IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 5, July 2014.

www.ijiset.com

ISSN 2348 – 7968

Static Timing Analysis (STA) of SAS Expander on Virtex7 FPGA by using Vivado

G. B. Munde[1], P. P. Bartakke[2]

[1] Electronics & Telecommunication Department, College of Engineering, Pune, India

[2]Professor, Electronics & Telecommunication Department, College of Engineering, Pune, India

Abstract

This paper describes roadblock on STA of SAS Expander design by using Vivado tool, while building Emulation system for the same design on Virtex7 FPGA device. To start with timing analysis it's necessary to apply constraints. Paper describes different constraint formats supported by Vivado, general idea about XDC format, and application of constraint. Vivado generate timing reports for timing analysis, in addition to these we can generate custom reports by writing TCL scripts. Applying different implementation performance strategies and selecting best strategy for further iterations. We can use different synthesis options like (compile for area, timing or balanced) for trade-off between area and timing optimization of design while synthesizing design.

Keywords

STA, SAS (Serial Attached SCSI), XDC, TCL.

1 Introduction

To get desired performance of a design, we need to meet timing in a design. So, timing analysis is essential. We can do timing analysis in different methods like Dynamic Timing Analysis (DTA), Static timing Analysis (STA). DTA verifies functionality of the design by applying input vectors and checking for correct output vectors. Static timing analysis is a method of validating the timing performance of a design by checking all possible paths for timing violations under worst-case conditions [i]. STA approach typically takes a fraction of the time it takes to run logic simulation (i.e. DTA) on a large design and guarantees 100% coverage of all true timing paths in the design without having to generate test vectors. We can say that, we have achieved timing closure. When design has neither setup violation nor hold violation and also we applied all constraints.

We did STA of SAS Expander, while building emulation system targeting Virtex7 FPGA device for SAS Expander Design in-circuit emulation [ii]. FPGA Emulation of SAS Expander helps concurrent firmware and hardware development of SAS expander. This will ultimately result in reduced time to market. SAS Expander is switch used to connect SAS devices like initiators (i.e. controllers or Host Bus Adapters), Targets (i.e. Hard Disk Drives (HDDs) or solid state drives (SSDs)), and SAS Expanders [iv]. SAS Expanders enhances your external storage environment with high scalability and performance, by supporting SAS data transfer rates of 12, 6, 3 Gb/s, and SATA data transfer rates of 6 and 3Gb/s [v]. Virtex7 is 7-series FPGA family with high performance and high capacity, it's from Xilinx.

We can do timing check at different stages in FPGA Emulation flow for timing estimations, in our case we did timing check post synthesis and post route. Post route is more accurate timing estimations rather than post logic synthesis, since they include routing delays.

2 Timing Analysis

2.1 Timing constraints

To start timing analysis, we need to apply appropriate timing constraint. There are different formats of constraints like User-defined Constraint files (UCF), Synopsis Design Constraints (SDC), Xilinx design constraints (XDC), and Tool Command Language (TCL). In our case we used XDC formats applying constraint in Vivado [vi]. Applying appropriate constraint in standard order is essential to met timing closure.

We had defined major clocks earlier using create clock but few clock definitions were missing which were reported in timing summary report under heading “No Clock”. So, we have to add clock definition of those missing clocks with corresponding frequency in constraints.

From designers we come to know, we can define false Paths from SClk to SERDES clocks and vice versa. We have written false path constraints on these paths in file false_path.xdc. From this onwards, it was iterative to check timing reports, apply constraint, check timing reports with added constraints, and again apply constraints for next critical paths.

There were large number of timing violations in part of design logic and that was redundant logic for 3Gb/s data rate (necessary for 12Gb/s). We removed this related logic to ground, and re-synthesized and implemented design which gave improved timing results.

Next critical paths are from SCLK to PCLKS (SCLK is 4 times faster than PCLKS). We used custom script with additional filters with get_timing_paths, to generate group of timing paths with violation on the basis of Slack, start-point, endpoint. Most of these paths were from or to configuration registers on which we can apply multi-cycle paths.

2.2 Timing Reports

After applying constraint, we can check timing from timing reports. We can generate different timing reports for timing checks. By using these we come to know timing violation paths, missing clocks, and details of complete timing on specific paths. There are two ways to generate timing reports in Vivado with following TCL commands:

1. Report_timing_summary: It will report summary of timing related to every clock in the design. We can write this into a file and analyses it or else we can check this report directly in GUI. For more information about this command you can report_timing_summary -h
2. Report_timing: Reports timing summary for a specific timing path that may be from clock to clock or Start-point to endpoint.

2.3 Custom Timing Report using TCL

In addition to tool generated timing reports, we can generate custom reports by writing TCL scripts to get required details about timing. We have written a script to report timing violation paths in custom format. In which you can mention the start point, end point, launch clock, capture clock, and slack of a path in tabular form in a file for all timing paths in design. By referring Vivado user guides and TCL script manual, we had written script to get timing paths in custom format (by referring Vivado user guide of TCL commands). This script will write timing paths in custom format as follow:

Start-point Endpoint launch_clock capture_clock slack

Sample script is as follow named SClk_to_CClk_paths.tcl

```
#file to store all the paths and override any
content of it
setfp [open "SClktoCClk_path_hold_0.05.txt" w
]
puts $fp "timing path report for setup and hold
violations"
close $fp

#procedure to write a custom report
proccustom_report { listOfPaths } {
setp ""
setfp [open "SClktoCClk_path_hold_0.05.txt" a ]
set q [format {%-40s %-40s %-20s %-20s %7s}
"Startpoint" "Endpoint" "Launch Clock"
"Capture Clock" "Slack"]
puts $fp $q
puts $fp "\n"
foreach path $listOfPaths {
setstartpoint [get_property STARTPOINT_PIN
$path]
setstartclock [get_property
STARTPOINT_CLOCK $path]
set endpoint [get_property ENDPOINT_PIN
$path]
setendclock [get_property ENDPOINT_CLOCK
$path]
set slack [get_property SLACK $path]
set p [format {%-40s %-40s %-20s %-20s %7s}
$startpoint $endpoint $startclock $endclock
$slack]
puts $fp $p
}
puts $fp "\n"
puts $fp "\n"
close $fp
}

#getting timing paths in a variable
set paths [get_timing_path -from SClk -to CClk -
slack_lesser_than -0.5 -max_path 10000 -hold ]
custom_report $paths
```

Above scripts finds timing paths from SCLK to CCLK and write details in custom format for each of them. As shown in script, we have written a TCL procedure with name custom_report with argument List of paths. In list variable paths, we have added all the timing paths with hold violation lesser than -0.5. If number of path exceeds number 10000, then it will add paths with more negative paths to the list variable.

This script is very useful when timing violations are on paths from same vector register to another group of vector register. Custom reports make easy to analyses these timing paths. In this way, we can write small scripts to get timing paths to/from a clock or to an endpoint.

3 Strategies for timing Enhancement

3.1 Implementation strategies

Implementation in Vivado tool has different implementation strategies [iii] which use different algorithms for placement and routing. Those strategies can improve either of area, performance, congestion, and power. For achieving timing closure, we tried all strategies of implementation to improve performance.

Post route timing results of these different strategies are given in Table 1. Here, timing parameters are WNS (Worst Negative Slack), TNS (Total Negative Slack), WHS (Worst Hold Slack), and THS (Total Hold Slack).

Comparing timing results in table 1, we conclude that in case of our design performance exploreSLLs gives better result. So, we have preferred performance exploreSLLs strategy for further implementation runs.

Table 1 Timing Summary of Different implementation strategies

Timing parameters / Strategy Implementation	WNS (ns)	TNS (ns)	WHS (ns)	THS (ns)
Performance_Explore	-9.6	-14489	-6.8	-20384
Performance_RefinePlacement	-14.9	-15930	-14.3	-11766
Performance_WLBlockPlacement	-7.3	-3541	-7.5	-13102
Performance_WLBlockPlacementFanoutOpt	-10.5	-13469	-7.3	-8039
Performance_LateBlockPlacement	-9.7	-4512	-8.0	-20971
Performance_NetDelay_high	-7.2	-3396	-7.4	-12738
Performance_NetDelay_medium	-9.2	-18340	-4.7	-4558
Performance_NetDelay_low	-9.5	-23901	-6.8	-8611
Performance_ExploreSLLs	-6.7	-3390	-8.8	-11428

3.2 Synthesis strategies

While synthesizing design, we can optimize it for area and/or timing (performance). We have synthesized SAS Expander design using Precision tool from Mentor Graphics. In precision, we can trade off optimization for timing and area with synthesis options compile_for_area, compile_for_timing, and default it is balanced.

Initially, we were synthesizing design for area optimization. Utilization of SAS Expander design has 75% post synthesis (67% post route). We had enough safety margins on overall implementation on utilization (area). We have synthesized design for timing optimization, area optimization, and balanced mode.

From results, we can conclude that using timing optimization mode while synthesis improved timing status to a great extent i.e. less violation comparatively for area optimization. It caused to increase utilization around 10% which is tolerable in case of our design.

We were synthesizing design with 37MHz clock for SClk (i.e 3Gb/s data rate). We did an experiment in which we synthesized SAS Expander design at 75MHZ for SClk (6Gb/s data rate) i.e. at higher frequency requirement. Synthesized netlist of design implemented at 37MHz, which resulted in less timing violation in Table 3 as well as it reduced LUT utilization shown in chart 2.

Different synthesis strategies include area optimization (area 37MHz, area 75MHz), timing optimization (timing 37MHz, timing 75MHz), and balanced. Chart 1, chart 2 summarizes resource utilization for different synthesis strategy runs post synthesis, post route respectively. Similarly, Table 2, Table 3 summarizes timing result of different synthesis strategy runs post synthesis, post route respectively.

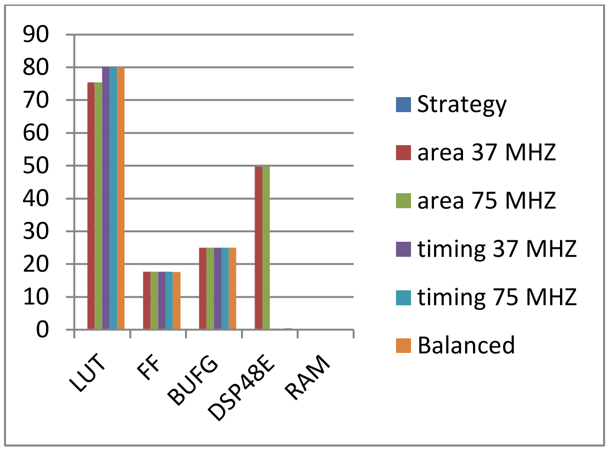

chart 1 Post Synthesis Utilization Summary

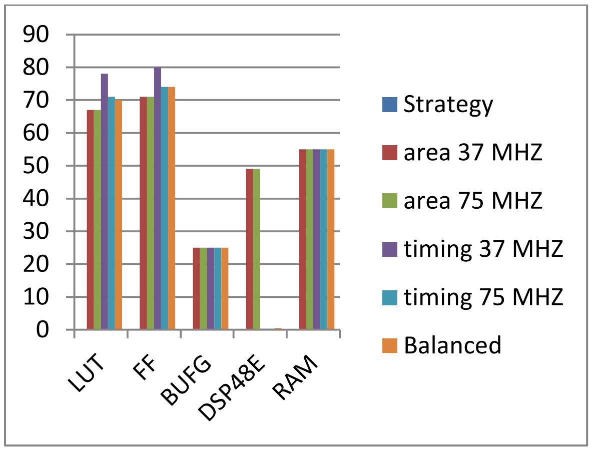

chart 2 Post Route Utilization Summary

Table 2 Post Synthesis Timing summary

Strategy	WNS(ns)	TNS(ns)	WHS(ns)	THS(ns)
area 37 MHZ	-6.052	-1410	-1.588	-19.25
area 75 MHZ	-6.052	-1410	-1.588	-19.25
timing 37 MHZ	-6.485	-1274	-1.588	-13.82
timing 75 MHZ	-6.485	-1274	-1.588	-13.64
balanced	-8.85	-5168	-1.588	-19.35

Table 3 Post Route Timing summary

Strategy	WNS	TNS	WHS	THS
area 37 MHZ	-7.9	-3169	-4.6	-15015
area 75 MHZ	-7.9	-3169	-4.6	-15015
timing 37 MHZ	-7.3	-2352	-7.6	-11640
timing 75 MHZ	-6.6	-1548	-7.9	-9009
balanced	-8.3	-6127	-6.9	-14106

4 Conclusion

Static Timing Analysis needs application of proper constraints, generating timing reports, and checking if timing met or not. Using TCL scripts helps to speed up timing analysis. Using different strategies while implementation and synthesis can improves timing. Synthesizing design for timing optimization results in improved performance at the cost of area.

5 Acknowledgments

Authors would like to acknowledge the help and support of several people who have supported this effort. Special thanks LSI India Research and Development Pvt. Ltd. for giving opportunity to work with them and manager Mohan Jindal and Hrushikesh Vaidya at LSI for their valuable suggestions.

6 References

[I]http://asic-soc.blogspot.in/2008/08/dynamic-vs-static-timing-analysis.html

[ii] Varghese, J.; Butts, M.; Batcheller, J.; "An efficient logic emulation system,"

[iii] Implementation, Using Constraints, and TCL scripting user guides for vivado.

[iv] LSI® 12Gb/s SAS/SATA Expander product brief

[v] www.xilinx.com