



EE577B PROJECT REPORT

Shishir Madhusudan

Shashank

Sai Prabhakar

TA: Aditya Deshpande, Abhay Patil

Instructor: Shahin Nazarian

December 5th, 2018

CONTENTS:

1. Introduction
2. Division of Tasks
3. PART I: CMP
 - A. Design Details
 - a. Router
 - b. Ring
 - c. ALU
 - d. Processor
 - e. NIC
 - f. CMP (Integrated Design)
 - B. Synthesis Analysis and Results
 - C. P&R Analysis and Results
 - D. PrimeTime Timing Analysis and Results
 - E. Performance Analysis and Optimizations
4. Part II: Hardware Accelerator
 - A. Hardware Accelerator Research
 - B. Design Details
 - C. Integration Details
 - D. Performance Analysis
5. Conclusion
6. Future Scope

INTRODUCTION

We have designed a 4 core Multi Processor and Network on Chip based router placed in a ring topology connecting the 4 processors for data exchange. The report starts with the Approach/Overview where the division of tasks among the teammates have been mentioned. Then the choices /tradeoffs have been briefly explained. Then implementation and results has been explained. The following section consists of the performance details of the design. Then the hardware accelerator has been explained. Then, we have the conclusion stating potential enhancements if we were given more and the lessons learnt.

APPROACH/OVERVIEW

Division of tasks:

-Phase 1(CMP)

- ✓ Router [Design, Verification] - Shashank
- ✓ Ring [Design, Verification] - Shashank
- ✓ ALU [Design, Verification] - Sai Prabhakar
- ✓ NIC [Design, Verification] - Shishir, Shashank
- ✓ Processor [Design, Verification] - Shishir, Sai Prabhakar

- ✓ Integration [Design]- Shashank
- ✓ Integration [Verification] - Sai Prabhakar

-Phase 2(Hardware Accelerator)

- ✓ Hardware Accelerator [Verilog Design, Verification] - Shishir, Sai Prabhakar
- ✓ Python Script for LFSR Matrix [Design, Verification] - Shashank

Design Partitioning

Phase 1:

Router

We have tried to create a highly modular design. In the router, we have 2 instances of the Input buffer channel, 1 instance of PE input buffer channel and 3 Output buffer channels. Every input/output channel has the even and odd register each 64 bit wide. The organization looks like this:

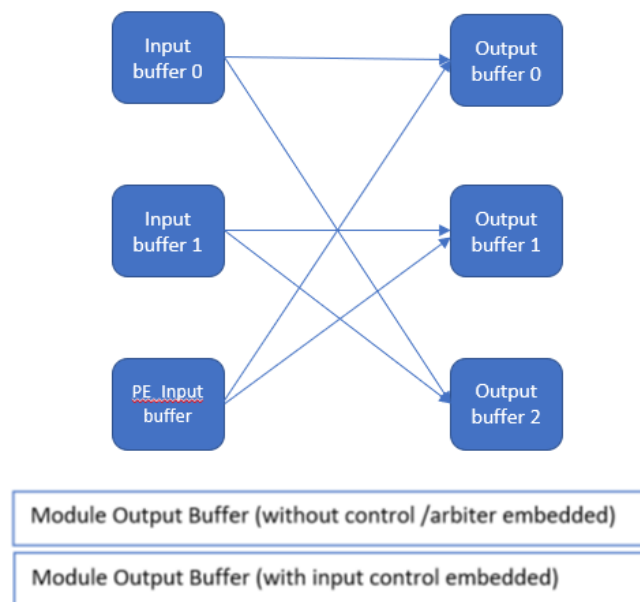


Figure 1. Router

Here the arbitration part of code is not a separate module but embedded into the Module Output Buffer handling the grant mechanism. We have some redundancy in for the processing element input buffer because of some issues faced during synthesis.

Processor

Here, we have PC, Register File and ALU as RTL modules each instantiated once in the CPU module. In processor design we have a 4-stage pipelined processor coded in behavioral way, with the DMEM module in Execution stage. Here, to be noted that the module DMEM.v has a stage register in itself, so, structurally the actual data memory is placed in the WB stage for read. An approximate (not all signals

and elements shown) structural view of our behavioral code is as follows, to help the reader to visualize the design:

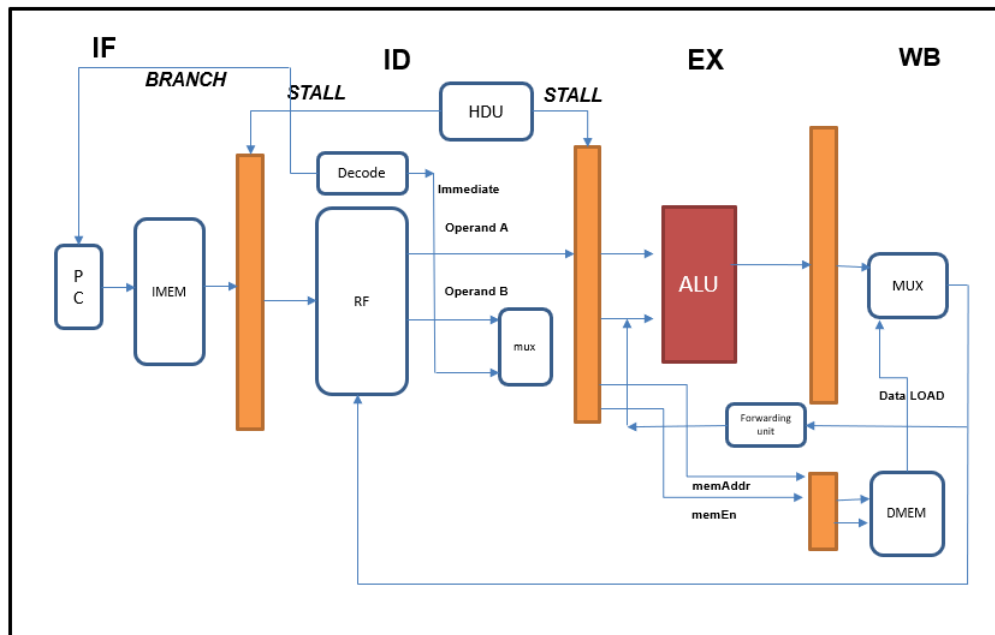


Figure 2

Network Interface Controller

The NIC has a simple design in which we are following the conditions for each signal as given in the manual. NIC is a single RTL module with no submodules in it.

CMP

The CMP has:

1. 4 instances of gold router
2. 4 instances of gold nic
3. 4 instances of gold processor
4. 4 instances of dmem
5. 4 instances of imem

Each gold processor has a dedicated dmem and imem.

The organization of these modules are given in the project manual.

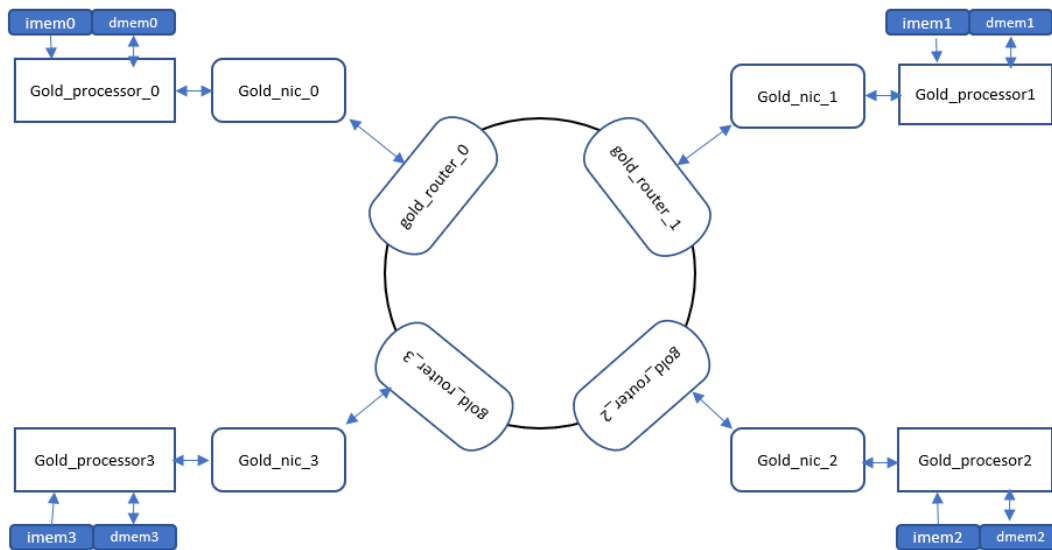


Figure 3

TRADE-OFFS/CHOICES

In every phase of the project, we tried to maintain the best hierarchy and modularity for our Verilog codes. However, in order to improve the debuggability and readability and due to issues faced while synthesis, some part of our design is redundant, especially in the router.

In processor we have PC, Register File and ALU as separate modules each instantiated once in the CPU module. The reason was to enable other team members (those who have not designed the CPU) to quickly grasp the design and ramp them for verification of the functionality. Also, it greatly helped us in debugging some issues related to the Internally Forwarding Register File logic. While debugging we could quickly segregate the sub-modules in CPU and other signals and registers in CPU, thus reducing the time complexity of our error search.

The testbenches were designed in an incremental manner. For e.g. in router we first checked a single packet on a single channel and then multiple packets on different virtual channels of same channel. This helped us to debug the issues quickly and accurately. Once these basic tests passed, we started stressing the design more by sending packets on three channels and then we checked the blocking case by filling all input and output buffers to check the consistency of the blocking and arbitration logic.

In the CMP, we performed different types of gather test, like all node to one node transfer, one to all node and all to all node transfer. We also tested the gather and computation test where CPU operated on a packet received from other core and then sent to other core for further computation. The rationale was to check the accuracy of the Router and NIC combination by observation whether the data after computation got corrupted or not when sent to the other core.

Similarly, for checking the CMP, packet transfer capability, we first built a dummy router and dummy processor attached to NIC and verified the handshaking between these elements and sent data to-and-fro from the dummy processor to the dummy router via the NIC design.

We are quite confident in our router and ring design as we checked them rigorously. The results of the 9 packets gather test and other tests during the demo were also correct. Though our design meets all the specifications in the manual and passed the demo, still no design is perfect especially post PnR, considering impact of the interconnect delays (RC) and the algorithms used for clock tree synthesis and optimizations. Since we have not exhaustively checked our design for all scenarios and data values due to time limitations, there could be some corner cases which may break the design Post-PnR. Also post fabrications PVT and electrical characteristics come into picture which may increase the vulnerabilities.

After developing and synthesizing each module separately, we performed the integration. Hence, Our verilog code was synthesizable after integration and performed the correct functionality both post synthesis and post place and route.

We tried several DC Compiler based optimization techniques, first being `<compile map_effort medium>`, which is the default setting, next we tried performing a high-effort incremental compile `< compile -map_effort high -incremental_mapping >`. To our surprise the latter gave up better results in Primetime ,data arrival time 8.02ns vs 12.04ns of the former, however worse Area * delay product in simulation of the post synthesis, reason being - to increase clock period, compiler uses higher-logic functions (more cells) which increases area significantly without decreasing the clock period by the same amount. Hence we tried some other set of options in the synthesis perl script, and finally we have used the following options to optimize the area- delay product:

1. `<compile map_effort high>` - This option enables DC Compiler to maximize its effort around the critical path by restructuring and re-mapping of logic, in order to meet the specified constraints
2. `<set_max_area 0>` - Instructs design compiler that uses as less area as possible.

We got some 1.25% reduction in area*delay of post synthesis and 1.35 % reduction after place and route. We performed around 4-5 iterations to get to the optimum value. We observed that in our design the time was not improving much but there was some improvement in area due to area-delay product. The 32 bit multiplier is the bottleneck in our design as it has not been pipelined.

The full data of timing and other metrics is available in the next two sections (Implementation, Performance).

IMPLEMENTATION

We were able to accomplish all the steps of the back-end-design stated in the project statement. At every step we made sure that all our designs are synthesizable and always took a modular and incremental approach. The observed some issues in the IFRF of the processor which was due to a wrong stall signal used for generating the data memory enable signal in the CPU code. Once we fixed that all processor functionalities started passing. We were able to simulate with correct functionality at each stage for both optimized and unoptimized designs.

We are showing the data for the optimized design:

Area of synthesized design as reported from Synopsys Design Compiler - 657039.233894 um sq.
Area after place & route - 897039.79235 um sq.

Percentage %increase - 36.52 %

Complete statistics in appendix

PERFORMANCE

	Critical Path (Primetime) (ns)	Delay (from TB) (ns)	Area * Delay (um. sq. ns)
post-synthesis	10.88	30	19711177.01682
post PnR	14.45	32	28705273.3552
% increase/slowdown	32.8 %	6.7%	45.6 %

When calculating the delay from testbench i.e. the minimum clock at which design gives correct results, we observe a slowdown by 6.5 %

The critical path delay pre and post PnR as shown in Primetime, slows down by 32.8%

The Area*Delay product increases by 45.6 %.

Hardware Accelerator

We have built a hardware accelerator for accelerating multiple CRC standards in individual cores of the Gold Chip Processor.

Motivation: In today's computer systems, various communication modules (Ethernet, WLAN, PCIe, USB, etc.) employing different CRC standards are integrated with the Central Processor Unit. Hence, usage of dedicated hardware accelerators for each application specific CRC standard is beneficial.

Description:

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption. CRCs can be used for error correction.

In our design, there will be a dedicated CRC block in each of the cores performing computation and CRC check for different CRC standards. The below schematic shows the organization of the CRC block.

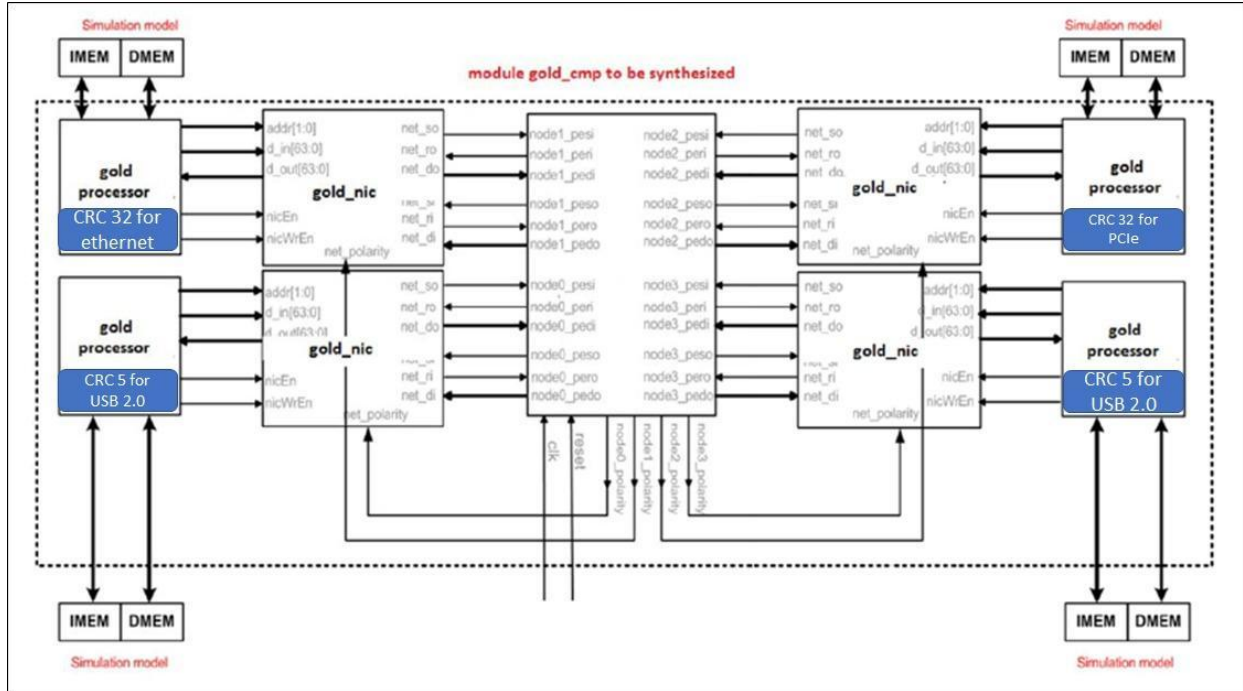


Figure 1. Top level Schematic after adding Hardware Accelerator blocks

The four CRC implementations are in accordance with the below standards:

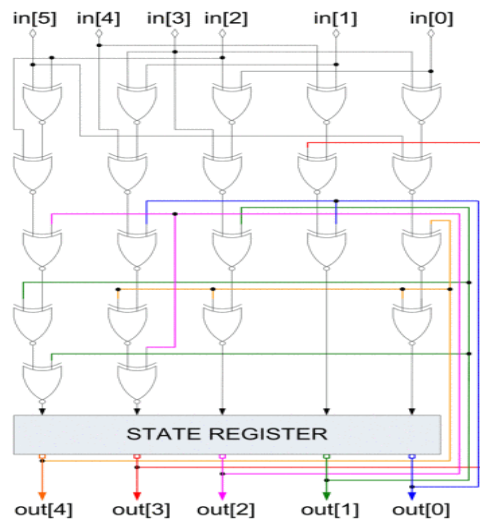
1. CRC32 for PCIe
2. CRC32 for IEEE802.3
3. CRC16 for USB2.0 (Data Packet)
4. CRC5 for USB2.0 (Token Packet)

We have the two additional instructions (common for all the cores) which will facilitate the computation and verification of the CRC are as follows:

1. `gencrc <data>` – This instruction would direct the targeted processor to accelerate the computation of the corresponding CRC standard for the given data.
2. `vercrc <data,crc>` - This instruction would direct the targeted processor to accelerate the computation of the corresponding CRC standard for the given data and verify if the data is erroneous.

Algorithm:

We have used a python script to generate a CRC matrix that is used as reference to develop CRC code. The matrix indicates which bits are to be involved in the XOR logic for each parallel CRC bit computation. An example of a parallel CRC circuit is as shown here.



Testing methodology:

CRC generation: Here we fill one of the memory location with the data for which CRC is to be generated and store it in another location. By comparing the generated CRC with that generated on the website: <http://www.ghsi.de/pages/subpages/Online%20CRC%20Calculation/>, we were able to verify the generated CRC.

CRC verification: Here we fill one of the memory locations with the data from the incoming data packet and another memory location with CRC extracted from the incoming packet. The CRC module would compute the CRC on the data packet and then compare the results with the CRC in the secondary memory location. The CRC module would provide a value '1' if the CRC values matches (i.e., the packet is error free), else the CRC module would provide a value '3'. By storing this result from the CRC module in one of the memory locations, we were able to observe the status of the packet validation.

In both cases, we tested for all the CRC polynomials in corresponding cores.

CONCLUSION

If we had more time, we would have pipelined the 32 bit multiplication in the ALU which would improve the design by a huge factor in terms of speed. On the hardware accelerator side we could have researched more and implemented some novel idea.

If power would be taken into consideration then we would have reduced the redundancies in our CPU/ALU which would give significant improvement in power savings.

We learnt the following lessons during the course of this project:

1. Debugging through signal dumps in text can be much rapid in comparison to debugging via waveforms in GUI mode.
2. We understood how to visualize design in terms of their behavior rather than structure which is difficult in big designs.

3. It is always better to move in step-wise / incremental way when designing and debugging.

Topics we would have like in 577b course:

1. Machine Learning Algorithm implementation in Verilog.
2. More detailed classes on Signal Integrity.

BIBLIOGRAPHY:

1. A Practical Parallel CRC Generation Method by Evgeni Stavinov
2. A tutorial on CRC computations by T.V. Ramabadran & S.S. Gaitonde
3. Pipelined Cyclic Redundancy Check (CRC) Calculation by Mathys Walma
4. High-Speed Parallel Architectures for Linear Feedback Shift Registers Manohar Ayinala

DEMO TEST DETAILS:

9 Packet Gather Test

Memory Location	dmem0.fill	dmem1.fill	dmem2.fill	dmem3.fill
0	0000000000000000	0000000000000000	0000000000000000	0000000000000000
1	1111111111111111	1111111111111111	1111111111111111	1111111111111111
2	2222222222222222	2222222222222222	2222222222222222	2222222222222222
3	3333333333333333	3333333333333333	3333333333333333	3333333333333333
4	4444444444444444	4444444444444444	4444444444444444	4444444444444444
5	5555555555555555	5555555555555555	5555555555555555	5555555555555555
6	6666666666666666	6666666666666666	6666666666666666	6666666666666666
7	7777777777777777	7777777777777777	7777777777777777	7777777777777777
8	8888888888888888	8888888888888888	8888888888888888	8888888888888888
9	9999999999999999	9999999999999999	9999999999999999	9999999999999999
10	80010000AAABBB00	80010001BBBCCC00	00010002CCCD00	00010003DDDA00
11	C0010000AAADDD00	C0010001BBBAAA00	40010002CCCB00	40010003DDDC00
12	C0020000AAACCC00	C0020001BBBDDD00	40020002CCCAA00	40020003DDDBB00
13	80010000AAABBB01	80010001BBBCCC01	00010002CCCD01	00010003DDDA01
14	C0010000AAADDD01	C0010001BBBAAA01	40010002CCCB01	40010003DDDC01
15	C0020000AAACCC01	C0020001BBBDDD01	40020002CCCAA01	40020003DDDBB01
16	80010000AAABBB02	80010001BBBCCC02	00010002CCCD02	00010003DDDA02
17	C0010000AAADDD02	C0010001BBBAAA02	40010002CCCB02	40010003DDDC02
18	C0020000AAACCC02	C0020001BBBDDD02	40020002CCCAA02	40020003DDDBB02
19	2D2D2D2D2D2D2D2D	2D2D2D2D2D2D2D2D	2D2D2D2D2D2D2D2D	2D2D2D2D2D2D2D2D

mem location	dmem0.dump	dmem1.dump	dmem2.dump	dmem3.dump
0	0000000000000000	0000000000000000	0000000000000000	0000000000000000
1	1111111111111111	1111111111111111	1111111111111111	1111111111111111
2	2222222222222222	2222222222222222	2222222222222222	2222222222222222
3	3333333333333333	3333333333333333	3333333333333333	3333333333333333
4	4444444444444444	4444444444444444	4444444444444444	4444444444444444
5	5555555555555555	5555555555555555	5555555555555555	5555555555555555
6	6666666666666666	6666666666666666	6666666666666666	6666666666666666
7	7777777777777777	7777777777777777	7777777777777777	7777777777777777
8	8888888888888888	8888888888888888	8888888888888888	8888888888888888
9	9999999999999999	9999999999999999	9999999999999999	9999999999999999
10	80010000aaabbb00	80010001bbbccc00	00010002cccd00	00010003ddd000
11	c0010000aaadd00	c0010001bbb000	40010002cccbbb00	40010003dddccc00
12	c0020000aaacc00	c0020001bbbddd00	40020002cccaaa00	40020003ddd000
13	80010000aaabbb01	80010001bbbccc01	00010002cccd01	00010003ddd001
14	c0010000aaadd01	c0010001bbb001	40010002cccbbb01	40010003dddccc01
15	c0020000aaacc01	c0020001bbbddd01	40020002cccaaa01	40020003ddd001
16	80010000aaabbb02	80010001bbbccc02	00010002cccd02	00010003ddd002
17	c0010000aaadd02	c0010001bbb002	40010002cccbbb02	40010003dddccc02
18	c0020000aaacc02	c0020001bbbddd02	40020002cccaaa02	40020003ddd002
19	2d2d2d2d2d2d2d2d	2d2d2d2d2d2d2d2d	2d2d2d2d2d2d2d2d	2d2d2d2d2d2d2d2d
20	3c3c3c3c3c3c3c3c	3c3c3c3c3c3c3c3c	3c3c3c3c3c3c3c3c	3c3c3c3c3c3c3c3c
21	4b4b4b4b4b4b4b4b	4b4b4b4b4b4b4b4b	4b4b4b4b4b4b4b4b	4b4b4b4b4b4b4b4b
22	5a5a5a5a5a5a5a5a	5a5a5a5a5a5a5a5a	5a5a5a5a5a5a5a5a	5a5a5a5a5a5a5a5a
23	6969696969696969	6969696969696969	6969696969696969	6969696969696969
24	7878787878787878	7878787878787878	7878787878787878	7878787878787878
25	00000003dddaaa00	40000002cccbbb00	80000001bbbccc00	00000002cccd00
26	00000003dddaaa01	40000003ddd000	40000003dddccc00	00000002cccd01
27	40000002cccaaa00	40000002cccbbb01	40000003dddccc01	00000002cccd02
28	00000003dddaaa02	40000003ddd001	80000001bbbccc01	c0000001bbbddd00
29	40000002cccaaa01	40000002cccbbb02	c0000000aaacc00	c0000000aaadd00
30	40000002cccaaa02	80000000aaabbb00	80000001bbbccc02	c0000000aaadd01
31	c0000001bbb000	80000000aaabbb01	40000003dddccc02	c0000000aaadd02
32	c0000001bbb001	80000000aaabbb02	c0000000aaacc01	c0000001bbbddd01
33	c0000001bbb002	40000003ddd002	c0000000aaacc02	c0000001bbbddd02
34	xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx

imem0	imem1	imem2	imem3
-------	-------	-------	-------

8020000A // VLD	8020000A // VLD	8020000A // VLD	8020000A // VLD
8040000B // VLD	8040000B // VLD	8040000B // VLD	8040000B // VLD
8060000C // VLD	8060000C // VLD	8060000C // VLD	8060000C // VLD
8080000D // VLD	8080000D // VLD	8080000D // VLD	8080000D // VLD
80A0000E // VLD	80A0000E // VLD	80A0000E // VLD	80A0000E // VLD
80C0000F // VLD	80C0000F // VLD	80C0000F // VLD	80C0000F // VLD
80E00010 // VLD	80E00010 // VLD	80E00010 // VLD	80E00010 // VLD
81000011 // VLD	81000011 // VLD	81000011 // VLD	81000011 // VLD
81200012 // VLD	81200012 // VLD	81200012 // VLD	81200012 // VLD
8140C001 // VLD	8140C001 // VLD	8140C001 // VLD	8140C001 // VLD
8D400024 // VBNEZ	8D400024 // VBNEZ	8D400024 // VBNEZ	8D400024 // VBNEZ
8420C000 // VSD	8420C000 // VSD	8420C000 // VSD	8420C000 // VSD
8160C001 // VLD	8160C001 // VLD	8160C001 // VLD	8160C001 // VLD
8D600030 // VBNEZ	8D600030 // VBNEZ	8D600030 // VBNEZ	8D600030 // VBNEZ
8440C000 // VSD	8440C000 // VSD	8440C000 // VSD	8440C000 // VSD
8180C001 // VLD	8180C001 // VLD	8180C001 // VLD	8180C001 // VLD
8D80003C // VBNEZ	8D80003C // VBNEZ	8D80003C // VBNEZ	8D80003C // VBNEZ
8460C000 // VSD	8460C000 // VSD	8460C000 // VSD	8460C000 // VSD
81A0C001 // VLD	81A0C001 // VLD	81A0C001 // VLD	81A0C001 // VLD
8DA00048 // VBNEZ	8DA00048 // VBNEZ	8DA00048 // VBNEZ	8DA00048 // VBNEZ
8480C000 // VSD	8480C000 // VSD	8480C000 // VSD	8480C000 // VSD
81C0C001 // VLD	81C0C001 // VLD	81C0C001 // VLD	81C0C001 // VLD
8DC00054 // VBNEZ	8DC00054 // VBNEZ	8DC00054 // VBNEZ	8DC00054 // VBNEZ
84A0C000 // VSD	84A0C000 // VSD	84A0C000 // VSD	84A0C000 // VSD
81E0C001 // VLD	81E0C001 // VLD	81E0C001 // VLD	81E0C001 // VLD
8DE00060 // VBNEZ	8DE00060 // VBNEZ	8DE00060 // VBNEZ	8DE00060 // VBNEZ
84C0C000 // VSD	84C0C000 // VSD	84C0C000 // VSD	84C0C000 // VSD
8200C001 // VLD	8200C001 // VLD	8200C001 // VLD	8200C001 // VLD
8E00006C // VBNEZ	8E00006C // VBNEZ	8E00006C // VBNEZ	8E00006C // VBNEZ
84E0C000 // VSD	84E0C000 // VSD	84E0C000 // VSD	84E0C000 // VSD
8220C001 // VLD	8220C001 // VLD	8220C001 // VLD	8220C001 // VLD
8E200078 // VBNEZ	8E200078 // VBNEZ	8E200078 // VBNEZ	8E200078 // VBNEZ
8500C000 // VSD	8500C000 // VSD	8500C000 // VSD	8500C000 // VSD
8240C001 // VLD	8240C001 // VLD	8240C001 // VLD	8240C001 // VLD
8E400084 // VBNEZ	8E400084 // VBNEZ	8E400084 // VBNEZ	8E400084 // VBNEZ
8520C000 // VSD	8520C000 // VSD	8520C000 // VSD	8520C000 // VSD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A600090 // VBEZ	8A600090 // VBEZ	8A600090 // VBEZ	8A600090 // VBEZ
8280C002 // VLD	8280C002 // VLD	8280C002 // VLD	8280C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A60009C // VBEZ	8A60009C // VBEZ	8A60009C // VBEZ	8A60009C // VBEZ
82A0C002 // VLD	82A0C002 // VLD	82A0C002 // VLD	82A0C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD

8A6000A8 // VBEZ	8A6000A8 // VBEZ	8A6000A8 // VBEZ	8A6000A8 // VBEZ
82C0C002 // VLD	82C0C002 // VLD	82C0C002 // VLD	82C0C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000B4 // VBEZ	8A6000B4 // VBEZ	8A6000B4 // VBEZ	8A6000B4 // VBEZ
82E0C002 // VLD	82E0C002 // VLD	82E0C002 // VLD	82E0C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000C0 // VBEZ	8A6000C0 // VBEZ	8A6000C0 // VBEZ	8A6000C0 // VBEZ
8300C002 // VLD	8300C002 // VLD	8300C002 // VLD	8300C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000CC // VBEZ	8A6000CC // VBEZ	8A6000CC // VBEZ	8A6000CC // VBEZ
8320C002 // VLD	8320C002 // VLD	8320C002 // VLD	8320C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000D8 // VBEZ	8A6000D8 // VBEZ	8A6000D8 // VBEZ	8A6000D8 // VBEZ
8340C002 // VLD	8340C002 // VLD	8340C002 // VLD	8340C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000E4 // VBEZ	8A6000E4 // VBEZ	8A6000E4 // VBEZ	8A6000E4 // VBEZ
8360C002 // VLD	8360C002 // VLD	8360C002 // VLD	8360C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000F0 // VBEZ	8A6000F0 // VBEZ	8A6000F0 // VBEZ	8A6000F0 // VBEZ
8380C002 // VLD	8380C002 // VLD	8380C002 // VLD	8380C002 // VLD
86800019 // VSD	86800019 // VSD	86800019 // VSD	86800019 // VSD
86A0001A // VSD	86A0001A // VSD	86A0001A // VSD	86A0001A // VSD
86C0001B // VSD	86C0001B // VSD	86C0001B // VSD	86C0001B // VSD
86E0001C // VSD	86E0001C // VSD	86E0001C // VSD	86E0001C // VSD
8700001D // VSD	8700001D // VSD	8700001D // VSD	8700001D // VSD
8720001E // VSD	8720001E // VSD	8720001E // VSD	8720001E // VSD
8740001F // VSD	8740001F // VSD	8740001F // VSD	8740001F // VSD
87600020 // VSD	87600020 // VSD	87600020 // VSD	87600020 // VSD
87800021 // VSD	87800021 // VSD	87800021 // VSD	87800021 // VSD
F0000000 // VNOP	F0000000 // VNOP	F0000000 // VNOP	F0000000 // VNOP
8020000A // VLD	8020000A // VLD	8020000A // VLD	8020000A // VLD
8040000B // VLD	8040000B // VLD	8040000B // VLD	8040000B // VLD
8060000C // VLD	8060000C // VLD	8060000C // VLD	8060000C // VLD
8080000D // VLD	8080000D // VLD	8080000D // VLD	8080000D // VLD
80A0000E // VLD	80A0000E // VLD	80A0000E // VLD	80A0000E // VLD
80C0000F // VLD	80C0000F // VLD	80C0000F // VLD	80C0000F // VLD
80E00010 // VLD	80E00010 // VLD	80E00010 // VLD	80E00010 // VLD
81000011 // VLD	81000011 // VLD	81000011 // VLD	81000011 // VLD
81200012 // VLD	81200012 // VLD	81200012 // VLD	81200012 // VLD
8140C001 // VLD	8140C001 // VLD	8140C001 // VLD	8140C001 // VLD
8D400024 // VBNEZ	8D400024 // VBNEZ	8D400024 // VBNEZ	8D400024 // VBNEZ
8420C000 // VSD	8420C000 // VSD	8420C000 // VSD	8420C000 // VSD
8160C001 // VLD	8160C001 // VLD	8160C001 // VLD	8160C001 // VLD

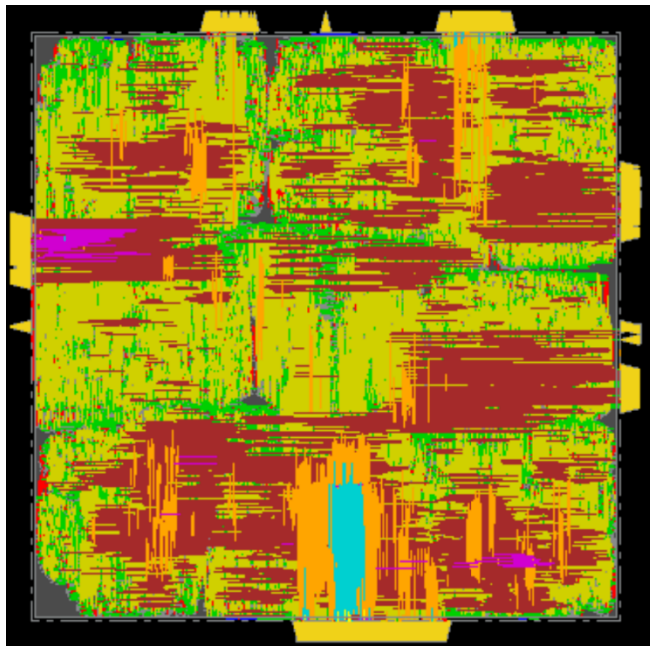
8D600030 // VBNEZ	8D600030 // VBNEZ	8D600030 // VBNEZ	8D600030 // VBNEZ
8440C000 // VSD	8440C000 // VSD	8440C000 // VSD	8440C000 // VSD
8180C001 // VLD	8180C001 // VLD	8180C001 // VLD	8180C001 // VLD
8D80003C // VBNEZ	8D80003C // VBNEZ	8D80003C // VBNEZ	8D80003C // VBNEZ
8460C000 // VSD	8460C000 // VSD	8460C000 // VSD	8460C000 // VSD
81A0C001 // VLD	81A0C001 // VLD	81A0C001 // VLD	81A0C001 // VLD
8DA00048 // VBNEZ	8DA00048 // VBNEZ	8DA00048 // VBNEZ	8DA00048 // VBNEZ
8480C000 // VSD	8480C000 // VSD	8480C000 // VSD	8480C000 // VSD
81C0C001 // VLD	81C0C001 // VLD	81C0C001 // VLD	81C0C001 // VLD
8DC00054 // VBNEZ	8DC00054 // VBNEZ	8DC00054 // VBNEZ	8DC00054 // VBNEZ
84A0C000 // VSD	84A0C000 // VSD	84A0C000 // VSD	84A0C000 // VSD
81E0C001 // VLD	81E0C001 // VLD	81E0C001 // VLD	81E0C001 // VLD
8DE00060 // VBNEZ	8DE00060 // VBNEZ	8DE00060 // VBNEZ	8DE00060 // VBNEZ
84C0C000 // VSD	84C0C000 // VSD	84C0C000 // VSD	84C0C000 // VSD
8200C001 // VLD	8200C001 // VLD	8200C001 // VLD	8200C001 // VLD
8E00006C // VBNEZ	8E00006C // VBNEZ	8E00006C // VBNEZ	8E00006C // VBNEZ
84E0C000 // VSD	84E0C000 // VSD	84E0C000 // VSD	84E0C000 // VSD
8220C001 // VLD	8220C001 // VLD	8220C001 // VLD	8220C001 // VLD
8E200078 // VBNEZ	8E200078 // VBNEZ	8E200078 // VBNEZ	8E200078 // VBNEZ
8500C000 // VSD	8500C000 // VSD	8500C000 // VSD	8500C000 // VSD
8240C001 // VLD	8240C001 // VLD	8240C001 // VLD	8240C001 // VLD
8E400084 // VBNEZ	8E400084 // VBNEZ	8E400084 // VBNEZ	8E400084 // VBNEZ
8520C000 // VSD	8520C000 // VSD	8520C000 // VSD	8520C000 // VSD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A600090 // VBEZ	8A600090 // VBEZ	8A600090 // VBEZ	8A600090 // VBEZ
8280C002 // VLD	8280C002 // VLD	8280C002 // VLD	8280C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A60009C // VBEZ	8A60009C // VBEZ	8A60009C // VBEZ	8A60009C // VBEZ
82A0C002 // VLD	82A0C002 // VLD	82A0C002 // VLD	82A0C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000A8 // VBEZ	8A6000A8 // VBEZ	8A6000A8 // VBEZ	8A6000A8 // VBEZ
82C0C002 // VLD	82C0C002 // VLD	82C0C002 // VLD	82C0C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000B4 // VBEZ	8A6000B4 // VBEZ	8A6000B4 // VBEZ	8A6000B4 // VBEZ
82E0C002 // VLD	82E0C002 // VLD	82E0C002 // VLD	82E0C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000C0 // VBEZ	8A6000C0 // VBEZ	8A6000C0 // VBEZ	8A6000C0 // VBEZ
8300C002 // VLD	8300C002 // VLD	8300C002 // VLD	8300C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000CC // VBEZ	8A6000CC // VBEZ	8A6000CC // VBEZ	8A6000CC // VBEZ
8320C002 // VLD	8320C002 // VLD	8320C002 // VLD	8320C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000D8 // VBEZ	8A6000D8 // VBEZ	8A6000D8 // VBEZ	8A6000D8 // VBEZ

8340C002 // VLD	8340C002 // VLD	8340C002 // VLD	8340C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000E4 // VBEZ	8A6000E4 // VBEZ	8A6000E4 // VBEZ	8A6000E4 // VBEZ
8360C002 // VLD	8360C002 // VLD	8360C002 // VLD	8360C002 // VLD
8260C003 // VLD	8260C003 // VLD	8260C003 // VLD	8260C003 // VLD
8A6000F0 // VBEZ	8A6000F0 // VBEZ	8A6000F0 // VBEZ	8A6000F0 // VBEZ
8380C002 // VLD	8380C002 // VLD	8380C002 // VLD	8380C002 // VLD
86800019 // VSD	86800019 // VSD	86800019 // VSD	86800019 // VSD
86A0001A // VSD	86A0001A // VSD	86A0001A // VSD	86A0001A // VSD
86C0001B // VSD	86C0001B // VSD	86C0001B // VSD	86C0001B // VSD
86E0001C // VSD	86E0001C // VSD	86E0001C // VSD	86E0001C // VSD
8700001D // VSD	8700001D // VSD	8700001D // VSD	8700001D // VSD
8720001E // VSD	8720001E // VSD	8720001E // VSD	8720001E // VSD
8740001F // VSD	8740001F // VSD	8740001F // VSD	8740001F // VSD
87600020 // VSD	87600020 // VSD	87600020 // VSD	87600020 // VSD
87800021 // VSD	87800021 // VSD	87800021 // VSD	87800021 // VSD
F0000000 // VNOP	F0000000 // VNOP	F0000000 // VNOP	F0000000 // VNOP
00000000	00000000	00000000	00000000

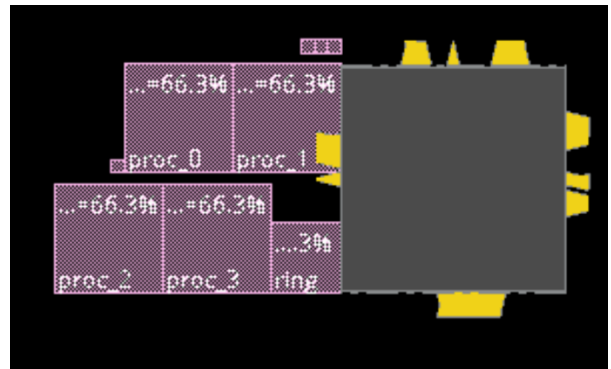
P&R SNAPSHOTS

The automatic place and route were done using Cadence Innovus. The three different views of the layout are shown in this section.

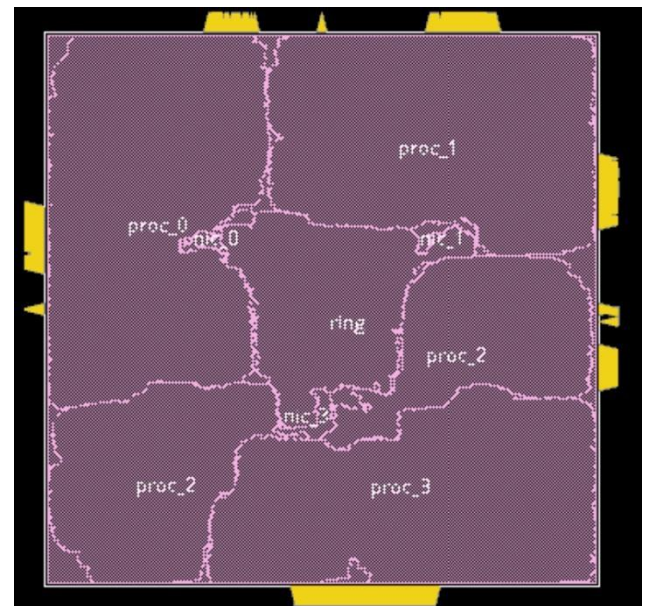
MANDATORY PART



Physical View

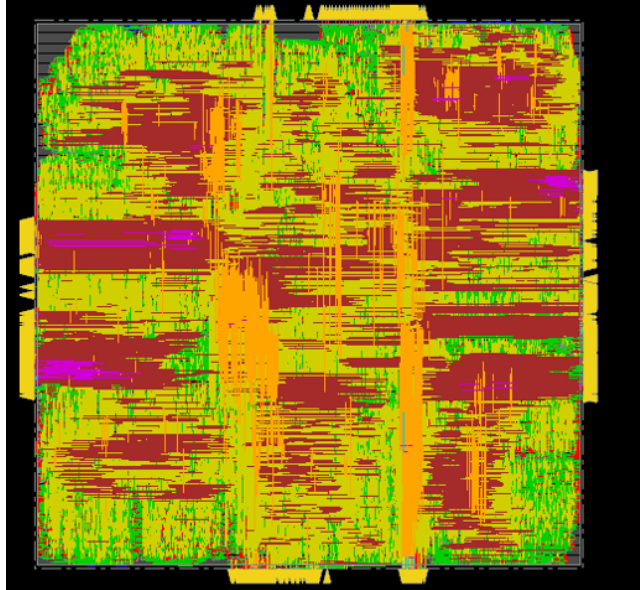


Floorplan View

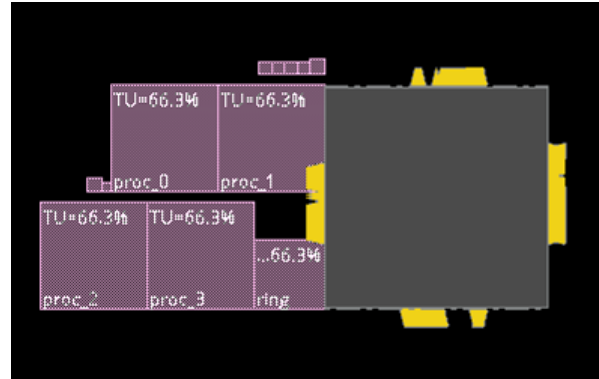


Amoeba View

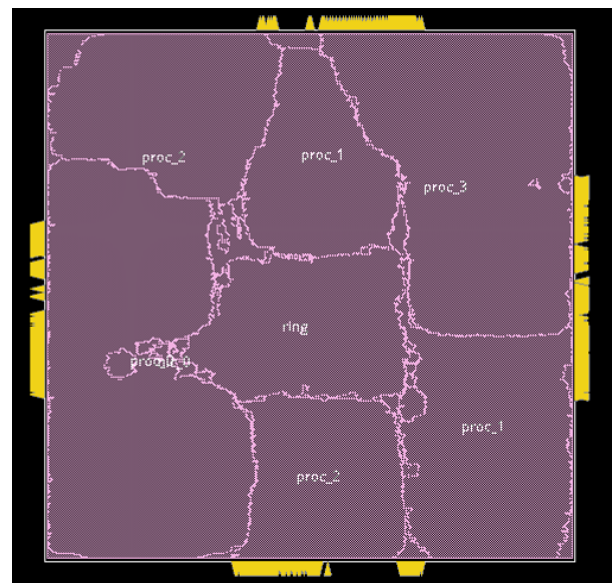
HARDWARE ACCELERATOR



Physical



Floorplan



Amoeba

Appendix

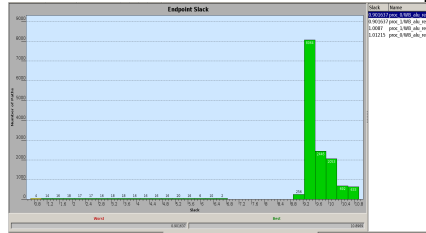
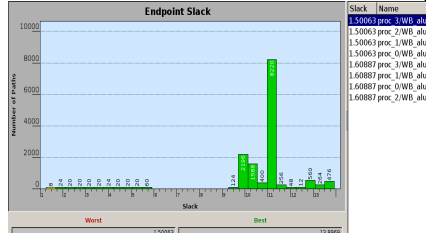
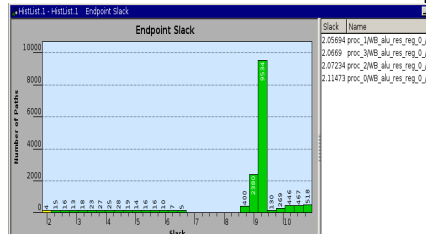
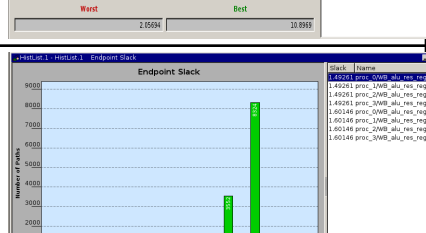
Synthesis Summary

Optimization State	Area (um sq.)	Timing (ns)	Check Design
With map effort medium (default)	665380.102568	Clk Period -> 12 ----- data required time 11.94 data arrival time -11.04 ----- slack (MET) 0.90	Pass [No Latches]
With map effort high and area constraint	657039.233894	Clk Period -> 10.6 ----- data required time 10.54 data arrival time -9.88 ----- slack (MET) 0.66	Pass [No Latches]

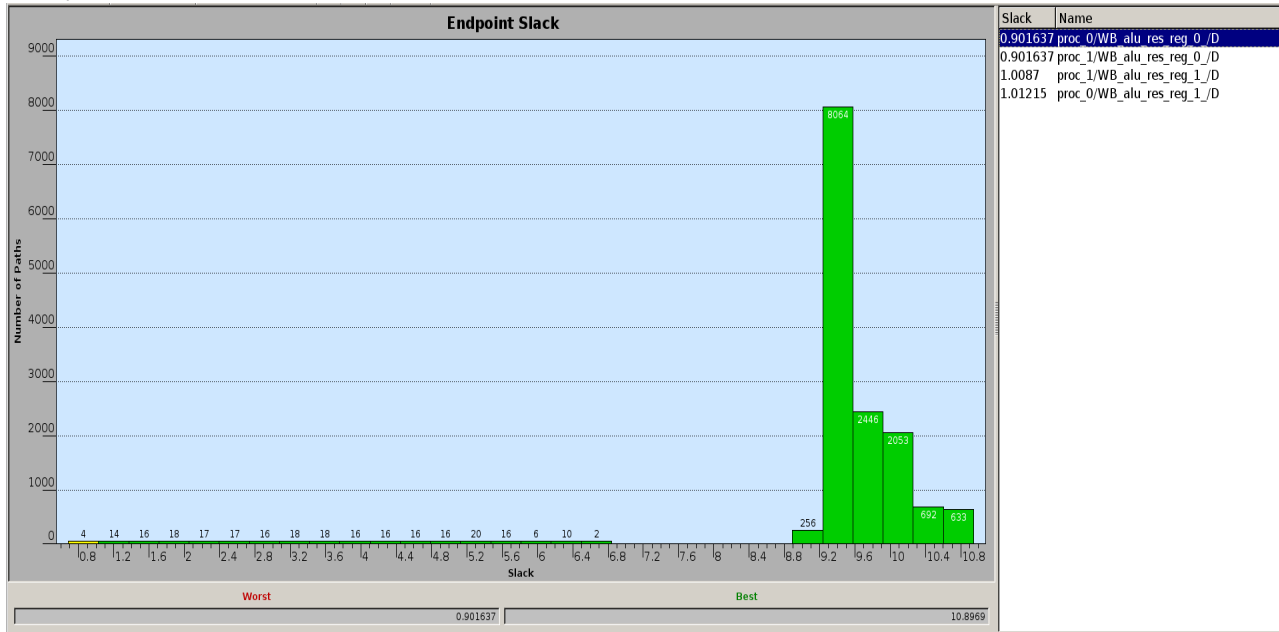
PNR summary

Optimization State	Stage	Area (um sq.)	Delay (ns)	Area * Delay (um. sq. ns)
With map effort medium(default)	Post -Synthesis	665380.102568	30	19961403.07704
	Post PnR	909190.8462	32	29094107.0784
	% Increase (due to PnR)	36.64 %	6.7%	45%
With map effort high and area constraint	Post -Synthesis	657039.233894	30	19711177.01682
	Post PnR	897039.79235	32	28705273.3552
	% Increase (due to PnR)	36.52 %	6.7%	45.6 %
Optimization advantage/summary	SYN (Area*Delay) decreasing by 1.25% PNR (Area*Delay) decreasing by 1.34%			

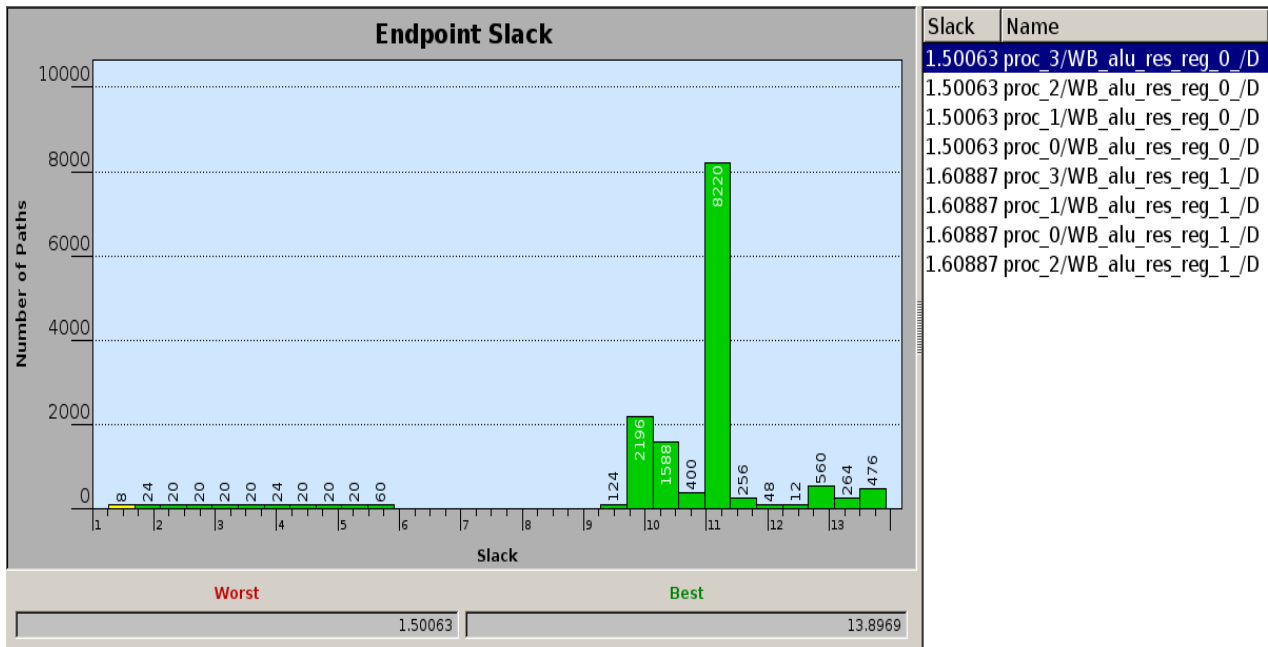
Primetime Summary

Optimizati on State	Stage	Clock Period (ns)	Data Required Time (ns)	Data Arrival Time (ns)	Slack (ns)	Worst Slack (ns)	Best Slack (ns)	End Point Slack Histogram
With map effort medium (default)	Post -Synthesis	12	12.94	12.04	0.90 [MET]	0.9016	10.8969	
	Post PnR	15	15.94	14.44	1.50 [MET]	1.50063	13.8969	
With Map effort High and area constraint	Post Synthesis	12	12.94	10.88	2.06	2.0569	10.8969	
	Post PnR	15	15.94	14.45	1.49	1.492	13.8969	

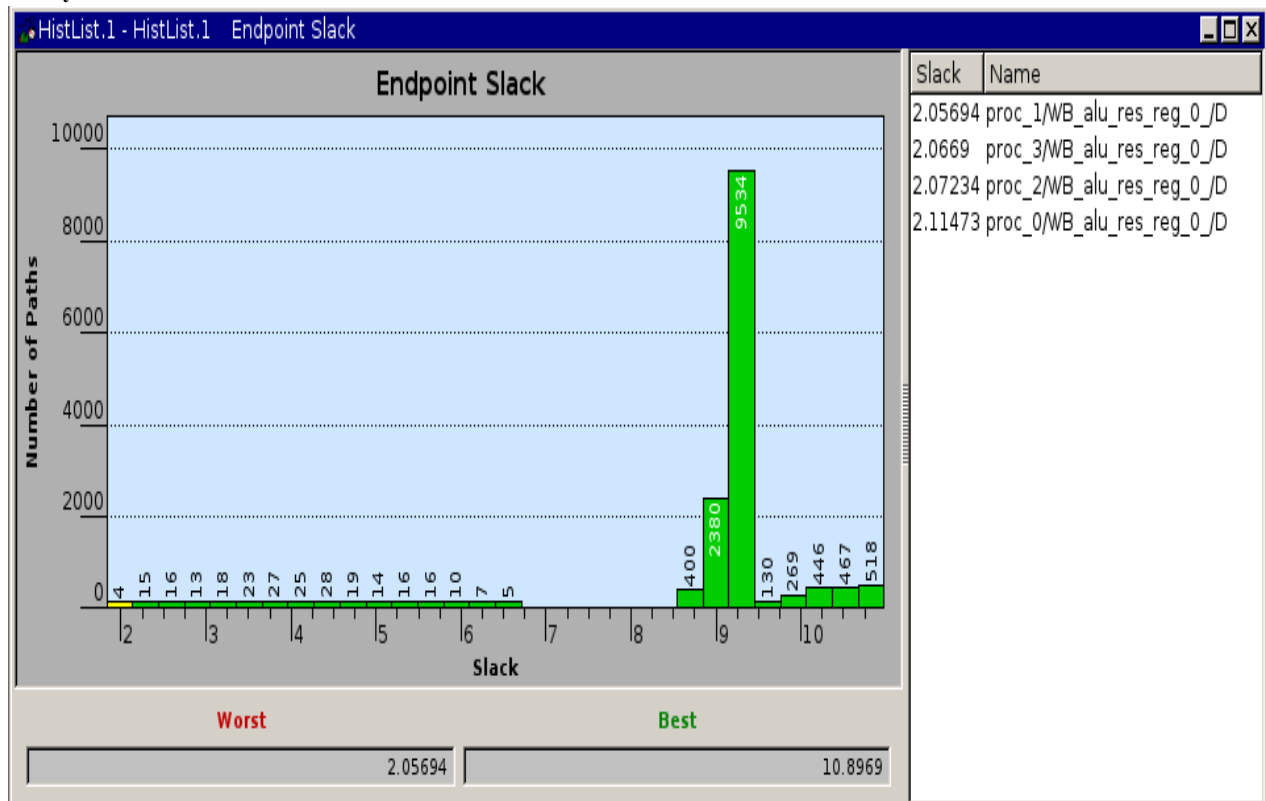
With map effort medium (default)
Post Syn



Post Pnr



With Map effort High and area constraint
Post syn



Post pnr

