advantage of sequential function loops in circuits, counters act as both factors by which clocks are divided into as well as sources of change for **duty cycles**.

In this lab, we were tasked to perform specific operations on the clock of a system:

Operation	Module Name (in file)
Divide by $2^n$ Clock	div_pow_2
Even Division Clock	div_32, div_26
Odd Division Clock	div_3, div_5, div_200
Glitchy counter	strobe

## Divide by $2^n$ Clock

### Task 1: Clock Division by 2, 4, 8, & 16

The learning objective of this module is that a more significant bit inside of a register changes at a different rate compared to less significant bits when incrementing. Each bit must go through all increments before its value before it changes between 0 and 1, making the rate by which it switches exponential. In this manner, by taking the bits themselves as the output of the clock, we can simulate clock division by factors of two.

In the table below, the times of each power of two's high states are recorded to demonstrate how the position of a bit correlates with the clock division factor.

*Clock Division with 10ns System Period*

Power	Corresponding Bit (4 bits)	High State (ns)
0 (Default)	None	5
1	0000	10
2	0000	20
3	0000	40
4	0000	80

The main case to keep track of in this part is whether or not the reset bit was set.

### Even Division Clocks with Counters

#### Task 2: Clock Division by 32

The second part of this section involved using registers that did not have enough bits for the power in question and performing division through the use of *counters*. Every time the counter reached *halfway* to 32 (15 in bits), the program would have to invert the output bit and continue from 0 again. This method succeeds because the amount of time the output bit is in a certain position actually doubles the total time spent in the high state for the adjusted clock. Every occurrence of a positive edge means a sustenance of the state that the

adjusted clock was previously in through both on and off system clock states until the counter reaches halfway to 32, and the output inverts.

### Task 3: Clock Division by 26

In a similar vein, dividing the system clock by 26 was done by inverting the output once it reached *halfway to 26*, or 13 (12 in bits). Since the methodology is a perfect replica of task 2, I will skip over explaining how this module was implemented.

### Odd Division Clock with Counters

A challenge in this section was altering the duty cycle of the waveforms. Simply changing the counter value to three didn't work as it only divided the clock by three, and the duty cycle remained the same. Therefore, a method that persistently (no matter which state the counter is in) had a 33% chance of being on was necessary. The method used in my work was bit shifting.

Given a 3-bit register with a single 1 in it, at any point in time, there is a 33% chance for a certain index of the register to have a 1 in it.

001  $\Rightarrow$  010  $\Rightarrow$  100  $\Rightarrow$  Reset

Every shift of the register places the 1 in a new location, so if the output bit is set to any one of the bits in the register, it now has a 33% chance of being on at any time. This successfully reproduced a waveform with a duty cycle close to 33%.

### Task 4: Positive Edge 33% Duty Cycle

The first step to performing an odd division is to get the positive version of the output. This output represents the time from the start of the system clock high to the start of the last high state period that a waveform is in its high state.

## Task 5: Negative Edge 33% Duty Cycle

Contrary to the positive edge, the negative edge is also necessary in order to get the time from the end of the first system clock high to the end of the last system clock high.

## Task 6: A True Division of 3

To get a true 50% duty cycle division of 3, the positively and negatively-induced outputs should be *united* to produce the total time of the original clock frequency being divided by 3. A diagram of how the positive and negative outputs relate to the duty cycle is shown below:

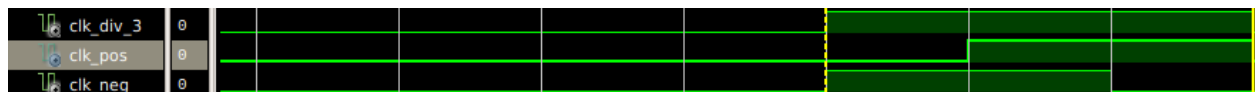


Fig: The union of the positive and negative outputs results in the true 33% duty cycle that was required of the program.

The union of the positive and negative 33% duty cycles cover for where each other misses out (in the clock cycle) and ends up producing a perfect Clock Division of 3 with 50% duty cycle.

## Task 7: Dividing by 5

The same method used in tasks 4, 5, and 6 were used for task 7 with the only difference being the register that was being used for bit shifting. In this task, a 5-bit register (instead of a 3-bit one) was used.

## Pulse & Strobe

### Task 8: 500 kHz Clock Divided by 200

This portion is broken up into two steps:

1. Creating a Clock divided by 100
2. Inverting the bits of the output wire every time the clock in the first step completed a full cycle

For the first step, I used a simple counter with 8 bits and reset the counter before it hit 100. This divides the clock by 100. This part was performed in the first always sequential block. The second step utilized the same inversion technique as in tasks 2 and 3. This part was performed in a second always sequential block. In order to verify that the waveform produced by this module follows the description: "**50% duty cycle divide by 200 clock running at 500Khz**", I took values for the period and high states.

#### *Final and Start Times of Waveforms*

	Start	Final
High State	105ns	1105ns
Period	105ns	2105ns

The results are  $\frac{1}{2105 * 10^{-9} - 105 * 10^{-9}} = 500KHz$ . Low state time can be calculated with  $period - high\ state = 1000$  and the duty cycle can be calculated with  $\frac{high\ state}{period} = 50\%$ , confirming the waveform adheres to the provided description.

### **Task 9: Strobing**

For this task, we were required to create a sequence of outputs that varied depending on how many increments had occurred in the past. Specifically, the pattern described was:

*Starting with 0, for every fourth clock cycle, subtract 5 to the original number and for every other clock cycle, add 3 to the number.*

I completed this task with two sequential always blocks. One acted as a counter to keep track of which clock cycle the program was on, and one altered the register holding the output. The only case to be careful of in this portion was implementing the counter to optimize space and performance. I used a simple Moore Machine that incremented a 2-bit register. The natural overflow allowed for the register to go through 4 different phases which reflected the 4 clock cycles that we had to keep track of.

## Testbench Design

The main cases to consider for testing include:

1. Reset is set
2. Reset is not set, clock is on
3. Reset is not set, clock is off

Thanks to the line

```
always #5 clk = ~clk;
```

we don't have to manually switch between the high and low clock states. The only thing we need to do is complete a global reset at the start of the initial block to cover the case of reset being set, and turn the clock on after the global reset to cover for the cases where clock is oscillating while reset is not set.

There was no initialization of outputs within the test bench because I believed that initialization should happen inside of the module itself (making sure that related tasks are grouped together).

After the passing of 1000ns (to allow for the div\_200 module to run a full period at least), I stopped the program with a `$finish;` to follow good practice.

## Waveform Outputs

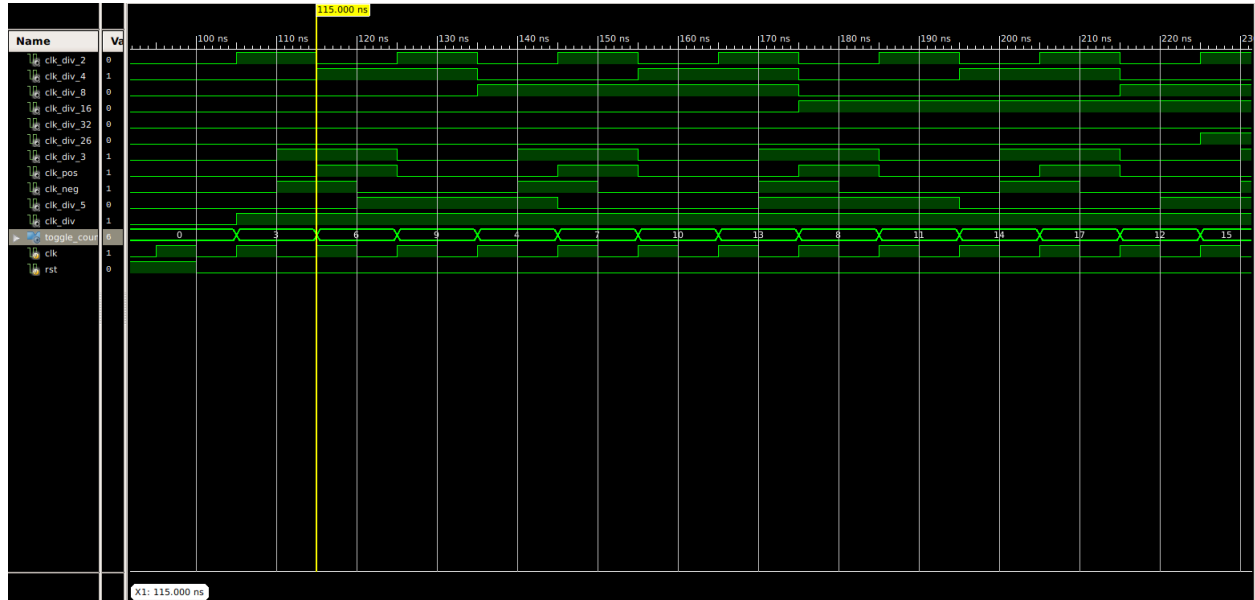


Fig: The waveforms of all clocks at the start of the program. The reset register was released after 100ns.

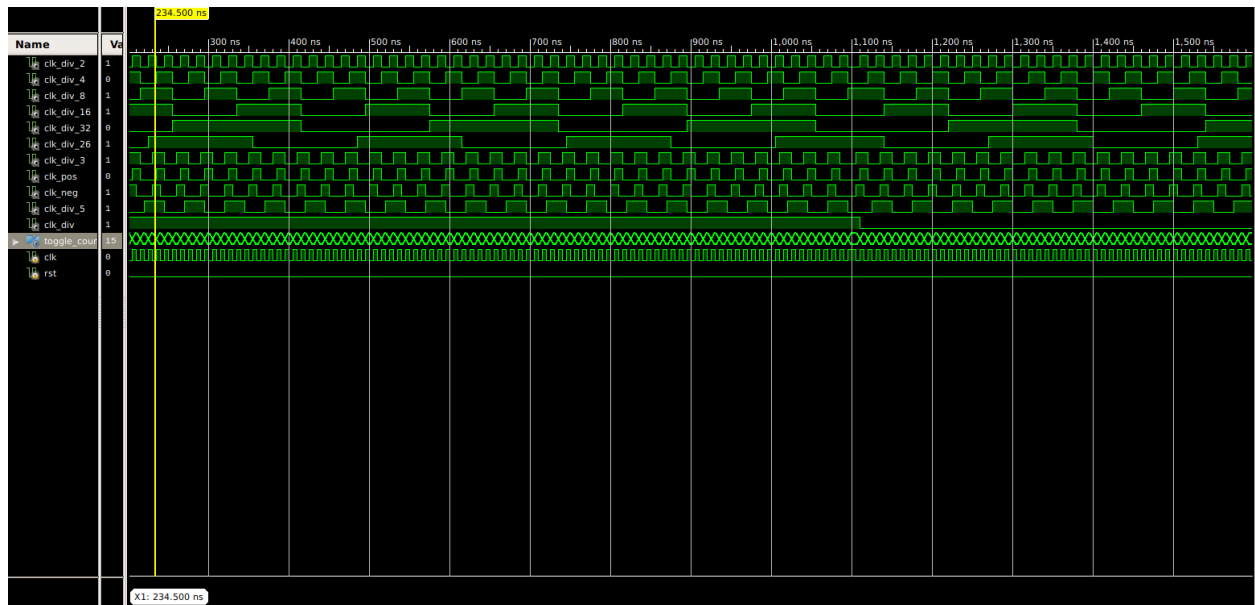




Fig: The latter half of the waveforms produced from the program. In this image, the tail end of the clock divided by 200 can be seen.

## Appendix

### Design Summary

clock_gen Project Status (05/02/2021 - 04:28:10)			
Project File:	CSM152ALab2_505308582.xise	Parser Errors:	No Errors
Module Name:	clock_gen	Implementation State:	Synthesized
Target Device:	xc6slx16-3csg324	• Errors:	No Errors
Product Version:	ISE 14.7	• Warnings:	No Warnings
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	<a href="#">Xilinx Default (unlocked)</a>	• Timing Constraints:	
Environment:	<a href="#">System Settings</a>	• Final Timing Score:	

Device Utilization Summary (estimated values)					
Logic Utilization	Used	Available	Utilization		
Number of Slice Registers	40	18224			0%
Number of Slice LUTs	43	9112			0%
Number of fully used LUT-FF pairs	26	57			45%
Number of bonded IOBs	21	232			9%
Number of BUFG/BUFGCTRLs	1	16			6%

Detailed Reports						
Report Name	Status	Generated	Errors	Warnings	Infos	
<a href="#">Synthesis Report</a>	Current	Sun May 2 04:28:09 2021	0	0	<a href="#">4 Infos (4 new)</a>	
Translation Report						
Map Report						
Place and Route Report						
Power Report						
Post-PAR Static Timing Report						
Bitgen Report						

## Synthesis Report

Release 14.7 - xst P.20131013 (lin64)

Copyright (c) 1995-2013 Xilinx, Inc. All rights reserved.

-->

Parameter TMPDIR set to xst/projnav.tmp

Total REAL time to Xst completion: 0.00 secs

Total CPU time to Xst completion: 0.03 secs

-->

Parameter xsthdmdir set to xst

Total REAL time to Xst completion: 0.00 secs

Total CPU time to Xst completion: 0.03 secs

-->

Reading design: clock\_gen.prj



## TABLE OF CONTENTS

- 1) Synthesis Options Summary
- 2) HDL Parsing
- 3) HDL Elaboration
- 4) HDL Synthesis
  - 4.1) HDL Synthesis Report
- 5) Advanced HDL Synthesis
  - 5.1) Advanced HDL Synthesis Report
- 6) Low Level Synthesis
- 7) Partition Report
- 8) Design Summary
  - 8.1) Primitive and Black Box Usage
  - 8.2) Device utilization summary
  - 8.3) Partition Resource Summary
  - 8.4) Timing Report
    - 8.4.1) Clock Information
    - 8.4.2) Asynchronous Control Signals Information
    - 8.4.3) Timing Summary
    - 8.4.4) Timing Details
    - 8.4.5) Cross Clock Domains Report

```
=====
*           Synthesis Options Summary           *
=====

---- Source Parameters
Input File Name           : "clock_gen.prj"
Ignore Synthesis Constraint File : NO

---- Target Parameters
Output File Name          : "clock_gen"
Output Format              : NGC
Target Device             : xc6slx16-3-csg324

---- Source Options
Top Module Name           : clock_gen
Automatic FSM Extraction   : YES
```

FSM Encoding Algorithm : Auto  
Safe Implementation : No  
FSM Style : LUT  
RAM Extraction : Yes  
RAM Style : Auto  
ROM Extraction : Yes  
Shift Register Extraction : YES  
ROM Style : Auto  
Resource Sharing : YES  
Asynchronous To Synchronous : NO  
Shift Register Minimum Size : 2  
Use DSP Block : Auto  
Automatic Register Balancing : No

---- Target Options

LUT Combining : Auto  
Reduce Control Sets : Auto  
Add IO Buffers : YES  
Global Maximum Fanout : 100000  
Add Generic Clock Buffer(BUFG) : 16  
Register Duplication : YES  
Optimize Instantiated Primitives : NO  
Use Clock Enable : Auto  
Use Synchronous Set : Auto  
Use Synchronous Reset : Auto  
Pack IO Registers into IOBs : Auto  
Equivalent register Removal : YES

---- General Options

Optimization Goal : Speed  
Optimization Effort : 1  
Power Reduction : NO  
Keep Hierarchy : No  
Netlist Hierarchy : As\_Optimized  
RTL Output : Yes  
Global Optimization : AllClockNets  
Read Cores : YES

Write Timing Constraints : NO  
Cross Clock Analysis : NO  
Hierarchy Separator : /  
Bus Delimiter : <>  
Case Specifier : Maintain  
Slice Utilization Ratio : 100  
BRAM Utilization Ratio : 100  
DSP48 Utilization Ratio : 100  
Auto BRAM Packing : NO  
Slice Utilization Ratio Delta : 5

=====

=====

\* HDL Parsing \*

=====

Analyzing Verilog file "/home/ashley/Documents/M152A/CSM152ALab2\_505308582/clock\_gen.v" into  
library work

Parsing module <power\_two\_division>.

Parsing module <even\_division>.

Parsing module <odd\_division>.

Parsing module <glitchy\_counter>.

Parsing module <clock\_gen>.

=====

\* HDL Elaboration \*

=====

Elaborating module <clock\_gen>.

=====

\* HDL Synthesis \*

=====

Synthesizing Unit <clock\_gen>.

Related source file is "/home/ashley/Documents/M152A/CSM152ALab2\_505308582/clock\_gen.v".

Found 8-bit register for signal <toggle\_counter>.

Found 8-bit adder for signal <toggle\_counter[7]\_GND\_1\_o\_add\_2\_OUT> created at line 76.

Summary:

inferred 1 Adder/Subtractor(s).

inferred 8 D-type flip-flop(s).

Unit <clock\_gen> synthesized.

=====

## HDL Synthesis Report

### Macro Statistics

# Adders/Subtractors : 1

8-bit adder : 1

# Registers : 1

8-bit register : 1

=====

\* Advanced HDL Synthesis \*

=====

Synthesizing (advanced) Unit <clock\_gen>.

The following registers are absorbed into counter <toggle\_counter>: 1 register on signal <toggle\_counter>.

Unit <clock\_gen> synthesized (advanced).

=====

## Advanced HDL Synthesis Report

### Macro Statistics

# Counters : 1

8-bit up counter : 1

\* Low Level Synthesis \*

=====

Optimizing unit <clock\_gen> ...

Mapping all equations...

Building and optimizing final netlist ...

Found area constraint ratio of 100 (+ 5) on block clock\_gen, actual ratio is 0.

Final Macro Processing ...

=====

Final Register Report

Macro Statistics

# Registers : 8

Flip-Flops : 8

=====

=====

\* Partition Report \*

=====

Partition Implementation Status

-----

No Partitions were found in this design.

-----

=====

\* Design Summary \*

=====

Top Level Output File Name : clock\_gen.ngc

#### Primitive and Black Box Usage:

```
-----  
# BELS           : 26  
#   GND          : 1  
#   INV          : 2  
#   LUT1         : 7  
#   MUXCY        : 7  
#   VCC          : 1  
#   XORCY        : 8  
# FlipFlops/Latches : 8  
#   FDR          : 8  
# Clock Buffers   : 1  
#   BUFGP        : 1  
# IO Buffers      : 9  
#   IBUF         : 1  
#   OBUF         : 8
```

#### Device utilization summary:

-----  
Selected Device : 6slx16csg324-3

#### Slice Logic Utilization:

Number of Slice Registers: 8 out of 18224 0%  
Number of Slice LUTs: 9 out of 9112 0%  
Number used as Logic: 9 out of 9112 0%

#### Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 9  
Number with an unused Flip Flop: 1 out of 9 11%  
Number with an unused LUT: 0 out of 9 0%  
Number of fully used LUT-FF pairs: 8 out of 9 88%  
Number of unique control sets: 1

#### IO Utilization:

Number of IOs: 21

Number of bonded IOBs: 10 out of 232 4%

Specific Feature Utilization:

Number of BUFG/BUFGCTRLs: 1 out of 16 6%

-----

Partition Resource Summary:

-----

No Partitions were found in this design.

-----

=====

Timing Report

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.

FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT  
GENERATED AFTER PLACE-and-ROUTE.

Clock Information:

-----

-----+-----+-----+

Clock Signal	Clock buffer(FF name)	Load	
--------------	-----------------------	------	--

-----+-----+-----+

clk_in	BUFGP	8	
--------	-------	---	--

-----+-----+-----+

Asynchronous Control Signals Information:

-----

No asynchronous control signals found in this design

Timing Summary:

-----

Speed Grade: -3

Minimum period: 1.837ns (Maximum Frequency: 544.292MHz)

Minimum input arrival time before clock: 3.238ns

Maximum output required time after clock: 3.634ns

Maximum combinational path delay: No path found

#### Timing Details:

-----

All values displayed in nanoseconds (ns)

=====

Timing constraint: Default period analysis for Clock 'clk\_in'

Clock period: 1.837ns (frequency: 544.292MHz)

Total number of paths / destination ports: 36 / 8

-----

Delay: 1.837ns (Levels of Logic = 9)

Source: toggle\_counter\_0 (FF)

Destination: toggle\_counter\_7 (FF)

Source Clock: clk\_in rising

Destination Clock: clk\_in rising

Data Path: toggle\_counter\_0 to toggle\_counter\_7

Gate Net

Cell:in->out fanout Delay Delay Logical Name (Net Name)

-----

FDR:C->Q	2	0.447	0.616	toggle_counter_0 (toggle_counter_0)
INV:I->O	1	0.206	0.000	Mcount_toggle_counter_lut<0>_INV_0 (Mcount_toggle_counter_lut<0>)
MUXCY:S->O	1	0.172	0.000	Mcount_toggle_counter_cy<0> (Mcount_toggle_counter_cy<0>)
MUXCY:CI->O	1	0.019	0.000	Mcount_toggle_counter_cy<1> (Mcount_toggle_counter_cy<1>)
MUXCY:CI->O	1	0.019	0.000	Mcount_toggle_counter_cy<2> (Mcount_toggle_counter_cy<2>)
MUXCY:CI->O	1	0.019	0.000	Mcount_toggle_counter_cy<3> (Mcount_toggle_counter_cy<3>)
MUXCY:CI->O	1	0.019	0.000	Mcount_toggle_counter_cy<4> (Mcount_toggle_counter_cy<4>)
MUXCY:CI->O	1	0.019	0.000	Mcount_toggle_counter_cy<5> (Mcount_toggle_counter_cy<5>)
MUXCY:CI->O	0	0.019	0.000	Mcount_toggle_counter_cy<6> (Mcount_toggle_counter_cy<6>)
XORCY:CI->O	1	0.180	0.000	Mcount_toggle_counter_xor<7> (Result<7>)
FDR:D		0.102		toggle_counter_7

-----

Total 1.837ns (1.221ns logic, 0.616ns route)



(66.5% logic, 33.5% route)

=====  
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk\_in'

Total number of paths / destination ports: 8 / 8  
-----

Offset: 3.238ns (Levels of Logic = 2)

Source: rst (PAD)

Destination: toggle\_counter\_0 (FF)

Destination Clock: clk\_in rising

Data Path: rst to toggle\_counter\_0

	Gate	Net			
Cell:in->out	fanout	Delay	Delay	Logical Name (Net Name)	
-----					
IBUF:I->O	1	1.222	0.579	rst_IBUF (rst_IBUF)	
INV:I->O	8	0.206	0.802	rst_inv1_INV_0 (rst_inv)	
FDR:R		0.430		toggle_counter_0	
-----					
Total		3.238ns	(1.858ns logic, 1.380ns route)		
			(57.4% logic, 42.6% route)		

=====  
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk\_in'

Total number of paths / destination ports: 8 / 8  
-----

Offset: 3.634ns (Levels of Logic = 1)

Source: toggle\_counter\_7 (FF)

Destination: toggle\_counter<7> (PAD)

Source Clock: clk\_in rising

Data Path: toggle\_counter\_7 to toggle\_counter<7>

	Gate	Net			
Cell:in->out	fanout	Delay	Delay	Logical Name (Net Name)	
-----					
FDR:C->Q	2	0.447	0.616	toggle_counter_7 (toggle_counter_7)	
OBUF:I->O		2.571		toggle_counter_7_OBUF (toggle_counter<7>)	

-----  
Total            3.634ns (3.018ns logic, 0.616ns route)  
                  (83.0% logic, 17.0% route)

=====  
  
Cross Clock Domains Report:  
-----

Clock to Setup on destination clock clk\_in

-----+-----+-----+-----+				
Src:Rise  Src:Fall  Src:Rise  Src:Fall				
Source Clock	Dest:Rise	Dest:Rise	Dest:Fall	Dest:Fall
-----+-----+-----+-----+				
clk_in	1.837			
-----+-----+-----+-----+				

=====  
  
Total REAL time to Xst completion: 5.00 secs  
Total CPU time to Xst completion: 3.60 secs

-->

Total memory usage is 376220 kilobytes

Number of errors : 0 ( 0 filtered)  
Number of warnings : 0 ( 0 filtered)  
Number of infos : 0 ( 0 filtered)