|  |
| --- |
| University of Southern California |
| EE577B – Project Phase 5 |
| Technical Report |

|  |
| --- |
| Andrew Adam, Harsh Bhardwaj  4-24-2016 |

Contents

[Introduction 2](#_Toc449285501)

[Approach 2](#_Toc449285502)

[Division of Tasks 2](#_Toc449285503)

[Design Partitioning 3](#_Toc449285504)

[Router Design 3](#_Toc449285505)

[Processor Design 4](#_Toc449285506)

[CMP Design 4](#_Toc449285507)

[RTL for synthesis 5](#_Toc449285508)

[Trade-Offs/Choices 5](#_Toc449285509)

[Implementation 8](#_Toc449285510)

[Performance 8](#_Toc449285511)

[Conclusion 9](#_Toc449285512)

[Bibliography 9](#_Toc449285513)

[Appendix 10](#_Toc449285514)

[Area-Delay Product 10](#_Toc449285515)

[STA 11](#_Toc449285516)

[Place-and-route Summary 12](#_Toc449285517)

# Introduction

Traditionally, integrated circuits were fabricated using dedicated wiring between each of the on chip modules. As transistor sizing continues to decrease, the dominant delay factor has become wiring and wiring congestion has become more of a concern. One of the ways to mitigate these issues is to reuse wiring using networking theory. In this report, we describe the architecture and implementation of a four-core chip multiprocessor, which utilizes a network-on-chip in a ring topology for communication between cores. We begin with our general approach and architecture definition for the design. Following this, we will describe the tradeoffs and other choices we made in optimizing our design. Next, we will go over how we implemented our chosen architecture. After this, we will describe the overall performance of our design. Finally, we will give our concluding remarks about the whole design.

# Approach

## Division of Tasks

The tasks for each phase of the project were divided as follows

* Phase 1 (Gold Router)
  + Andrew
    - Router RTL
    - Ring RTL
    - Synthesis
  + Harsh
    - Router test-bench
    - Ring test-bench
  + Both
    - Verification of results
* Phase 2 (Gold Processor)
  + Andrew
    - Register File RTL
    - Program Counter RTL
    - Logical & Shift instructions
    - Memory instructions
    - Synthesis
  + Harsh
    - Arithmetic instructions
    - Branch instructions
    - Perl compiler script
  + Both
    - Pipeline design and RTL
    - ALU design and RTL
    - Verification of results
* Phase 3 (Gold NIC)
  + Andrew
    - NIC RTL
    - Synthesis
  + Harsh
    - CPU modifications for new load/store instructions
    - NIC test-bench
    - CMP test-bench
  + Both
    - Verification of results
* Phase 4 (Synthesis, Layout, and Verification)
  + Andrew
    - Synthesis
    - LEC
    - Post-synthesis and post-route simulation
  + Harsh
    - Place and Route
    - PrimeTime STA

## Design Partitioning

### Router Design

The router consists of inputBuffer and outputBuffer RTL modules. There are three inputBuffer and three outputBuffer instances in the main gold\_router module. The inputBuffer modules have routing logic decrement the hop-count of an incoming packet, and determine if the packet has to be sent to the processing element. The outputBuffer module contains arbitration logic that arbitrates between two incoming requests. Both these modules have two 64-bit registers to hold packets (one each for even and odd polarity packets). These modules also contain grant logic to indicate if the buffer is full/empty.

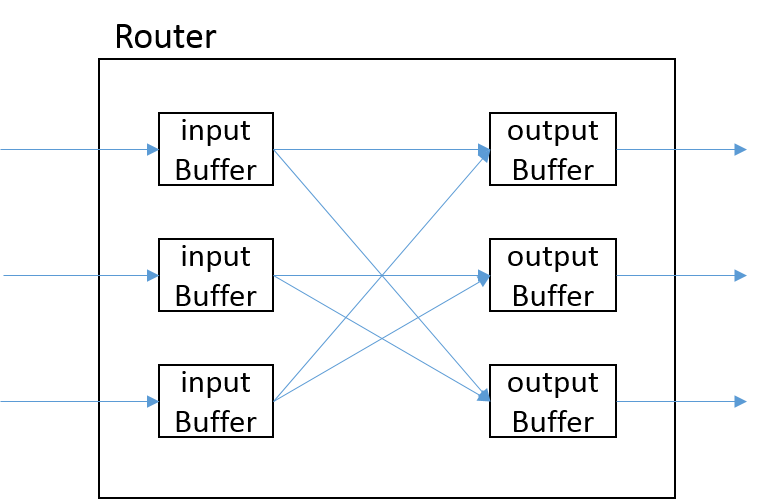


Fig-1 – The gold\_router module

### Processor Design

The processor we implemented was a four-stage pipelined CPU. It consisted of four explicit RTL modules – the program counter, register file and the ALU. The pipeline registers, and the remainder of the logic (decode, multiplexers, stalling and branching) were implemented in the main CPU module (gold\_cpu) – in a RTL rather than structural fashion. Fig-2 shows the high-level block diagram of our 4-stage CPU design. The modules that were defined as separate RTL modules have a dotted boundary.

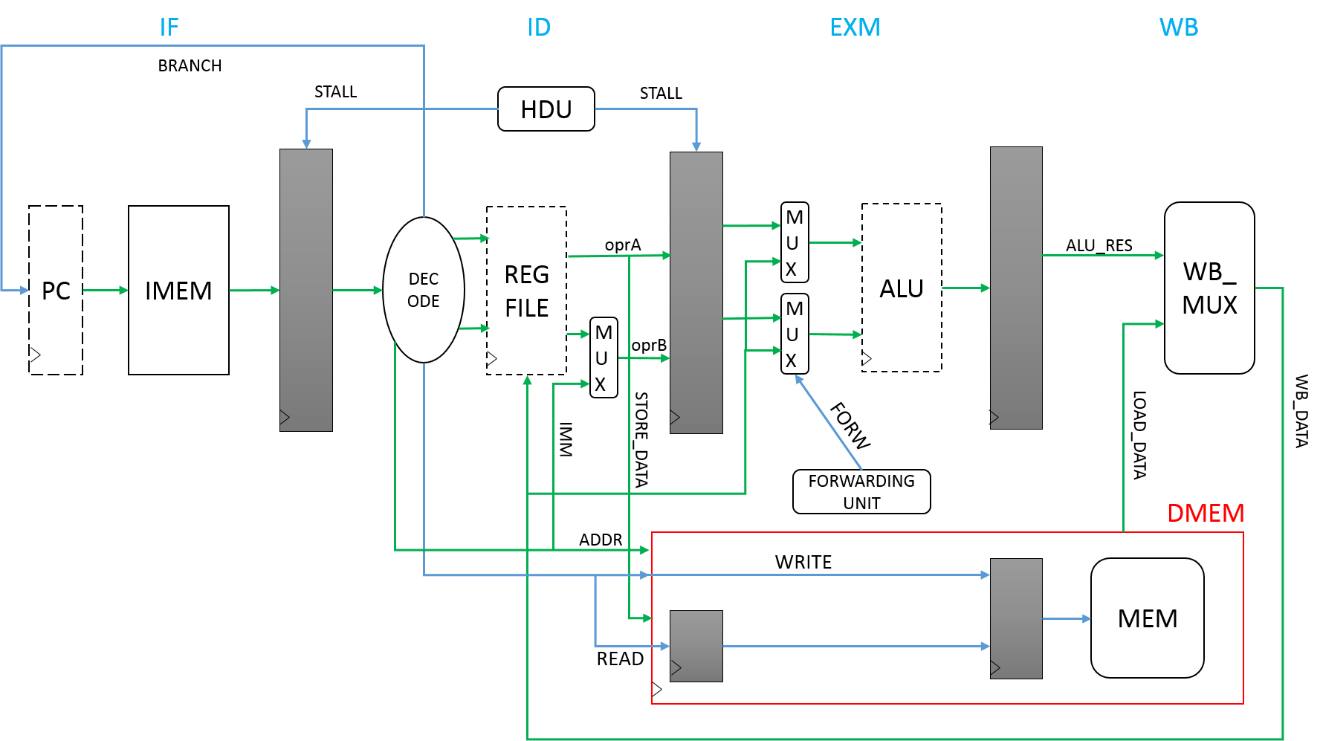


Fig-2 – Four-stage CPU

### CMP Design

The CMP design consists of the gold\_ring module (which is internally composed of four gold\_router modules connected in a ring fashion) with connections to four NICs (gold\_nic) each of which interfaces to a single processor (gold\_cpu).

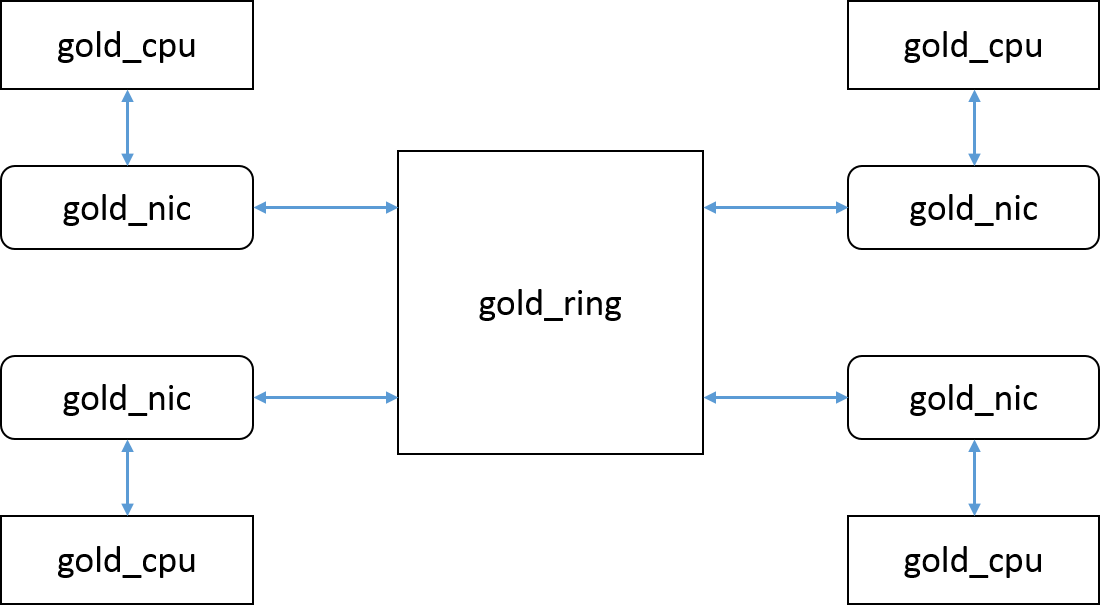


Fig-3 – The gold\_cmp module

## RTL for synthesis

All of our RTL code was synthesizable so we did not need to make any modifications to it for Phase 4.

# Trade-Offs/Choices

In each phase of the project, we needed to decide the best modularization and hierarchy for our Verilog code. In the first part of the project where we developed the gold router, we noticed that within each router, the output buffers and output control logic are the same. Additionally, all of the input buffers and input control logic are the same, except for the PE input control logic. Since it would greatly improve readability to reuse these similar sections of code than to duplicate it multiple times, we decided that each output buffer and control logic could be contained within one module and each input buffer/control logic could be contained within another module that contains a parameter to specify the PE input buffer.

Within the processor design phase of the project, there were several decisions we made for modularization. We decided it would be best to separate the program counter, alu, register file, and processor control logic into separate modules. The reason for this decision is that we believe it would be best to group sections of code that have similar inputs and outputs. Having the snippets of code that depend on one another nearby would improve readability and understandability of the code. We could have further modularized the alu or processor control logic, but had we done this, be believed it would have led to redundant code and make tracing the flow of signals more difficult.

In each phase of the project, we needed to determine the best method for designing our test benches. In the first part of our project, we believed that if we thoroughly test our router design, it would most likely work when instantiated four times in the gold ring configuration. This is because the network topology was shown to be deadlock-free and each router’s control logic should be able to correctly handle blocking situations. Therefore, we focused our efforts on testing specific cases on the router design.

For the router test bench, we first tested individual packets on a single channel. This test verifies that the router forwards each packet coming from the corresponding input buffer is forwarded to the right output buffer. Second, we tested the case where multiple packets are sent to the same input channel on different polarities. This test verifies that the packet is stored into the correct input/output buffers depending on the polarity. Third, we tested the case where multiple packets arrive on different channels and on different polarities. This verifies that multiple packets can pass through the router simultaneously when there is not any blocking. Next, we performed a gather test where two channels request the same output on different polarities. This test further verifies that the packets should be able to pass through, since there is not any blocking. In the next test case, we performed the gather test where multiple input channels request the same output on the same polarity. This test case verifies the arbitration functionality. In our final test case, we performed several blocking tests where we held the ready out signal low and filled up the internal buffers. This test showed that the ready in signal becomes low when the input buffer is full and that the internal buffers are not corrupted by a write when the router is not ready to receive. Through all of these tests, we believed that the router provided the correct functionality, since each test case was testing each specific section of Verilog code. Furthermore, the gather test for the ring with four routers instantiated also showed that each packet would arrive at the intended destination even when blocking is present.

For the testing of the processor, we first developed a Perl script that would convert assembly instructions to memory fill files to be read by Verilog. This greatly simplified the verification process, since we would not need to spend time manually converting the desired instruction sequence into an input readable by our Verilog test bench. Our basic approach for verification was to create an input test vector that verifies each specific code block. For example, since we included the participation field processing within the register file, if we only test each PPP value once, it must work for each other instruction that uses PPP, since the processing of the participation field is independent of the instruction. Using this same principle of finding what sections of code are independent, we were able to reduce the required tests. By testing each individual section of code using a separate test vector, we believed that out test bench covered all cases and proved that our design is functionally correct.

In the third part of the project where we connected four processors and routers in a ring network configuration, we used the following methodology for developing our test benches. First, we performed a gather test where each processor sends to the three other processors. This test verifies the connection and functionality of the network interface component. In addition to this test, we also modified the teaching assistant’s test bench from part two of this project such that it would work for part three of our design. This test bench provides one hundred different cases of random vectors and applies it to each of the four processors. By using a random input set, we were more certain that there are not any cases that we missed by only testing according to each specific code block and it would very likely work for any input assembly code.

In the final phase of the project where we did synthesis, placement, and routing, we did not need to make any changes to make our code synthesizable, since all of our previous parts were synthesizable. However, when optimizing our area-delay product, we added a Designware 2-stage multiplier since this was our critical path and this led to issues getting a positive Conformal result. When debugging Conformal we isolated the multiplier as causing the nonequivalence and thus decided to keep our original 1-stage multiplier in order to get a correct Conformal result.

When optimizing the area-delay product, we were able to achieve a good reduction in the product by trying several different options within design compiler. By using the compile -map\_effort high -area\_effort high command, we noticed that design compiler is able to automatically map operators within our alu such that unused blocks within one operation are shared with another operation, thus reducing area. Using a for loop within the design compiler tcl script, we obtained the following results for our first design of the 2-stage multiplier:

|  |  |  |
| --- | --- | --- |
| Area (µm2) | Delay (ns) | Area\*Delay (ns ·µm2) |
| 745603 | 2.92 | 2177161.29 |
| 732140 | 3 | 2196419.71 |
| 682187 | 3.36 | 2292148.81 |
| 670838 | 3.43 | 2300974.55 |
| 665307 | 3.6 | 2395104.81 |
| 664331 | 3.7 | 2458023.77 |
| 726157 | 3.6 | 2614164.47 |
| 706697 | 3.81 | 2692516.61 |
| 659434 | 4.1 | 2703679.7 |
| 720077 | 3.86 | 2779499.09 |
| 708678 | 4.06 | 2877233.44 |
| 703351 | 4.13 | 2904840.31 |
| 696710 | 4.2 | 2926180.45 |
| 714009 | 4.18 | 2984559.44 |
| 714009 | 4.18 | 2984559.44 |
| 702077 | 4.3 | 3018931.16 |
| 698020 | 4.4 | 3071289.7 |
| 697941 | 4.5 | 3140732.72 |
| 693237 | 4.7 | 3258213.01 |
| 687990 | 5.08 | 3494989.39 |
| 694642 | 5.23 | 3632977.11 |
| 687097 | 5.31 | 3648487.35 |
| 689379 | 5.31 | 3660603.37 |

Table 1 – Area and delay data

Fig-4 – Period vs Area\*Delay product for design with 2-stage multiplier

By performing 23 iterations, we see a general linear trend in the area\*delay product when increasing the clock period. Since we could not achieve a positive Conformal result using the 2-stage multiplier, we needed to redo our synthesis iterations using a new 1-stage multiplier. The results are shown in Figure 5 and Table 2 in the appendix.

Our optimal area delay product was achieved with a clock period of 2.7 ns and an area of 845137 µm2. We believe this is the best achievable result by modifying design compiler options because further reduction of the clock period resulted in a significant increase in area and therefore increased our area delay product.

# Implementation

We were able to complete all the steps in phase 4 (synthesis, STA and place-and-route) of the project. The functionality of the generated net-lists after synthesis and place-and-route was successfully tested by running back-annotated SDF simulations.

The area reported by Synopsys DC Compiler was 845,136.55 µm2, and the area reported by Cadence Encounter after place-and-route was 1,121,604.81 µm2 (total core area). This means that we had an increase of 37.12% in area due to routing.

There were a few issues we encountered during Phase 4 which are listed below

* LEC non-equivalences due to DesignWare instance
  + We had initially used a DesignWare 2-stage multiplier in our design – but this caused a lot of LEC non-equivalence errors. When we switched to a single-stage (using the generic ‘\*’ operator), we did not get any non-equivalence errors.
  + The TAs suggested that we treat this module as a black-box during LEC – this worked for us only when the entire ALU module (which instantiated the DW block) was treated as a black-box.
* LEC abort-points
  + There were a lot of abort points reported in our design, which would cause non-equivalence warnings. This issue was solved by running special analysis commands for the abort points.

# Performance

The critical path delay of our design as reported by Synopsys PrimeTime was **2.70 ns**.

The area-delay product for our final design was **2,281,868.694 µm2·ns**.

The post-place-and-route delay of our design was **8.416 ns** – which means that routing increased the delay by 211 %.

# Conclusion

If we had more time, we would have focused more on area optimizations – especially the ALU which contains a lot of redundant components. If power was taken into consideration, we would have tried to eliminate redundancies in our ALU – which would have resulted in power as well as area savings.

We learnt quite a few lessons during the course of this project, some of which are listed below

* Efficient division of task between team-members saves time and reduces confusion
* Starting early allowed us to give us more time to testing and fixing errors in our designs
* Using scripts instead of the slow and buggy GUIs helped us save time during phase-4 (even though writing and debugging those scripts took a little bit of time).
* Testing for trivial cases is important – we learnt this the hard way during phase 2 (we allowed forwarding when the destination register of the parent instruction is $0).

Topics we would have liked to be included in this course

* DRAM design – 577A covered only SRAMs
* More about what the tools do – the tools demos barely scratched the surface and we had very little knowledge about what sort of things we could do when we faced errors (especially during place-and-route).

# Bibliography

*Encounter Conformal Equivalence Checking User Guide* – Cadence Design Systems, Inc., San Jose, CA 95134

# Appendix

## Area-Delay Product

Table 2 shows the different values of clock period and corresponding area, and area-delay products.

|  |  |  |
| --- | --- | --- |
| Area (µm2) | Delay (ns) | Area\*Delay (ns ·µm2) |
| 926598.6 | 2.56 | 2372092.372 |
| 845136.6 | 2.7 | 2281868.694 |
| 926598.6 | 2.56 | 2372092.372 |
| 817226.8 | 3.12 | 2549747.653 |
| 813349 | 3.71 | 3017524.741 |
| 815359.9 | 4.17 | 3400050.93 |
| 773002.8 | 4.53 | 3501702.674 |
| 753269.2 | 5 | 3766346.014 |
| 745961.7 | 5.51 | 4110249.148 |
| 749387.2 | 6 | 4496322.921 |
| 730186.2 | 6.57 | 4797323.423 |

Table 2 – Area\*Delay products for final design

Fig-5 shows a graph for the values listed in Table 2.

Fig-5 – Delay vs. Area\*Delay product

* The delay as reported by Synopsys PrimeTime was **2.7 ns**
* The area reported by Synopsys DC Compiler was **845,136.55 µm2**
* The power reported by Synopsys DC Compiler was **79.4254 mW**

When optimizing the area-delay product, we were able to achieve a good reduction in this product by trying several different options within design compiler. By using the compile -map\_effort high -area\_effort high command, we noticed that design compiler is able to automatically map operators within our alu such that unused blocks within one operation are shared with another operation, thus reducing area. Using a for-loop within the design compiler tcl script, we obtained the above results for optimizing our area delay product.

## STA

The critical path reported was from *node0\_d\_in[15]* to *proc\_0/WB\_alu\_result\_reg[0]*.

The histograms generated from PrimeTime are shown in Fig. 2 and 3. Fig-2 shows the details for the worst group for path slack, and Fig-3 shows the details for endpoint slack.

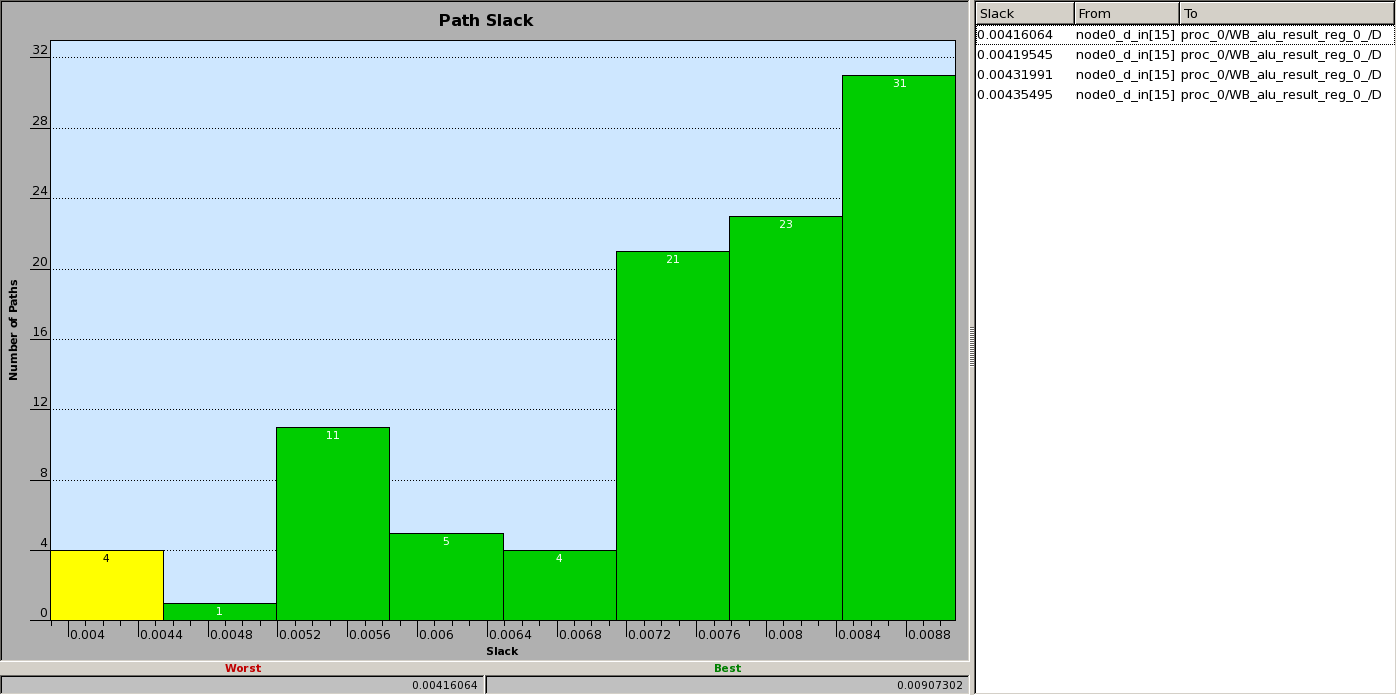


Fig-6 – Path slack histogram from PrimeTime

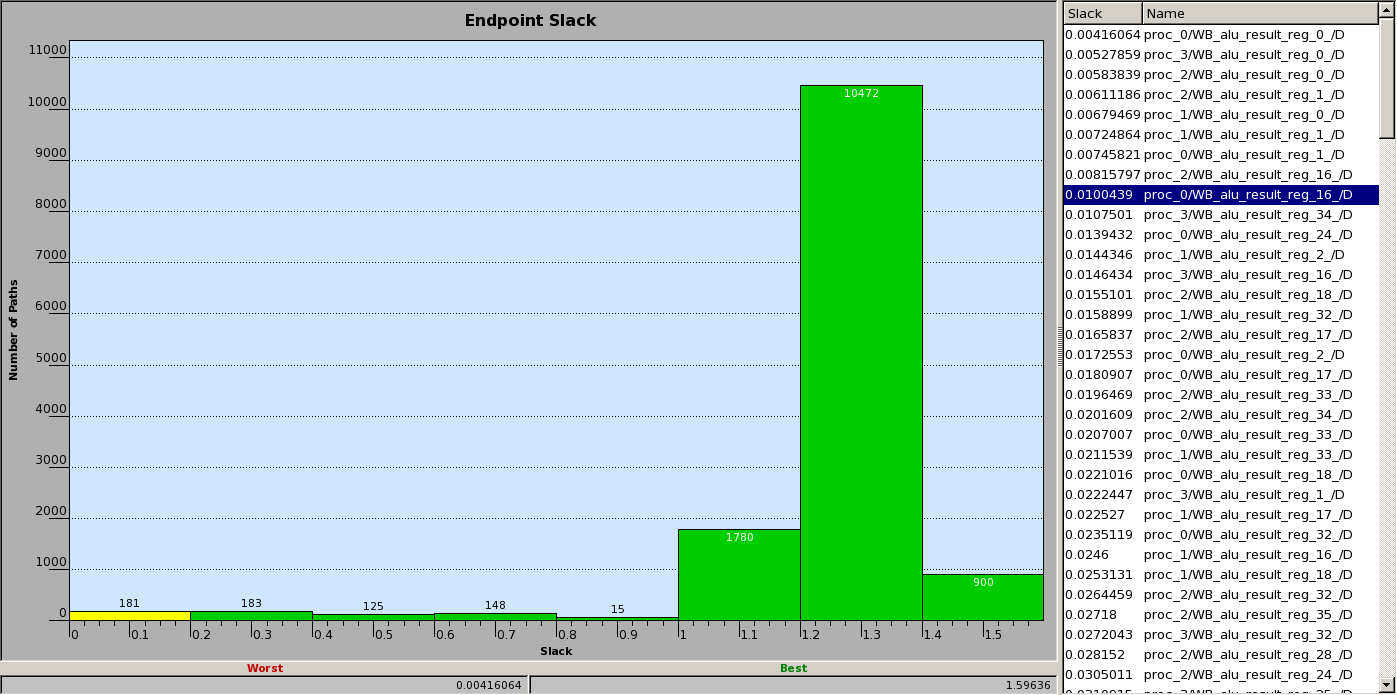


Fig-7 – Endpoint slack histogram from PrimeTime

## Place-and-route Summary

gold\_cmp\_preCTS.summary:

# generated on Fri Apr 22 01:16:57 2016

# Top Cell: gold\_cmp

------------------------------------------------------------

timeDesign Summary

------------------------------------------------------------

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

| Setup mode | all | reg2reg | in2reg | reg2out | in2out | clkgate |

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

| WNS (ns):| 3.040 | 3.040 | 3.717 | 3.542 | 6.410 | N/A |

| TNS (ns):| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | N/A |

| Violating Paths:| 0 | 0 | 0 | 0 | 0 | N/A |

| All Paths:| 13804 | 13348 | 13348 | 456 | 256 | N/A |

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

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

| | Real | Total |

| DRVs +------------------+------------+------------------|

| | Nr nets(terms) | Worst Vio | Nr nets(terms) |

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

| max\_cap | 12577 (12577) | -0.202 | 12577 (12577) |

| max\_tran | 0 (0) | 0.000 | 0 (0) |

| max\_fanout | 0 (0) | 0 | 0 (0) |

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

Density: 64.600%

Routing Overflow: 0.00% H and 0.43% V

------------------------------------------------------------

gold\_cmp\_postCTS.summary:

# generated on Fri Apr 22 01:16:57 2016

# Top Cell: gold\_cmp

------------------------------------------------------------

timeDesign Summary

------------------------------------------------------------

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

| Setup mode | all | reg2reg | in2reg | reg2out | in2out | clkgate |

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

| WNS (ns):| 3.040 | 3.040 | 3.717 | 3.542 | 6.410 | N/A |

| TNS (ns):| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | N/A |

| Violating Paths:| 0 | 0 | 0 | 0 | 0 | N/A |

| All Paths:| 13804 | 13348 | 13348 | 456 | 256 | N/A |

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

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

| | Real | Total |

| DRVs +------------------+------------+------------------|

| | Nr nets(terms) | Worst Vio | Nr nets(terms) |

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

| max\_cap | 12577 (12577) | -0.202 | 12577 (12577) |

| max\_tran | 0 (0) | 0.000 | 0 (0) |

| max\_fanout | 0 (0) | 0 | 0 (0) |

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

Density: 64.600%

Routing Overflow: 0.00% H and 0.43% V

------------------------------------------------------------

gold\_cmp\_postCTSHold.summary:

# generated on Fri Apr 22 17:31:48 2016

# Top Cell: gold\_cmp

------------------------------------------------------------

timeDesign Summary

------------------------------------------------------------

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

| Hold mode | all | reg2reg | in2reg | reg2out | in2out | clkgate |

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

| WNS (ns):| 0.116 | 0.116 | 1.055 | 1.070 | 2.414 | N/A |

| TNS (ns):| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | N/A |

| Violating Paths:| 0 | 0 | 0 | 0 | 0 | N/A |

| All Paths:| 13804 | 13348 | 13348 | 456 | 256 | N/A |

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

Density: 64.740%

Routing Overflow: 0.00% H and 0.50% V

------------------------------------------------------------

gold\_cmp\_postRoute.summary:

# generated on Fri Apr 22 17:52:01 2016

# Top Cell: gold\_cmp

------------------------------------------------------------

timeDesign Summary

------------------------------------------------------------

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

| Setup mode | all | reg2reg | in2reg | reg2out | in2out | clkgate |

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

| WNS (ns):| 5.584 | 5.584 | 6.759 | 6.367 | 9.813 | N/A |

| TNS (ns):| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | N/A |

| Violating Paths:| 0 | 0 | 0 | 0 | 0 | N/A |

| All Paths:| 13804 | 13348 | 13348 | 456 | 256 | N/A |

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

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

| | Real | Total |

| DRVs +------------------+------------+------------------|

| | Nr nets(terms) | Worst Vio | Nr nets(terms) |

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

| max\_cap | 12710 (12710) | -0.250 | 12712 (12712) |

| max\_tran | 0 (0) | 0.000 | 0 (0) |

| max\_fanout | 0 (0) | 0 | 0 (0) |

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

Density: 64.740%

------------------------------------------------------------

gold\_cmp\_postRouteHold.summary:

# generated on Fri Apr 22 17:55:36 2016

# Top Cell: gold\_cmp

------------------------------------------------------------

timeDesign Summary

------------------------------------------------------------

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

| Hold mode | all | reg2reg | in2reg | reg2out | in2out | clkgate |

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

| WNS (ns):| 0.117 | 0.117 | 1.055 | 1.071 | 2.514 | N/A |

| TNS (ns):| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | N/A |

| Violating Paths:| 0 | 0 | 0 | 0 | 0 | N/A |

| All Paths:| 13804 | 13348 | 13348 | 456 | 256 | N/A |

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

Density: 64.740%

------------------------------------------------------------

clock.report:

Nr. of Subtrees : 1

Nr. of Sinks : 13348

Nr. of Buffer : 443

Nr. of Level (including gates) : 5

Root Rise Input Tran : 0.1(ps)

Root Fall Input Tran : 0.1(ps)

Max trig. edge delay at sink(R): proc\_3/reg\_file/ram\_reg\_8\_\_60\_/CLK 777.1(ps)

Min trig. edge delay at sink(R): proc\_3/reg\_file/ram\_reg\_6\_\_22\_/CLK 558.2(ps)

(Actual) (Required)

Rise Phase Delay : 558.2~777.1(ps) 500~500(ps)

Fall Phase Delay : 516.1~810.2(ps) 500~500(ps)

Trig. Edge Skew : 218.9(ps) 300(ps)

Rise Skew : 218.9(ps)

Fall Skew : 294.1(ps)

Max. Rise Buffer Tran : 216.9(ps) 400(ps)

Max. Fall Buffer Tran : 211.6(ps) 400(ps)

Max. Rise Sink Tran : 228.8(ps) 400(ps)

Max. Fall Sink Tran : 166.4(ps) 400(ps)

Min. Rise Buffer Tran : 36.8(ps) 0(ps)

Min. Fall Buffer Tran : 67.9(ps) 0(ps)

Min. Rise Sink Tran : 167.8(ps) 0(ps)

Min. Fall Sink Tran : 111.4(ps) 0(ps)

summaryReport.rpt:

==============================

Floorplan/Placement Information

==============================

Total area of Standard cells: 725863.494 um^2

Total area of Standard cells(Subtracting Physical Cells): 725863.494 um^2

Total area of Macros: 0.000 um^2

Total area of Blockages: 0.000 um^2

Total area of Pad cells: 0.000 um^2

Total area of Core: 1121604.807 um^2

Total area of Chip: 1208237.966 um^2

Effective Utilization: 6.6206e-01

Number of Cell Rows: 427

% Pure Gate Density #1 (Subtracting BLOCKAGES): 64.717%

% Pure Gate Density #2 (Subtracting BLOCKAGES and Physical Cells): 64.717%

% Pure Gate Density #3 (Subtracting MACROS): 64.717%

% Pure Gate Density #4 (Subtracting MACROS and Physical Cells): 64.717%

% Pure Gate Density #5 (Subtracting MACROS and BLOCKAGES): 64.717%

% Pure Gate Density #6 (Subtracting MACROS and BLOCKAGES and Physical Cells): 64.717%

% Core Density (Counting Std Cells and MACROs): 64.717%

% Core Density #2(Subtracting Physical Cells): 64.717%

% Chip Density (Counting Std Cells and MACROs and IOs): 60.076%

% Chip Density #2(Subtracting Physical Cells): 60.076%

# Macros within 5 sites of IO pad: No

Macro halo defined?: No

==============================

Wire Length Distribution

==============================

Total metal1 wire length: 104243.8750 um

Total metal2 wire length: 1130025.7400 um

Total metal3 wire length: 1659804.5300 um

Total metal4 wire length: 1180514.5650 um

Total metal5 wire length: 724528.3500 um

Total metal6 wire length: 261362.8850 um

Total metal7 wire length: 28924.5750 um

Total metal8 wire length: 8338.2150 um

Total metal9 wire length: 354.4650 um

Total metal10 wire length: 593.5250 um

Total wire length: 5098690.7250 um

Average wire length/net: 20.0654 um

Area of Power Net Distribution:

------------------------------

Area of Power Net Distribution

------------------------------

Layer Name Area of Power Net Routable Area Percentage

metal1 30469.5768 1121604.8070 2.7166%

metal2 0.0000 1121604.8070 0.0000%

metal3 0.0000 1121604.8070 0.0000%

metal4 0.0000 1121604.8070 0.0000%

metal5 0.0000 1121604.8070 0.0000%

metal6 0.0000 1121604.8070 0.0000%

metal7 0.0000 1121604.8070 0.0000%

metal8 0.0000 1121604.8070 0.0000%

metal9 8761.9200 1121604.8070 0.7812%

metal10 52161.1200 1121604.8070 4.6506%