**AXI SHA256 Optimization Report and Change Log**

**Revision 1.0**

Table of Contents

[Introduction 3](#_Toc41656474)

[Initial Procedure, Research, and Desired Outcomes 3](#_Toc41656475)

[Unoptimized Timing Report Summery 3](#_Toc41656476)

[Helpful Links/Documents 5](#_Toc41656477)

# Introduction

This is a working document for attempted optimization procedures and results and will help decide the direction to move in. This document will be informal and mostly serve the purpose of me documenting what problems I have, what I tried, and what worked.

# Initial Procedure, Research, and Desired Outcomes

In first deciding which optimizations should be done, the culprits of the bogged down performance need to be identified. During 1.0 development, I noticed there were certain nets with a high number of inputs (high fan-out) which could account for the >50% routing delay. So, I need to identify which nets and whether I can apply certain techniques or rid of them entirely. Secondly, there is a long data path for the compressor and scheduler modules that are inherent to the algorithm. I did not realize that Vivado could not have leveled the adders but instead serialized them – using parenthesis on certain additions can parallelize these adders. There is also the issue of pipelining those long data paths. In the compressor (and this is similar to the scheduler) the next state of A, At+1, is a function of At – Ct, Et – Ht, Wt, and kt. Because of this, pipelining those additions will require the state registers to wait until the calculation is done, halving throughput and doubling the frequency so no net gain. The only viable datapath optimization can be done on the compressor (unless there is something I can do with the padder or update state machines) to match the data path with the scheduler (3 adders). One other optimization that comes to mind is an area optimization of the hasher module. Instead of using 8 adders, the 8 additions can be serialized using decoders and muxes to control which two pieces of data are being added. I also noticed that the AXI4-Lite interface is using 27% of the total number of LUTs utilized – something else to investigate. I also remember reading that XOR operations require more LUTs and the compressor uses significantly more LUTs than the scheduler, which is similar, and the SHA256 specification says that OR can be used instead. The summary of the desired optimizations is:

* Address high fan-out by
  + Reducing control signals
  + Duplicate/register signals
* Address high routing delay (>60%)
  + Research how
* Reduce the compressor’s critical path by performing a pre-calculation
* Serialize the hasher’s additions
* Change XOR operations used in the sha256 spec to OR

# Unoptimized Timing Report Summery

A timing report was generated for a clock at 150 MHz. The Microblaze CPU (utilizing instruction and data cache) and the accelerator failed to meet timing. The report and excel files (unoptimized-timing-summary-wCache-150MHz) are in the “docs” folder of the repository. Some important observations were made:

1. The Microblaze’s cache hardware severely did not meet timing
2. The compressor (as expected) did not meet timing BUT the scheduler did indicate that the datapath optimization will be well worth it.
3. There are some paths observed, specifically some paths in the AXI interface of the accelerator that had logic delays well within the constraints but has huge routing delays (>70%) (example shown in Figure 1).

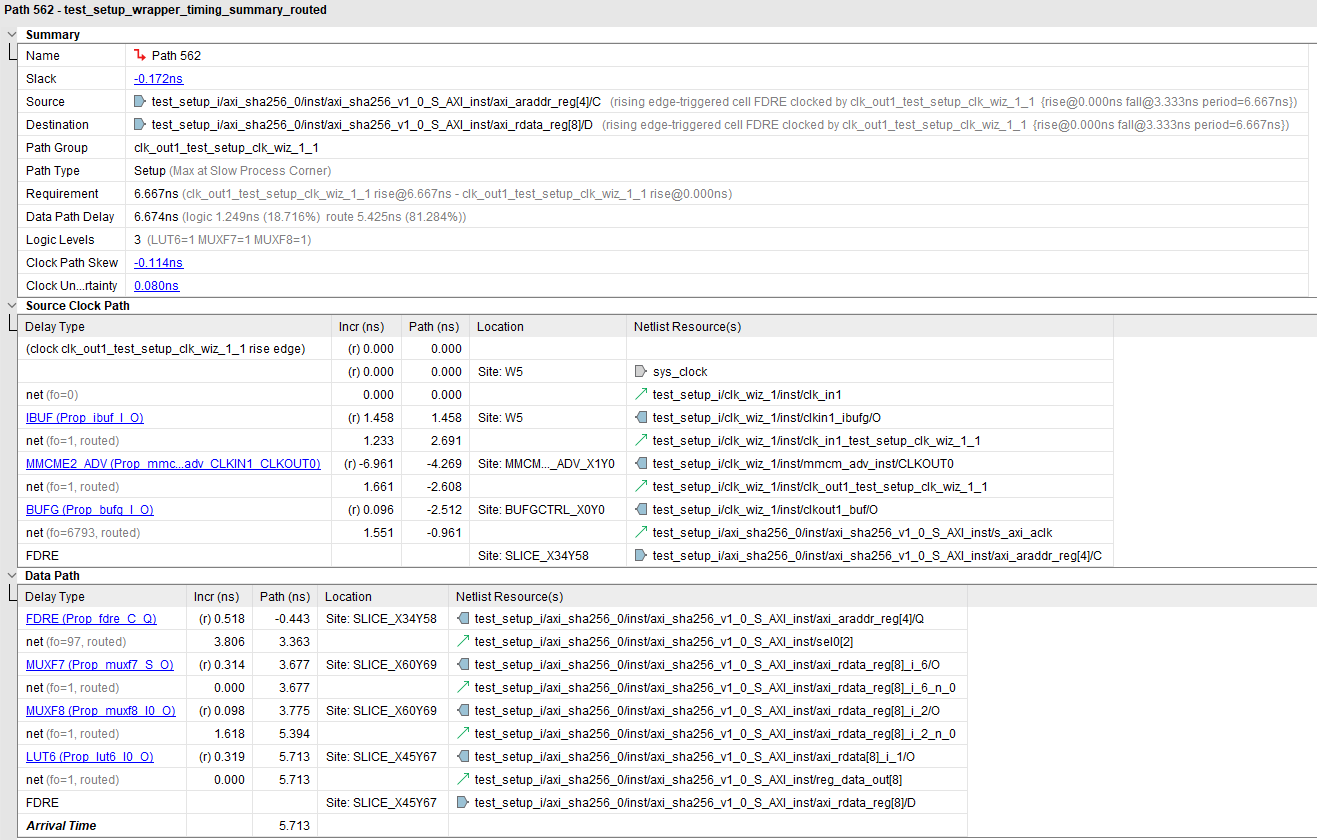


Figure 1: High routing delay with low logic delay example

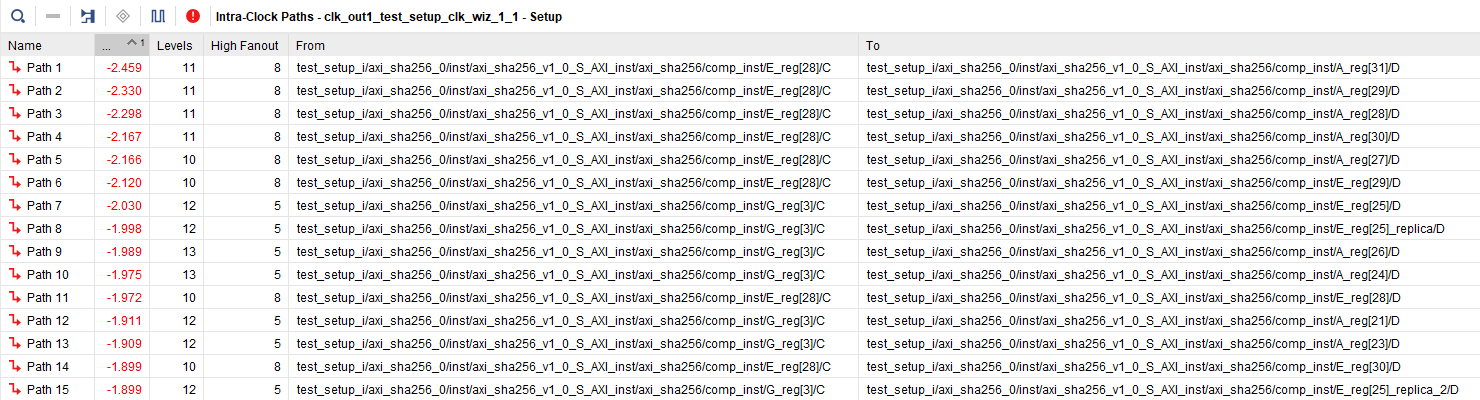


Figure 2: Failed timing paths with longest delay

Once these issues are resolved and at least the data path optimization has been made, a frequency improvement of 50% can be achieved (previously operating at 100 MHz).

The fan-out counts also appear to be within acceptable levels (<1000) as indicated in “unoptimized-high-fanout-report.rpt”.

# AXI Master DMA

## Questions for Operation

1. **What signals need to be generated for a 64-byte burst read?**
2. **What does the INCR signals do?**
3. **How do I request then wait for the data?**

# Migrating to the Zynq MPSoC

## Questions to be Resolved for Migration

1. **Which AXI interface should I use? High Performance (HP), Coherent High Performance (HPC), Two-Way AXI Coherency Extension (ACE), or Cache-Coherent Accelerator Slave Port (ACP)?**
2. **What are their trade-offs?**

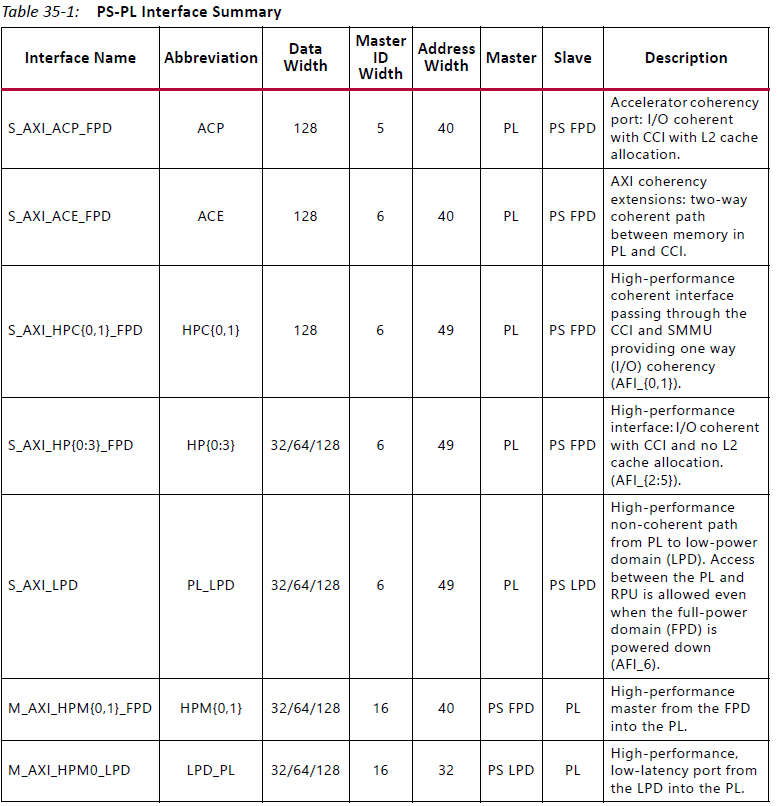
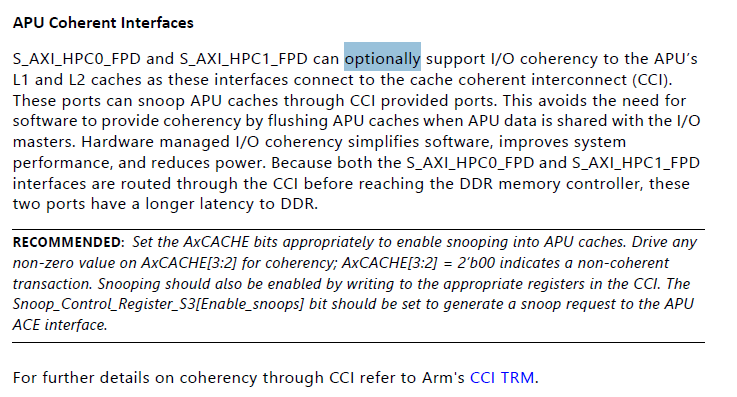


Figure 3: UG1085 pg. 1056

1. **Depending on which one I choose, what steps are needed to maintain coherency, if needed?**

****

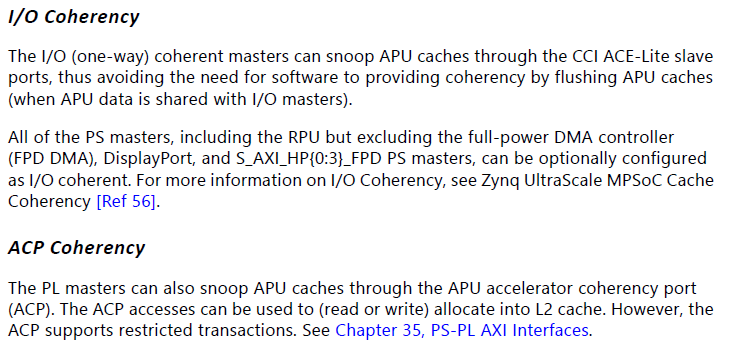


Figure 4: UG1085 pg. 366

1. **What other steps do I need to take to successfully create a valid AXI memory request transaction?**
2. **Do I need to worry about the exception level and whether data could be protected?**
3. **Is the L2 cache coherent with the L1 cache?**

# Change Log

# Helpful Links/Documents

1. [Vivado design analysis and timing closure hub](https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0006-vivado-design-analysis-and-timing-closure-hub.html)
2. [Suggestions for high fan-out nets](https://www.xilinx.com/support/answers/9410.html)
3. [Timing closure techniques](https://www.xilinx.com/publications/prod_mktg/club_vivado/presentation-2015/paris/Xilinx-TimingClosure.pdf)