Petabit Switch Fabric Design



Jingxue Zhou Jen-Hung Lo Yue Cao

Electrical Engineering and Computer Sciences University of California at Berkeley

Technical Report No. UCB/EECS-2016-105 http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-105.html

May 13, 2016

Copyright © 2016, by the author(s). All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

Special thanks to our advisors Vladimir Stojanovic, Elad Alon and our faculty consult committee John Wawrzynek. Also many thanks to the graduate students at BWRC who helped us tremendously with the tools setup for our project: Christopher Yarp, Angie Wang, Taewhan Kim, Paul Rigge, Ranko Sredojevic. Also, we would also like to thank to LBNL staffs Farzad Fatollahi-Fard and David Donofrio for letting us use their OpenSoC as our project baseline and giving us a presentation about their research

University of California, Berkeley College of Engineering MASTER OF ENGINEERING - SPRING 2016

Electrical Engineering and Computer Science Physical Electronics and Integrated Circuits Petabit Switch Fabric Design Jingxue Zhou

This Masters Project Paper fulfills the Master of Engineering degree requirement.

Approved by:

1. Capstone Project Advisor:

Signature: _____ Date _____

Print Name/Department: Vladimir Stojanovic/EECS

2. Faculty Committee Member #2:

Signature: _____ Date _____

Print Name/Department: John Wawrzynek/EECS

Table of Contents

Chapter 1 Technical Contribution

1. General Introduction	-3
2. Knowledge Domain	-4
3. Methods and Materials	- 8
4. Result and Discussion	-12
5. Conclusion and Future Work	-20

Chapter 2 Engineering Leadership

Appendix A	30
Work Cited	-28
4. Market Analysis	-26
3. Tech Strategy	-25
2. Industry Analysis	-23
1. Introduction	-22

Chapter 1 Technical Contribution

I. GENERAL INTRODUCTION

RESEARCH BACKGROUND

During the past ten years, commercial needs in data consumption have been greatly increased. Therefore, in order to support the rapid growth in data consumption, a great number of companies have been shifting to cloud computing because of its efficient resource allocation and less data cost. In fact, this trend has already stimulated several big companies such as google and facebook to develop their own data centers because of a great many advantages cloud computing can offer, such as economies of scale of large-scale datacenters and "pay-as-you-go" resource usage (Armbrust at el 2009). Thus, in order to ensure that these data centers can operate more efficiently, a robust and reliable infrastructure is necessary. One possible solution is to scale up the radix (number of ports) of the infrastructure to make sure more data has been going through.

PROJECT OVERVIEW

The ultimate goal of our project Petabit-Switch Fabric design is to design a router with higher radix to achieve better performance (more throughputs, low latency). To reach that, we explored several architectures and different radix numbers of the router and examined the designs on various performance metrics including area, power consumption, and timing. Our router design was implemented in chisel, which is a scala embedded hardware language developed recently by UC Berkeley. Our design process included four main stages (Fig. 1): I. Chisel and Router pre- learning; II. Open SoC code extraction; III. Sub-module and test harness design; IV. Front end and back end testing flow. Since both the chisel language and the router architecture were new topics to us, we've spent plenty of time doing pre-learning before actual designs during the first semester. Once we were capable of programming in chisel and understood the fundamentals in router design, we went through the OpenSoc fabric by LBNL, which is a network-on-chip generator design capable of creating a synthesizable network to connect processors, memory and I/O devices (OpenSoC, 2014). Our three main milestones during this period included learning chisel, studying the router infrastructure from William James Dally's book *Principles and Practices of Interconnection Networks*, and understanding the

OpenSoC code written by Lawrence Berkeley National Lab (LBNL). For the second semester, we designed the modules for a single router by assigning each router block to a group member. At this stage, we had a clear work breakdown as shown in Fig. 1: Yue was in charge of arbiter design; Jen-Hung was responsible for the routing function block which determine the travel route of flits from input to output; I was in charge of building the test harness around a single router. My work was based upon Yue and Jen-Hung's block designs of the router, with the purpose of verifying the functionality of their design. After we finished designing and testing the router blocks, we cooperated together and started our final front end and back end simulation.



Fig 1. Work breakdown for the design flow

II. KNOWLEDGE DOMAIN

OPENSOC

Our router design is based upon the OpenSoC fabric developed by LBNL. The OpenSoC fabric design, implemented in chisel, is a parameterized on-chip network generator with high hierarchy tree (see Fig. 2) (OpenSoC, 2014). The design includes the router design, the network at the top level as well as the test harness for each module and the whole network. It implements the flow control of the message using pipelined input-queued routers with round and robin arbiters and separable allocators (Fig. 3) (OpenSoC, 2014). Also, in

order to test how depth and concentration may affect the performance, two main topologies -- flattened butterfly and mesh are introduced. Fig. 3 introduces how the OpenSoC works: flits generated from some out sources such as processors first pass through the network interface and enter the injection queue. Depending on the topology we choose and the routing function we define, the flits will go through multiple nodes across the topology and arrive at the destinations as defined in the head flit. After the flits leave the router, they go through the ejection queue followed by the network interface. Since our design focused on a single router, our first work was to extract the router from the network hierarchy. Second, as one of our purposes was to find the router architecture with better performance, we modified two classes: arbiters and routing function and compared the results with those before the modification. For the arbiters (which control the access to the shared resources), Yue implemented two more types: carry-lookahead arbiter and matrix arbiter besides the original round and robin arbiter being used in OpenSoc. For the routing function which computes the travel route of the flits within the router network based on its destination information, Jen-Hung changed the approach to look-up-table to enable more topologies being instantiated. Lastly, since we modified the parameters and modules within the original classes in OpenSoc, I was responsible for developing the new test harness to ensure our single router was working well.



Fig 2. OpenSoC Fabric Hierarchy(OpenSoC, 2014)



Fig 3. OpenSoC Fabric Block Diagram(OpenSoC, 2014)

DESIGN BLOCKS

Router Overview: Fig. 4 is the block diagram for the router microarchitecture with VC channels. The main components within the microarchitecture include: Input Buffer, Routing Function, Virtual Channel (VC)

Allocator, Switch Allocator and Crossbar Switch. The input to the router is the data packet. A data packet can be divided into strings of flits. There are three main types of flits: head flits, body flits, and tail flits. Head flit is the most important one, which includes the essential information to define the packet, and declare the final destination while body flit and tail flit are followed by the head lit which mainly contain payloads and tail information correspondingly.

Input Buffer: Once the flit flows into the router, it will first be stored into the input buffer. At the same time, the route computation and allocation starts. After the computation and allocation are done, the flit will pop up the input buffer and enter the router network. Since it takes certain time to get the route computation and allocation done, it is important to keep the size (length) of the input buffer long enough to prevent the delay caused by the overflow at the input port.

Routing Function: As mentioned already, the head flit includes the destination information (output port) for the entire data packet. The routing function manages how the flits should flow from the input node to the output port depends on the algorithm being used in the design. The specific travel route being computed for a head flit correlates with the topology of the network. Unlike the mesh and flattened butterfly topologies which have already been defined in the OpenSoC, we didn't want to be restricted by one specific topology. In our design, we used a lookup table instead to approach more topologies.

Virtual Channel (VC) Allocator: Once the input flit has been stored in the input buffer, with output port being selected, it is also necessary to make sure that the VC at the output port is free to pass the flits. Again, as the head flit includes the essential information for the entire packet, body flit and tail flit will follow their head flit to the corresponding output port after the VC allocation has been completed.

Switch Allocator: After we've selected the output port and conduct the VC allocation, it is also critical to ensure that each output port has granted only one input port at a time. To achieve that, the switch allocator connects the input port and the output port via a crossbar link. At the same time, it measures the crossbar connection time for each flit.

Crossbar Switch: After the completion of the switch allocation, a grant signal will be sent to the crossbar switch. Once the connection between the input port and the output port has been established, the head

flit will travel from the input port to the selected output port buffer during the next cycle. The body flit and the tail flit should follow the same connection as the head flit in the next two cycles. The output buffer will release once the external downstream receiver is ready to receive the packets.



Fig 4. Router Microarchitecture (Becker, 2012)

III. METHODS AND MATERIALS

CHISEL IMPLEMENTATION

Fig. 8 shows the entire design flow for our project. At the top level, we have the chisel implementation for the RTL (Register Transfer Language) design. In current VLSI world, the most common language being used for RTL design is verilog. We chose chisel -- a relatively new hardware language over verilog for two main reasons. First, since chisel is embedded in scala, it includes a wider range of concepts based upon scalar library compared with verilog, such as object orientations and parameterized types. Secondly, chisel is able to generate both C++ based software simulator and verilog designed for VLSI flow (Fig. 7). On chisel side, Yue worked on the arbiter module: besides the original round and robin arbiter in OpenSoC, two more types of arbiters (carry-lookahead arbiter and matrix arbiter) were implemented. Jen-Hung extracted a single router from OpenSoC and implemented lookup table as the routing function for the switch. I was developing test harness around the extracted single router with lookup table to verify its functionality.

The harness module "SimpleVCRouterTester" tested a single router with 8 radix, 4 VC channels and 64 nodes. With the destination of the packet being defined in the testbench, I aimed to verify if the router was able to transfer the packet to the correct output channel with correct contents. The testbench was connecting to the I/Os of "SimpleVCRouterTestWrapper" module. Therefore, "SimpleVCRouterTestWrapper" module and "SimpleVCRouterTester" harness were linked together in the top level "main" module to enable test running.

Within the test harness module, each packet is instantiated with a head flit and a body flit as Fig. 5 shows. Since Jen-Hung has integrated the lookup table into the single router, the destination of the packet can be directly declared compared with the original router test with its destination in 3D coordinate system from LBNL. All the values for each term within the flitmap were chosen randomly. The "packetID" should be setted the same for headflit and bodyflit of the same packet. The "isTail" is an indication that if the packet has reached its last flit. After the definition of the input packet, the lookup table is instantiated as Fig 6. shows.

```
// ---- First Packet Definition ----
        headFlitMap_1 = LinkedHashMap(
                ("Dest"
                                         -> 0).
                ("packtType"
                                         -> 3 ),
                                         -> 0),
                ("vcPort"
                ("isTail"
                                         -> 0 ),
                ("packetID"
                                         -> 0 )
        )
        bodyFlitMap_1 = LinkedHashMap(
                ("payload"
                                         -> 0xBEEF ),
                ("flitID"
                                         -> 0xC ),
                ("vcPort"
                                         -> 0 ),
                ("isTail"
                                         -> 0 ),
                ("packetID"
                                         -> 0 )
        )
```



```
//Instatntiate look up table
val nums = (0 until c.numNodes).map(x => BigInt( (x + 3) % c.numRadix ))
for (i <- 0 until c.numNodes) {
    poke(c.io.lutWriteEnable, true)
    poke(c.io.lutWriteAddress, i)
    poke(c.io.lutWriteData, nums(i))
    step(1)
}
poke(c.io.lutWriteEnable, false)</pre>
```

Fig. 6 Lookup table instantiation within the harness

I wrote two sets of main tests for the single router with lookup table. The detailed chisel code is in Appendix A. For Test 1, a single packet with 1 headflit and 1 bodyflit was driven to the router. Each output channel is checked during the test to guarantee the packet has been delivered to the expected output port with the right content without mismatching the rest of the output ports. Test 1.5 is an extension of Test 1. It drove a single packet with 1 headflit and multiple bodyflits. The purpose of this test is to ensure that the router is able to deliver the longer packet with the right order of its flits to the expected output port. In Test 2, multiple packets, each with 1 headflit and 1 bodyflit were pushed into the router one by one. I did two sub-tests in this set (refer to Test 2 and Test 2.5 in Appendix A): one with 2 packets and one with 8 packets. The goal of this test is to see as the pushing pressure goes up if the router is still able to function correctly without messing up the destination, the order and the contents of different packets. After verifying the function of the single router, we were able to move to the "CMesh_CombinedTester_VarInjRate_lut" harness around the whole switch design. After all the software simulation was done, we transferred our router design to verilog through sbt tool.

ASIC DESIGN FLOW

Overview: After we successfully obtained the verilog code generated and verified from chisel, we followed the standard ASIC (Application Specification Integrated Circuit) design flow as shown in Fig 8. At the front end, we had RTL design in verilog based on the function specification we want (which has already been generated from chisel), logic synthesis with gate level netlists was generated after dc synthesis. At the back end, we had floor planning and place-and-route.

RTL Design: As already been discussed in the previous paragraph, the verilog code was generated from the chisel code we've written.

Logic Synthesis: At this stage, the verilog generated from chisel was transferred to gate level netlists. More specifically, standard logic gates including INV, AND, OR, XOR etc are used to construct the actual circuits to realize the implementation described in our RTL code. The logic synthesis process was performed on dc compiler tool from Synopsys. During the dc synthesis, we got reports and results corresponding to timing, power, and area cost. We used these reports in our result discussion later because they were good

indications on the performance of the design. Yet they were not the final results as we still missed the placeand-route information.

Floor-planning and Place and Rout: This is the back end simulation we will do during the asic design flow. At this stage, the gate level netlists will be mapped into the layout plan. The design tool being used at this stage is ic compiler from Synopsys. By editing the scripts within the tcl files, a wide range of optimizations will be implemented, such as clock gating and clock tree synthesis with the ultimate goal to achieve better performance and less power consumption. The timing report and power report generated at this stage are more reliable and are accounted as our final results. This is part of the future work we will work on.



Fig 7. Chisel Design Flow(OpenSoC, 2014)



Fig 8. ASIC Design Flow

IV. Result and Discussion

As we discussed in the previous sections, our goal is to find the radix of the router with better performance, less area cost, and less power consumption. Since we were only able to push our router design up to 64 radix through software simulation, we did the extrapolation for the router with radix higher than 64 based on current results we obtained from software simulation and dc synthesis. All the results we obtained are all based on the following assumptions: 1. the number of nodes is held constant at 256; 2. the flit size is fixed at 55; 3. the number of the Virtual Channels is 2; 4. the injection rate is 10%. The variables during the test are types of arbiters and the number of radix.

In order to measure the performance, we defined the following four critical parameters and compared them across the routers having different combinations of arbiters choices and radix selection:

Latency (cycle)=Avg(SW_{cycle-64packets})

This parameter indicates the average amount of time one packet takes to travel from the starting point to the destination. During Latency measurement, we injected 64 packets to each port and recorded the software cycles of each packet. The latency is calculated by taking the average of the software cycles of these packets.

No.Radix*Flit size *Channel Utilization Throughput (bits/s)= Clock Period

This parameter specifies the data bit transferring rate per second of our router. The channel utilization is included in the computation for a more real throughput value.

This is the key parameter we used to measure the router performance because we would like to achieve better throughputs while less latency for an ideal choice.

Arbiter Choice

In our testing, three types of arbiters were implemented: round and robin arbiter, carry-lookahead arbiter, and matrix arbiter. We pushed the routers of different arbiters through dc compiler and examined their performance under three design metrics: power consumption, area cost, and throughput/latency rate.

We collected results of three arbiters by pushing the router up to 8 radix. Table 1, 2, and 3 summarizes the area cost, power consumption and Throughput/Latency rate of the three arbiters. We can clearly see that carry-lookahead arbiter stands out among the three: although the power consumption from carry-lookahead arbiter is comparable to the other two, it has the highest Throughput/Latency rate while taking the least area cost.

Area(um^2)	2	4	8
RR Arbiter	104491.3082	223193.841	486472.7992
Matrix Arbiter	105031.8725	229982.0274	548598.3014
CL Arbiter	104736.5572	221551.0542	476628.531

Table 1. Area cost when using different arbiters with the increase in radix up to 8

Power(uW)	2	4	8
RR Arbiter	24800	48400	95600
Matrix Arbiter	24800	50700	123000
CL Arbiter	26500	50700	101000

Table 2. Power consumption when using different arbiters with the increase in radix up to 8

Throughput/latency(Gb/ (s*cycle))	2	4	8
RR Arbiter	1.526	2.300	3.485
Matrix Arbiter	1.400	2.240	3.948
CL Arbiter	1.633	2.641	4.282

Table 3. Performance measurement (Throughput/Latency) when using different arbiters with the increase in
radix up to 8

Besides performance metrics, it is also important to consider fairness of the three arbiters by examining the data distribution pattern. Fig. 9, Fig. 10 and Fig. 11 represent the latency histogram of the three arbiters when injecting 1024 packets at 10% rate. By comparing these three figures, we can see that round and robin arbiter and matrix arbiter both have a relatively fair data distribution, while carry-lookahead arbiter has a higher peak on the left and a longer tail on the right of its data distribution. It indicates that carry-lookahead arbiter transfers most of packets with a short delay but leaves a few to be waiting for a long time. The reason is that those channels with the lowest priorities in carry-lookahead arbiter may encounter the problem of starvation of some packets because carry-lookahead arbiter has fixed priority (Yue 2016). On the other hand, round and robin arbiter and matrix arbiter don't have this problem because they rotate the priority around all the channels. Despite this disadvantage in the arbitration fairness, carry-lookahead arbiter still wins because of

its outstanding performance in throughput/latency, less area cost, and fair power consumption. Therefore we choose carry-lookahead arbiter in the rest of our analysis when we scale up our radix.



Histogram of Packet Latency under Matrix Arbiter





Histogram of Packet Latency under RR Arbiter

Fig 10. Packet latency histogram for RR Arbiter

Histogram of Packet Latency under Lookahead Arbiter



Fig 11. Packet latency histogram for Carry-lookahead Arbiter

Radix Choice

As we determined our arbiter choice as carry-lookahead, we again pushed our router with various radix numbers from 2 to 64 through dc compiler, and extrapolated the results for radix higher than 64 (up to 128). We chose the best radix based on the same performance metrics -- Throughput/Latency rate, power consumption and area cost we've been using in the arbiter choice section.

Fig. 12, Fig. 13 and Fig. 14 show the average packet latency, throughput, and Throughput/Latency results as we scaled up the radix. We didn't include the radix 2 result for extrapolation as it deviated too much from the result trend of other radix number. A potential cause may be that the sample size is so small which causes big variations. From Fig. 14, we can see that radix 64 yields the best performance on Throughput/Latency rate.



Average packet latency vs radix

Fig 12. Average packet latency vs Number of radix plot









Throughput/latency vs radix

Fig 14. Throughput/Latency vs Number of radix plot

Besides Throughput/Latency rate, we are also interested in examining if the power consumption and the area cost of a radix 64 router are fair. From the post-synthesis report, total area cost is 7736093.7304 um^2 and the total power consumption is 1.32W. Fig. 15 and Fig. 16 show the area and power distribution of the router of radix 64. From the figure we can see that buffer takes the largest portion of the area as well as the greatest power among all the main components because the whole buffer portion includes both the injection and ejection queues and VC buffers, which grow linearly with the radix number. It is noticeable that switch takes the second largest area cost. Since its inner complexity grows quadratically with the radix number, we predicted that switch may overtake buffer and become the largest component as we keep scaling up the radix.

In short, 64 is an optimum radix choice for its highest Throughput/Latency over other radix choices on the performance perspective. For power consumption and area cost, although the results are large, it is reasonable and affordable under this radix number.



Fig 15. Power distribution of post-synthesis router with radix 64



Fig 16. Area cost distribution of post-synthesis router with radix 64

Post place-and-route result

Because of the hardware limitation, we've only pushed our router design up to radix 16 to ic compiler for place-and-route results. Fig. 17 shows the layout for our router design of radix 16. The green portion represents the lookup table; the red portion represents the switch; the yellow portion represents switch allocator; the orange portion represents VC allocator; the blue portion represents buffers. From the icc report, the clock period for post place-and-route router design of radix 16 is 2.91ns; the total power consumption is 0.275W; the total area cost is 868896.994713um^2.



Fig 17. Post place-and-route layout for router of radix 16

V. Conclusion and Future Work

Throughout the project, we explored various router architectures as well as number of radix to reach better performance metrics. In order to approach the design, we extracted single router from the network of OpenSoC fabric and integrated lookup table into the design. We also implemented two more arbiters -- carrylookahead arbiter and matrix arbiter besides round and robin arbiter and compared their performance in dc synthesis. We analyzed performance based upon three metrics: Throughput/Latency rate, power consumption, and area cost. During dc synthesis, carry-lookahead arbiter has been proven to have the best performance among the three. We also reached the optimal radix of 64 in dc synthesis using the above three performance metrics. Since the results we obtained in this report, including Throughput/Latency, area cost, and power consumption correspond to post synthesis results, one of the main future work is to push the router with radix 64 through ic compiler tools to reach final results. (So far, we only pushed router through ic compiler up to radix 16.) To achieve that, a potential solution/ future work is to use hierarchical synthesis where modules can be synthesized individually and then be combined on the top level and be synthesized together again.

Chapter 2 Engineering Leadership

Introduction

Driven by the growing demand for faster processing speed in recent years, chip companies such as Intel and AMD have turned to multi-core CPUs as the solution to scaling system performance (Wolfe, 2009). Unlike single-core processors, multi-core processors integrate hundreds or thousands of processing elements together on small chips. Given the physical proximity of myriads of processors on a single die, significant boost in performance can be achieved while maintaining minimal communication latency. As the number of architectural elements integrated on a single die continues to grow, the network-on-chip (NoC) implementation becomes the major bottleneck in how fast the multicore chip can operate (Becker, 2012). Network-on-chip is essentially the communication system integrated directly on the chip that ties all the processors, memories and external devices together. Figure 1 illustrates a multi-core NoC platform that features multiple cores, memories and other devices linked together by a central NoC switch fabric.



Figure 1. Multi-core Network-on-Chip Layout (Benini, 2007)

The switch fabric itself consists of several network nodes, or routers, that are interweaved together in certain geometrical topology to make up the entire NoC system. Hence, the times it takes to communicate between two network endpoints ultimately depends on the number of router hops along the path of data

traversal (Dally, 2004). The numbers of router hops are directly related to the number of ports –or radix – of a router, and by scaling up the radix of a router we can connect additional endpoint devices and communicate with fewer router hops, thus achieving the level of efficiency required by a multicore system. However, there exist design tradeoffs within router microarchitecture that limit the scope of radix's scalability, hence marking a point of diminishing return in network quality.

Our project, Petabit Switch Fabric Design, thus is to experiment and analyze the design tradeoffs in question and observe how they may help or hinder the performance of a router as it scales up its radix. Using the router design prototype based on the open source code developed by Lawrence Berkeley National Lab as a baseline, we will be investigating the ways in which different parameters may impact the performance of the router design. Ultimately, our end goal is to find the most efficient configuration for high radix router.

Industry Analysis

One of the biggest current technology trends is the shift towards cloud computing. Major companies like Dell, Microsoft, and Amazon have started to provide cloud computing services. For example, Dell announced the Dell Private Cloud Solution, which is powered by Intel architecture, and provides infrastructure that helps to reduce ownership cost by having superior automatic allocation of computing resources (2016). Instead of managing their own localized hardware, enterprises can rent data computing resources from these big companies to obtain more flexible resources and to reduce overall cost (Hassan, 2011).

Such trends lead to the collection of data computing resources towards the few big companies mentioned above. To provide the storage for such a large amount of resources, these companies need to construct data centers with warehouse-scale computers (WSC), that is, warehouses full of supercomputers interconnected together. In order for all the computers within such a warehouse to communicate to each other and to the outside world while maintaining high performance, having powerful interconnection infrastructure is extremely critical. Hence, these data center giants become obvious target customers for our high-speed router.

To assess the profitability of this product, we will use the Porter's five forces model: new entrants, substitutes, buyers, suppliers, and existing rivals (2008). Firstly, consider the force of the new entrants. Routers are highly specialized pieces of hardware that are sold in the form of chips. The biggest part of the chip cost is the non-recurring engineering cost, which is the one-time cost for a chip design, so the overall cost of the chips will decrease drastically when increasing the sale amount. However, it is hard for new entrants to sell as many chips as the existing companies. Therefore, the new entrants have a critical cost disadvantage and thus the effect should be weak. Secondly, the substitute of a router chip is its software counterpart. Nowadays routers are a combination of software and hardware so as to fill in the shortcomings of each other. For example, Broadcom's Trident II ASIC switch is currently being used as top-of-rack switch configuration in Facebook's Wedge and FBOSS. Wedge is the physical hardware of the top-of-rack switch and FBOSS is the software agent that controls the ASIC (Simpkins, 2014). Therefore, the effect of the substitute software should be weak. Thirdly, the bargaining power of suppliers (the chip manufacturing companies) and the customers (warehouse-scale data centers) are quite strong since they don't come in high volume.

Finally, the rivals of our products are the products from existing network companies such as Cisco, Juniper and Broadcom. Since these companies are already firmly established in the networking landscape, the force of rivalry is strong. Fortunately, these companies are providing products with strong features instead of strong cost advantage, which may not have a great impact on the market price. For example, Broadcom announced the StrataXGS Tomahawk[™] Series in September of 2014. This chip is used for Ethernet switch for cloud-scale network and the promised bandwidth is 3.2 terabits per second (Broadcom, 2014). This product can support from 32 to 128 ports based on the speed of Ethernet, and the data transfer rate of the data center network can be largely improved while keeping the same cabling complexity and equipment footprint (Broadcom, 2014). This is a good example of competitors with powerful features.

After considering these five forces, we can see that except the rivalry force, we have two strong and two weak forces. As for rivalry, the strong force towards feature usually improves the profitability of the industry. However, our product will be a new entrant, which is determined as a disadvantage previously. In general, the profitability of our product should be on average level since the five forces are almost balanced.

Based on the profit trend convention of rivalry above, we should focus on developing strong features to further improve the overall profit. Meanwhile, since it will be hard for us to compete with the existing strong rivals on all kinds of features, we should first concentrate on a niche market and design our product with few special features.

Tech Strategy

As mentioned previously, there is a clear indication in the current trend that enterprises and consumers alike are moving towards cloud services and solutions. A little more than a decade ago however, this trend was less obvious and most companies were still using localized servers with switches and routers they are managed individually (Morgan, 2015). Google, a pioneer in distributed computing and data processing, was the only company that foresaw the need of transformative networking technology required by the increasingly powerful computing infrastructure. Indeed, for the past decade or so, Google has been developing and deploying its own networking infrastructures to complement the computing power required from Google's large-scale cluster architecture starting from Google File System in 2002 to Spanner in 2012.

Armin Vahdat, the technical lead for networking at Google, succinctly described this mutual dependency between network and computing in his keynote in ONS 2015: "Networking is an inflection point and what computing means is going to be largely determined by our ability to build great networks over the coming years (2015)". By discovering before everybody else that traditional network was not able to scale up to meet the computing requirements in the near future and proactively improving and transforming their network infrastructure in response to the growing bandwidth demands from their servers, Google was able to become one of the biggest players in the computing industry today.

With the advancement of memory technology – for example, the 3D XPoint nonvolatile memory that offers up to 1,000 times the speed and up to 10 times the storage (Intel, 2015) – playing a major role in the future scene of datacenters, it is imperative for the networking technology to evolve even further than before. Vahdat has predicted in his keynote that a 5 Petabit per second network, in comparison to the Gigabit per second network commercially available today, may be needed in the near future (2015). Currently, Google's

latest-generation network Jupiter employs high-radix switch with 128 ports and 40 Gigabytes per port, allowing it to deliver 1.3 Petabit per second (Singh et al, 2015). In light of the successful deployment of highradix switch from Google and Vahdat's foresight on networking trend, our team pursues to find the optimal high-radix router architecture that enables data to be communicated at the Petabit level and beyond.

Marketing

Fast router technology has ample opportunities in the tech market because it addresses the need for fast and efficient network infrastructure. This section assesses the success of our router technology in the market by applying the 4P marketing analysis which considers four main aspects of go-to-market elements: price, product, promotion and place.

One can find routers being used in almost all digital systems where there are at least two endpoints that can communicate with each other. However, as a new entrant, it is important to find a specific niche market in which our product best fits. According to Andre Barroso, the manufacturing cost is directly proportional to the number of radix (Andre Barroso, 2013). The increased cost means that our product will be an enterprise, business-to-business product rather than a commodity sold directly to consumers. Moreover, companies such as Broadcom, Cisco and Juniper are already dominant in the networking world, thus making it a difficult process for us as new entrant to compete. As previously mentioned, our router technology is designed to enable fast and efficient communication between large collections of machines in computing centers. Therefore, it may be in our best interest to zoom in our market focus to companies such as Google and Facebook that house homegrown warehouse-scale datacenters. Moreover, in recent years many major players on par with Google and Facebook have starting to develop their own data servers, thus forming a growing pool of demand for robustness and efficiency in the underlying networking infrastructures.

Since our market segment is quite narrow and our product fits business-to-business commerce the most, our distribution channel should just be a team of professional salespeople that are highly familiar and experienced in this market. Therefore, the appropriate promotion strategy is definitely not huge-scale advertisement; rather, if our technology is exactly what Google or Facebook is looking for, their adoption of

our product will publicize it to other potential customers. Another common way for new techs to raise awareness is by showcasing them at technology trade shows such as Consumer Electronic Show. Linksys and Netgear – companies that sells data networking hardware products – for example have seen huge success in CES with announcements of new generation of routers.

In general, as we determined our product as a business-to-business one, we will first focus our market on big companies such as Google and Facebook who need router technology for their datacenters. As a new entrant, we will keep track on what our competitors are doing, and specialize in our feature – using high radix to achieve high speed. Once we succeed in our first target market, we plan to promote our product to a broader potential market to gain more recognition by publicizing the product through existing consumers and showcasing the product in Electronic Show.

Work Cited

1. D. Becker and William J. Dally."Allocator Implementations for Network-on-Chip Routers". Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009: 1-12. Portland, OR. 14-20 Nov. 2009

2. Armbrust, Michael, Armando Fox, Rean Griffith, Above the Clouds: A Berkeley View of Cloud Computing. Feb 2009. Accessed March 21.2016

3. Dally, William, and Towles, Brian. Principles and Practices of Interconnection Networks. San Francisco: Morgan Kaufmann Publishers, 2004.

4. Berkeley Lawrence Lab. OpenSoC Fabric. 2014. https://github.com/LBL-CoDEx/OpenSoCFabric/wiki, Accessed March 21, 2016.

5. Lo, Jen-Hung. "Technical Contribution – Petabit Switch Fabric Design" Capstone Project Report. Berkeley, CA. 2016

6. Cao, Yue. "Technical Contribution – Petabit Switch Fabric Design" Capstone Project Report. Berkeley, CA. 2016

7. Andre Barroso, Luiz, Jimmy Calidaras, Urs Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines.*" Morgan & Claypool Publishers, 2013.

8. Becker, Daniel. "Efficient microarchitecture for network-on-chip routers". Doctoral dissertation submitted to Stanford University. August 2012. http://purl.stanford.edu/wr368td5072

9. Benini, Luca and Giovanni De Micheli. "The Challenges of Next-gen Multicore Networks-on-Chip Systems." n.p. 26 Feb. 2007

< http://www.embedded.com/design/mcus-processors-and-socs/4006822/The-challenges-of-next-gen-multicore-networks-on-chip-systems-Part-4>

10. Broadcom. Broadcom Delivers Industry's First High-Density 25/100 Gigabit Ethernet Switch for Cloud-Scale Networks. Press Release. n.p., 24 Sept. 2014. Web. 1 Mar. 2015. <<u>http://www.broadcom.com/press/release.php?id=s872349></u>.

11. Dell. "Dell Private Cloud Solutions". www.dell.com. n.d. Accessed 4 March 2016

12. Hassan, Qusay. "Demystifying Cloud Computing". The Journal of Defense Software Engineering (CrossTalk) (2011 Jan/Feb): 16–21. Retrieved 4 March 2016

13. Porter. "*The Five Competitive Forces That Shape Strategy*". hbr.org. January 2008. Accessed 4 March 2016

14. Simpkins, Adam. "Facebook Open Switching System ("FBOSS") and Wedge in the Open." n.p. 10 Mar. 2014.

< https://code.facebook.com/posts/843620439027582/facebook-open-switching-system-fboss-and-wedge-in-the-open/>

15. Singh, Arjun, et al. "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network". In SIGCOMM (2015)

16. Vahdat, Amin. "A Look Inside Google's Data Center Network." Open Networking Summits 2015. Santa Clara Convention Center, Santa Clara. 17 Jun. 2015. Keynote.

Appendix A: Single Router Test Harness

SimpleVCRouterTester Module

```
1 package OpenSoC
3 import Chisel.
4 import scala.collection.mutable.HashMap
5 import scala.collection.mutable.LinkedHashMap
6 import scala.util.Random
7 import Array.__
8
9
10 class SimpleVCRouterTester(c: SimpleVCRouterTestWrapper) extends Tester(c) {
      implicit def bool2BigInt(b:Boolean) : BigInt = if (b) 1 else o
11
12
      val routerLatencyInClks = 6
13
      var headFlitMap_1 = LinkedHashMap [String, BigInt] ()
14
      var bodyFlitMap_1 = LinkedHashMap [String, BigInt] ()
15
       var headFlitMap 2 = LinkedHashMap [String, BigInt]()
16
      var bodyFlitMap 2 = LinkedHashMap [String, BigInt] ()
17
       var headFlitMap 3 = LinkedHashMap [String, BigInt] ()
18
       var headFlitMap 4 = LinkedHashMap [String, BigInt]()
19
       var headFlitMap_5 = LinkedHashMap [String, BigInt] ()
20
       var headFlitMap_6 = LinkedHashMap [String, BigInt] ()
21
       var headFlitMap_7 = LinkedHashMap [String, BigInt] ()
22
       var headFlitMap_8 = LinkedHashMap [String, BigInt] ()
23
       var bodyFlitMap_3 = LinkedHashMap [String, BigInt] ()
24
       var bodyFlitMap_4 = LinkedHashMap [String, BigInt] ()
25
       var bodyFlitMap_5 = LinkedHashMap [String, BigInt] ()
26
       var bodyFlitMap_6 = LinkedHashMap [String, BigInt] ()
27
       var bodyFlitMap_7 = LinkedHashMap [String, BigInt] ()
28
       var bodyFlitMap 8 = LinkedHashMap [String, BigInt] ()
29
30
31 // ---- First Packet Definition ----
       headFlitMap 1 = LinkedHashMap(
32
                           -> 0 ), //destination coordinates
           ("Dest"
33
           ("packtType"
34
                              -> 3),
           ("vcPort"
                           -> 0 ), //vc channel#
35
36
           ("isTail"
                           -> 0),
           ("packetID"
37
                             ->0)
     )
38
39
       bodyFlitMap_1 = LinkedHashMap(
40
41
           ("payload"
                            -> oxBEEF ),
           ("flitID"
42
                           -> oxC ),
           ("vcPort"
43
                           -> 0),
           ("isTail"
                           -> 0),
44
           ("packetID"
                             ->0)
45
46
       )
47
     // ---- Second Packet Definition ----
48
       headFlitMap_2 = LinkedHashMap(
49
           ("Dest"
                           -> 0), //destination coordinates
50
           ("packtType"
                             -> 2),
51
           ("vcPort"
                           -> 0 ), //vc channel#
52
```

```
("isTail"
                            -> 0),
53
            ("packetID"
                              ->1)
54
       )
55
56
       bodyFlitMap_2 = LinkedHashMap(
57
            ("payload"
                            -> oxDEAD ),
58
            ("flitID"
                            -> oxF ),
59
            ("vcPort"
60
                            -> 0),
61
            ("isTail"
                           -> 0),
            ("packetID"
                              ->1)
62
       )
63
64
65 // ---- Third Packet Definition ----
       headFlitMap_3 = LinkedHashMap(
66
            ("Dest"
                            -> 0), //destination coordinates
67
            ("packtType"
68
                               -> 1),
            ("vcPort"
                             -> 0 ), //vc channel#
69
            ("isTail"
                            -> 0),
70
           ("packetID"
                              ->2)
71
       )
72
73
       bodyFlitMap_3 = LinkedHashMap(
74
           ("payload"
                             -> oxABCD ),
75
   ("flitID"
                -> oxA ),
           ("vcPort"
                            -> 0),
77
            ("isTail"
78
                            -> 0),
            ("packetID"
                              -> 2)
79
       )
80
81
82 // ---- 4th Packet Definition ----
83
       headFlitMap_4 = LinkedHashMap(
            ("Dest"
                            -> 0), //destination coordinates
84
            ("packtType"
85
                               -> 4 ),
            ("vcPort"
86
                             -> 0 ), //vc channel#
87
            ("isTail"
                            -> 0),
            ("packetID"
88
                              ->3)
       )
89
90
       bodyFlitMap_4 = LinkedHashMap(
("payload" -> oxBACA ),
91
92
            ("flitID"
                            -> oxD ),
93
            ("vcPort"
                            -> 0 ),
94
            ("isTail"
                            -> 0),
95
            ("packetID"
96
                              ->3)
       )
97
98
99
100 // ---- 5th Packet Definition ----
        headFlitMap_1 = LinkedHashMap(
101
            ("Dest"
                            -> o), //destination coordinates
102
            ("packtType"
103
                               -> 3),
            ("vcPort"
                             -> 0 ), //vc channel#
104
            ("isTail"
                            -> 0),
105
            ("packetID"
106
                               ->4)
        )
107
108
        bodyFlitMap_1 = LinkedHashMap(
109
```

```
("payload"
                              -> oxBCCA ),
110
            ("flitID"
111
                            -> oxD ).
            ("vcPort"
                             -> 0),
112
            ("isTail"
                            -> 0),
113
            ("packetID"
                              ->4)
114
       )
115
116
117
118
119 // ---- 6th Packet Definition ----
        headFlitMap_1 = LinkedHashMap(
120
            ("Dest"
                            -> o), //destination coordinates
121
            ("packtType"
                               -> 2),
122
            ("vcPort"
                             -> 0 ), //vc channel#
123
            ("isTail"
124
                            -> 0),
125
            ("packetID"
                               -> 5)
        )
126
127
        bodyFlitMap_1 = LinkedHashMap(
128
            ("payload"
                              -> oxFFFF ),
129
            ("flitID"
                            \rightarrow oxB),
130
            ("vcPort"
                             -> 0),
131
            ("isTail"
                            -> 0),
132
            ("packetID"
133
                               ->5)
        )
134
135
136
137 // ---- 7th Packet Definition ----
        headFlitMap_1 = LinkedHashMap(
138
            ("Dest"
                             -> o), //destination coordinates
139
            ("packtType"
140
                                ->1),
141
            ("vcPort"
                             -> 0 ), //vc channel#
            ("isTail"
142
                            -> 0),
            ("packetID"
                               ->6)
143
        )
144
145
        bodyFlitMap_1 = LinkedHashMap(
146
            ("payload"
                              -> oxDDDD ),
147
148
            ("flitID"
                            -> oxD ),
            ("vcPort"
                             -> 0),
149
       ("isTail"
                       -> 0 ),
            ("packetID"
                              -> 6)
151
        )
152
153
154
155 // ---- 8th Packet Definition ----
        headFlitMap 1 = LinkedHashMap(
156
            ("Dest"
                            -> 0), //destination coordinates
157
            ("packtType"
                               -> 1),
158
            ("vcPort"
                             -> 0 ), //vc channel#
159
            ("isTail"
160
                             -> 0),
            ("packetID"
                               ->7)
161
        )
162
163
        bodyFlitMap_1 = LinkedHashMap(
164
            ("payload"
                              -> OXEEEE ),
165
            ("flitID"
                            -> oxE ),
166
```

```
167
            ("vcPort"
                            -> 0),
168
            ("isTail"
                            -> 0),
169
            ("packetID"
                              ->7)
        )
170
171
172
        poke(c.io.headFlitIn, headFlitMap 1.values.toArray)
173
        poke(c.io.bodyFlitIn, bodyFlitMap 1.values.toArray) //poke variables into module n
174
        step (1)
175
        var zeroFlit = peek(c.io.bodyFlitOut) // peek the varible from the module needs to
176
177
        for (i <- o until c.numInChannels) {
178
            poke(c.io.inChannels(i).flitValid, o)
179
            poke(c.io.inChannels(i).credit.grant, 0)
180
       }
181
182
     step(1)
183
184
185 //Instatutiate look up table
      //val nums = (o until c.numNodes).map(x => BigInt((x + 1) % c.numRadix))
186
     val nums = (0 until c.numNodes).map(x => BigInt( (x + 3) % c.numRadix ))
187
188
189
      for (i <- 0 until c.numNodes) {</pre>
        poke(c.io.lutWriteEnable, true)
190
       poke(c.io.lutWriteAddress, i)
191
        poke(c.io.lutWriteData, nums(i))
192
        step(1)
193
     }
194
     poke(c.io.lutWriteEnable, false)
195
196
197
        step(1)
        printf("------\n")
198
        printf("Drive single 2-flit packetn")
199
        // ---- Packet 1 ----
200
        headFlitMap 1("Dest")
201
                                  = 5
        headFlitMap_1("isTail") = 0
202
        headFlitMap_1("packetID") = 0
203
        bodyFlitMap_1("packetID") = 0
204
205
        bodyFlitMap_1("isTail") = 1
206
        poke(c.io.headFlitIn, headFlitMap 1.values.toArray)
207
        poke(c.io.bodyFlitIn, bodyFlitMap 1.values.toArray)
208
209
        step(1)
210
        var myHeadFlit_1 = peek(c.io.headFlitOut)
211
        var myBodyFlit_1 = peek(c.io.bodyFlitOut)
212
213
        step(1)
214
        for (i <- o until c.numInChannels) {</pre>
215
            poke(c.io.inChannels(i).flitValid, o)
216
            poke(c.io.outChannels(i).credit.grant, o)
217
        }
218
219
        poke(c.io.inChannels(4).flitValid, 1)
220
        poke(c.io.inChannels(4).flit, myHeadFlit_1)
221
222
        peek(c.io.lutReadData)
        peek(c.io.lutReadAddress)
223
```

```
224
        step(1)
225
        poke(c.io.inChannels(4).flitValid, 1)
226
        poke(c.io.inChannels(4).flit, myBodyFlit_1)
227
228
        peek(c.io.lutReadData)
        peek(c.io.lutReadAddress)
229
230
231
        step(1)
232
        poke(c.io.inChannels(4).flit, zeroFlit)
        poke(c.io.inChannels(4).flitValid, 0)
233
        peek(c.io.lutReadData)
234
        peek(c.io.lutReadAddress)
235
236
        step(routerLatencyInClks-5)
237
        expect(c.io.outChannels(7).flit, myHeadFlit 1)
238
        expect(c.io.outChannels(6).flit, myHeadFlit 1)
239
        expect(c.io.outChannels(5).flit, myHeadFlit_1)
240
        expect(c.io.outChannels(4).flit, myHeadFlit 1)
241
        expect(c.io.outChannels(3).flit, myHeadFlit 1)
242
        expect(c.io.outChannels(2).flit, myHeadFlit 1)
243
        expect(c.io.outChannels(1).flit, myHeadFlit 1)
244
        expect(c.io.outChannels(0).flit, myHeadFlit 1)
245
246
        step(1)
247
        expect(c.io.outChannels(7).flit, mvBodyFlit 1)
248
        expect(c.io.outChannels(6).flit, myBodyFlit_1)
249
        expect(c.io.outChannels(5).flit, myBodyFlit_1)
250
        expect(c.io.outChannels(4).flit, myBodyFlit 1)
251
        expect(c.io.outChannels(3).flit, myBodyFlit_1)
252
        expect(c.io.outChannels(2).flit, myBodyFlit_1)
253
        expect(c.io.outChannels(1).flit, mvBodyFlit 1)
254
255
        expect(c.io.outChannels(0).flit, myBodyFlit_1)
256
        printf ("------End Test 1 ------\n\n")
257
258
259
        printf("------ Test 1.5 ------ \n")
260
        printf("Drive single packet with longer flits\n")
261
        // ---- Packet 1 ----
262
263
        headFlitMap_1("Dest")
                                  = 5
        headFlitMap 1("isTail") = 0
264
        headFlitMap_1("packetID") = 0
265
266
        bodyFlitMap_1("packetID") = 0
        bodyFlitMap_1("isTail") = 0
267
268
        poke(c.io.headFlitIn, headFlitMap_1.values.toArray)
269
        poke(c.io.bodyFlitIn, bodyFlitMap 1.values.toArray)
270
271
272
        step(1)
        myHeadFlit_1 = peek(c.io.headFlitOut)
273
        mvBodvFlit 1 = peek(c.io.bodvFlitOut)
274
275
276
        step(1)
        bodyFlitMap 1("isTail") = 1
277
278
        poke(c.io.bodyFlitIn, bodyFlitMap 1.values.toArray)
279
        step(1)
280
```

```
281
        var my2ndBodyFlit 1 = peek(c.io.bodyFlitOut)
282
283
         step(1)
         for (i <- 0 until c.numInChannels) {
284
             poke(c.io.inChannels(i).flitValid, 0)
285
             poke(c.io.outChannels(i).credit.grant, o)
286
         }
287
288
289
         poke(c.io.inChannels(4).flitValid, 1)
         poke(c.io.inChannels(4).flit, myHeadFlit_1)
290
        peek(c.io.lutReadData)
291
292
         peek(c.io.lutReadAddress)
293
         step(1)
294
         poke(c.io.inChannels(4).flitValid, 1)
295
         poke(c.io.inChannels(4).flit, myBodyFlit 1)
296
         peek(c.io.lutReadData)
297
         peek(c.io.lutReadAddress)
298
299
300
         step(1)
         poke(c.io.inChannels(4).flitValid, 1)
301
         poke(c.io.inChannels(4).flit, my2ndBodyFlit 1)
302
         peek(c.io.lutReadData)
303
         peek(c.io.lutReadAddress)
304
305
306
         step(1)
         poke(c.io.inChannels(4).flit, zeroFlit)
307
         poke(c.io.inChannels(4).flitValid, 0)
308
         peek(c.io.lutReadData)
309
        peek(c.io.lutReadAddress)
310
311
312
        step(routerLatencyInClks-5)
        expect(c.io.outChannels(7).flit, myHeadFlit_1)
313
        expect(c.io.outChannels(6).flit, myHeadFlit 1)
314
        expect(c.io.outChannels(5).flit, myHeadFlit 1)
315
        expect(c.io.outChannels(4).flit, myHeadFlit 1)
316
        expect(c.io.outChannels(3).flit, myHeadFlit_1)
317
        expect(c.io.outChannels(2).flit, myHeadFlit 1)
318
319
        expect(c.io.outChannels(1).flit, mvHeadFlit 1)
320
         expect(c.io.outChannels(0).flit, myHeadFlit_1)
321
         step(1)
322
         expect(c.io.outChannels(7).flit, myBodyFlit_1)
323
         expect(c.io.outChannels(6).flit, myBodyFlit_1)
324
         expect(c.io.outChannels(5).flit, myBodyFlit_1)
325
326
         expect(c.io.outChannels(4).flit, myBodyFlit_1)
         expect(c.io.outChannels(3).flit, myBodyFlit 1)
327
        expect(c.io.outChannels(2).flit, mvBodvFlit 1)
328
         expect(c.io.outChannels(1).flit, myBodyFlit 1)
329
         expect(c.io.outChannels(0).flit, myBodyFlit 1)
330
331
332
         step(1)
         expect(c.io.outChannels(7).flit, my2ndBodyFlit 1)
333
         expect(c.io.outChannels(6).flit, my2ndBodyFlit 1)
334
         expect(c.io.outChannels(5).flit, my2ndBodyFlit 1)
335
         expect(c.io.outChannels(4).flit, my2ndBodyFlit_1)
336
         expect(c.io.outChannels(3).flit, my2ndBodyFlit_1)
337
```

```
338
        expect(c.io.outChannels(2).flit, my2ndBodyFlit 1)
        expect(c.io.outChannels(1).flit, my2ndBodyFlit 1)
339
        expect(c.io.outChannels(0).flit, my2ndBodyFlit_1)
340
        printf ("------End Test 1.5 ------\n\n")
341
342
        step(5)
343
        printf("------ Test 2 ------ \n")
344
        printf("Drive 2 packets of two flits each\n")
345
        // ---- Packet 1 ----
346
        headFlitMap_1("Dest")
                                  = 4
347
        headFlitMap_1("isTail")
348
                                    = 0
        headFlitMap_1("packetID") = 5
349
        bodyFlitMap_1("packetID") = 5
350
        bodyFlitMap_1("isTail")
                                    = 1
351
        // ---- Packet 2 ----
352
        headFlitMap_2("Dest")
                                  = 1
353
        headFlitMap_2("isTail")
                                    = 0
354
        headFlitMap 2("packetID") = 3
355
        bodyFlitMap_2("packetID") = 3
356
        bodyFlitMap 2("isTail")
                                    = 1
357
358
        poke(c.io.headFlitIn, headFlitMap 1.values.toArray)
359
360
        poke(c.io.bodyFlitIn, bodyFlitMap_1.values.toArray)
        //peek(c.io.headFlitIn)
361
        //peek(c.io.bodyFlitIn)
362
363
        step(1)
         myHeadFlit_1 = peek(c.io.headFlitOut)
364
         myBodyFlit_1 = peek(c.io.bodyFlitOut)
365
366
        poke(c.io.headFlitIn, headFlitMap_2.values.toArray)
367
368
        poke(c.io.bodyFlitIn, bodyFlitMap 2.values.toArray)
        //peek(c.io.headFlitIn)
369
370
        //peek(c.io.bodyFlitIn)
      step(1)
371
        var myHeadFlit 2 = peek(c.io.headFlitOut)
372
        var myBodyFlit 2 = peek(c.io.bodyFlitOut)
373
374
        headFlitMap_2("vcPort")
375
                                     = 0
376
        bodyFlitMap_2("vcPort")
                                     = 0
        poke(c.io.headFlitIn, headFlitMap_2.values.toArray)
377
        poke(c.io.bodyFlitIn, bodyFlitMap 2.values.toArray)
378
379
      step(1)
380
        var myHeadFlit_2_vcmod = peek(c.io.headFlitOut)
381
382
        var myBodyFlit_2_vcmod = peek(c.io.bodyFlitOut)
383
384
        step(1)
385
        for (i <- 0 until c.numInChannels) {
            poke(c.io.inChannels(i).flitValid, o)
386
            poke(c.io.outChannels(i).credit.grant, o)
387
        }
388
389
        poke(c.io.inChannels(4).flitValid, 1)
390
        poke(c.io.inChannels(4).flit, myHeadFlit 1)
391
392
      peek(c.io.lutReadData)
393
      peek(c.io.lutReadAddress)
394
```

395 396	step(1)
397 398 399	poke(c.io.inChannels(4).flit, myBodyFlit_1)
400 401	peek(c.io.lutReadData) peek(c.io.lutReadAddress)
402	step(1)
404 405 406	poke(c.io.inChannels(4).flitValid, 0)
407 408 400	peek(c.io.lutReadData) peek(c.io.lutReadAddress)
410 411	step(1) poke(c.io.inChannels(4).flitValid, 1)
412 413	poke(c.io.inChannels(4).flit, myHeadFlit_2)
414 415 416	peek(c.io.lutReadData) peek(c.io.lutReadAddress)
417 418	step(1)
419 420	poke(c.io.inChannels(4).flit, myBodyFlit_2)
421 422	peek(c.io.lutReadData) peek(c.io.lutReadAddress)
423 424	step(1)
425 426 427	poke(c.io.inChannels(4).flitValid, 0)
428	peek(c.io.lutReadData)
429	ston (nouton) ston out_(like =)
431 432	expect(c.io.outChannels(7).flit, myHeadFlit_1)
433 434	expect(c.io.outChannels(6).flit, myHeadFlit_1) expect(c.io.outChannels(5).flit, myHeadFlit_1)
435 436	expect(c.io.outChannels(4).flit, myHeadFlit_1) expect(c.io.outChannels(3).flit, myHeadFlit_1)
437	expect(c.io.outChannels(2).flit, myHeadFlit_1)
430	expect(c.io.outChannels(0).flit, myHeadFlit_1)
440 441	step(1)
442 443	expect(c.io.outChannels(7).flit, myBodyFlit_1) expect(c.io.outChannels(6).flit, myBodyFlit_1)
444 445	expect(c.io.outChannels(5).flit, myBodyFlit_1) expect(c.io.outChannels(4).flit, myBodyFlit_1)
446	expect(c.io.outChannels(3).flit, myBodyFlit_1)
447 448 449	expect(c.io.outChannels(2).flit, myBodyFlit_1) expect(c.io.outChannels(0).flit, myBodyFlit_1)
450 451	step(2)

```
expect(c.io.outChannels(7).flit, myHeadFlit 2 vcmod)
452
        expect(c.io.outChannels(6).flit, mvHeadFlit 2 vcmod)
453
        expect(c.io.outChannels(5).flit, myHeadFlit_2_vcmod)
454
        expect(c.io.outChannels(4).flit, myHeadFlit_2_vcmod)
455
        expect(c.io.outChannels(3).flit, myHeadFlit_2_vcmod)
456
        expect(c.io.outChannels(2).flit, myHeadFlit_2_vcmod)
457
        expect(c.io.outChannels(1).flit, myHeadFlit_2_vcmod)
458
459
        expect(c.io.outChannels(o).flit, myHeadFlit 2 vcmod)
460
461
      step(1)
        expect(c.io.outChannels(7).flit, myBodyFlit_2_vcmod)
462
        expect(c.io.outChannels(6).flit, myBodyFlit 2 vcmod)
463
        expect(c.io.outChannels(5).flit, myBodyFlit_2_vcmod)
464
        expect(c.io.outChannels(4).flit, myBodyFlit_2_vcmod)
465
        expect(c.io.outChannels(3).flit, myBodyFlit 2 vcmod)
466
        expect(c.io.outChannels(2).flit, myBodyFlit 2 vcmod)
467
        expect(c.io.outChannels(1).flit, myBodyFlit 2 vcmod)
468
        expect(c.io.outChannels(0).flit, myBodyFlit 2 vcmod)
469
470
        printf ("-----End Test 2 -----\n\n")
471
472
473
        step(5)
474
        printf("------ Test 2.5 ------\n")
475
        printf("Drive 8 packets of two flits each\n")
476
        // ---- Packet 1 ----
477
        headFlitMap_1("Dest")
478
                                 = 5
        headFlitMap_1("isTail")
479
                               = 0
        headFlitMap_1("packetID") = 1
480
        bodyFlitMap_1("packetID") = 1
481
        bodyFlitMap 1("isTail") = 1
482
483
        // ---- Packet 2 ----
484
        headFlitMap 2("Dest")
485
                                  = 4
        headFlitMap_2("isTail")
486
                                 = 0
487
        headFlitMap_2("packetID") = 2
488
        bodyFlitMap_2("packetID") = 2
        bodyFlitMap 2("isTail")
489
                                 = 1
490
        // ---- Packet 3 ----
491
        headFlitMap 1("Dest")
492
                                 = 3
        headFlitMap_1("isTail")
                                 = 0
493
        headFlitMap_1("packetID") = 3
494
        bodyFlitMap_1("packetID") = 3
495
        bodyFlitMap_1("isTail") = 1
496
        // ---- Packet 4 ----
497
        headFlitMap 2("Dest")
498
                                  = 3
        headFlitMap 2("isTail")
499
                                 = 0
        headFlitMap_2("packetID") = 4
500
        bodyFlitMap_2("packetID") = 4
501
        bodyFlitMap_2("isTail") = 1
502
503
        // ---- Packet 5 ----
504
        headFlitMap 1("Dest")
505
                                 = 4
        headFlitMap_1("isTail")
                                 = 0
506
        headFlitMap_1("packetID") = 5
507
        bodyFlitMap_1("packetID") = 5
508
```

```
509
        bodyFlitMap 1("isTail") = 1
        // ---- Packet 6 ----
510
       headFlitMap_2("Dest")
                                 = 2
511
        headFlitMap_2("isTail")
                                = 0
512
        headFlitMap_2("packetID") = 6
513
        bodyFlitMap_2("packetID") = 6
514
        bodyFlitMap_2("isTail") = 1
515
516
        // ---- Packet 7 ----
517
        headFlitMap_1("Dest")
518
                                  = 4
        headFlitMap_1("isTail")
519
                                  = 0
        headFlitMap_1("packetID") = 7
520
        bodyFlitMap_1("packetID") = 7
521
        bodyFlitMap_1("isTail") = 1
522
        // ---- Packet 8 ----
523
        headFlitMap_2("Dest")
                                  = 1
524
        headFlitMap_2("isTail")
                                  = 0
525
        headFlitMap 2("packetID") = 8
526
        bodyFlitMap_2("packetID") = 8
527
        bodyFlitMap 2("isTail")
528
                                 = 1
529
        poke(c.io.headFlitIn, headFlitMap_1.values.toArray)
530
        poke(c.io.bodyFlitIn, bodyFlitMap_1.values.toArray)
531
        step(1)
532
        myHeadFlit_1 = peek(c.io.headFlitOut)
533
        myBodyFlit_1 = peek(c.io.bodyFlitOut)
534
535
        poke(c.io.headFlitIn, headFlitMap 2.values.toArray)
536
        poke(c.io.bodyFlitIn, bodyFlitMap_2.values.toArray)
537
538
539
      step(1)
540
        mvHeadFlit 2 = peek(c.io.headFlitOut)
        myBodyFlit_2 = peek(c.io.bodyFlitOut)
541
542
        headFlitMap 2("vcPort")
543
                                     = 1
        bodyFlitMap_2("vcPort")
                                     = 1
544
        poke(c.io.headFlitIn, headFlitMap_2.values.toArray)
545
546
        poke(c.io.bodyFlitIn, bodyFlitMap_2.values.toArray)
547
548
      step(1)
        myHeadFlit 2 vcmod = peek(c.io.headFlitOut)
549
        myBodyFlit_2_vcmod = peek(c.io.bodyFlitOut)
550
551
        poke(c.io.headFlitIn, headFlitMap_3.values.toArray)
552
        poke(c.io.bodyFlitIn, bodyFlitMap_3.values.toArray)
553
554
      step(1)
555
        var myHeadFlit_3 = peek(c.io.headFlitOut)
556
        var myBodyFlit 3 = peek(c.io.bodyFlitOut)
557
558
        headFlitMap_3("vcPort")
                                     = 2
559
        bodyFlitMap_3("vcPort")
560
                                     = 2
        poke(c.io.headFlitIn, headFlitMap 3.values.toArray)
561
562
        poke(c.io.bodyFlitIn, bodyFlitMap 3.values.toArray)
563
      step(1)
564
        var myHeadFlit_3_vcmod = peek(c.io.headFlitOut)
565
```

```
566
        var myBodyFlit 3 vcmod = peek(c.io.bodyFlitOut)
567
        poke(c.io.headFlitIn, headFlitMap_4.values.toArray)
568
        poke(c.io.bodyFlitIn, bodyFlitMap_4.values.toArray)
569
570
571
      step(1)
        var myHeadFlit_4 = peek(c.io.headFlitOut)
572
        var myBodyFlit 4 = peek(c.io.bodyFlitOut)
573
574
        headFlitMap_4("vcPort")
                                     = 3
575
        bodyFlitMap_4("vcPort")
576
                                     = 3
        poke(c.io.headFlitIn, headFlitMap_4.values.toArray)
577
        poke(c.io.bodyFlitIn, bodyFlitMap_4.values.toArray)
578
579
580
      step(1)
        var myHeadFlit 4 vcmod = peek(c.io.headFlitOut)
581
        var myBodyFlit 4 vcmod = peek(c.io.bodyFlitOut)
582
583
584
        poke(c.io.headFlitIn, headFlitMap_5.values.toArray)
585
        poke(c.io.bodyFlitIn, bodyFlitMap 5.values.toArray)
586
587
588
      step(1)
        var myHeadFlit_5 = peek(c.io.headFlitOut)
589
        var myBodyFlit_5 = peek(c.io.bodyFlitOut)
590
        headFlitMap_5("vcPort")
591
                                     = 4
        bodyFlitMap_5("vcPort")
592
                                     = 4
        poke(c.io.headFlitIn, headFlitMap_5.values.toArray)
593
        poke(c.io.bodyFlitIn, bodyFlitMap_5.values.toArray)
594
595
596
597
      step(1)
598
        var myHeadFlit_5_vcmod = peek(c.io.headFlitOut)
        var myBodyFlit 5 vcmod = peek(c.io.bodyFlitOut)
599
600
601
        poke(c.io.headFlitIn, headFlitMap 6.values.toArray)
602
        poke(c.io.bodyFlitIn, bodyFlitMap 6.values.toArray)
603
604
      step(1)
605
        var myHeadFlit_6 = peek(c.io.headFlitOut)
        var myBodyFlit 6 = peek(c.io.bodyFlitOut)
606
607
        headFlitMap_6("vcPort")
                                      = 5
608
        bodyFlitMap_6("vcPort")
                                      = 5
        poke(c.io.headFlitIn, headFlitMap_6.values.toArray)
609
        poke(c.io.bodyFlitIn, bodyFlitMap_6.values.toArray)
610
611
612
613
      step(1)
        var myHeadFlit 6 vcmod = peek(c.io.headFlitOut)
614
        var myBodyFlit_6_vcmod = peek(c.io.bodyFlitOut)
615
616
        poke(c.io.headFlitIn, headFlitMap 7.values.toArray)
617
618
        poke(c.io.bodyFlitIn, bodyFlitMap_7.values.toArray)
619
620
      step(1)
        var myHeadFlit_7 = peek(c.io.headFlitOut)
621
        var myBodyFlit_7 = peek(c.io.bodyFlitOut)
622
```

```
623
        headFlitMap_7("vcPort")
                                      = 6
        bodyFlitMap_7("vcPort")
624
                                      = 6
        poke(c.io.headFlitIn, headFlitMap_7.values.toArray)
625
        poke(c.io.bodyFlitIn, bodyFlitMap_7.values.toArray)
626
627
628
629
      step(1)
         var myHeadFlit_7_vcmod = peek(c.io.headFlitOut)
630
        var myBodyFlit 7 vcmod = peek(c.io.bodyFlitOut)
631
632
         poke(c.io.headFlitIn, headFlitMap_8.values.toArray)
633
         poke(c.io.bodyFlitIn, bodyFlitMap_8.values.toArray)
634
635
      step(1)
636
        var myHeadFlit 8 = peek(c.io.headFlitOut)
637
638
         var myBodyFlit 8 = peek(c.io.bodyFlitOut)
         headFlitMap_8("vcPort")
639
                                      = 7
         bodyFlitMap 8("vcPort")
640
                                      = 7
        poke(c.io.headFlitIn, headFlitMap 8.values.toArray)
641
        poke(c.io.bodyFlitIn, bodyFlitMap 8.values.toArray)
642
643
      step(1)
644
645
        var myHeadFlit_8_vcmod = peek (c.io.headFlitOut)
         var myBodyFlit_8_vcmod = peek (c.io.bodyFlitOut)
646
647
648
      step(1)
         for (i <- 0 until c.numInChannels) {
649
             poke(c.io.inChannels(i).flitValid, o)
650
             poke(c.io.outChannels(i).credit.grant, o)
651
         }
652
653
         poke(c.io.inChannels(0).flitValid, 1)
654
        poke(c.io.inChannels(0).flit, myHeadFlit_1)
655
656
        peek(c.io.lutReadData)
657
658
         peek(c.io.lutReadAddress)
659
660
         step(1)
661
        poke(c.io.inChannels(0).flitValid, 1)
662
        poke(c.io.inChannels(0).flit, myBodyFlit_1)
663
         peek(c.io.lutReadData)
664
        peek(c.io.lutReadAddress)
665
666
667
        step(1)
         poke(c.io.inChannels(0).flit, zeroFlit)
668
         poke(c.io.inChannels(0).flitValid, 0)
669
670
        peek(c.io.lutReadData)
671
672
        peek(c.io.lutReadAddress)
673
        step(1)
674
        poke(c.io.inChannels(0).flitValid, 1)
675
676
        poke(c.io.inChannels(0).flit, myHeadFlit 2)
677
678
         peek(c.io.lutReadData)
        peek(c.io.lutReadAddress)
679
```

690	
000 601	stop(1)
690	step(1)
602	poke(c.io.inChannels(o).intvalid, 1)
083	poke(c.io.inchannels(0).int, myBodyFitt_2)
084	
685	peek(c.io.lutReadData)
686	peek(c.io.lutReadAddress)
687	
688	step(1)
689	poke(c.io.inChannels(0).flit, zeroFlit)
690	poke(c.io.inChannels(0).flitValid, 0)
691	
692	peek(c.io.lutReadData)
693	peek(c.io.lutReadAddress)
694	
695	step(1)
696	poke(c.io.inChannels(0).flitValid, 1)
697	poke(c.io.inChannels(0).flit, myHeadFlit_3)
698	
699	peek(c.io.lutReadData)
700	peek(c.io.lutReadAddress)
701	
702	step(1)
703	poke(c.io.inChannels(0).flitValid, 1)
704	poke(c.io.inChannels(0).flit, myBodyFlit_3)
705	
706	peek(c.io.lutReadData)
707	peek(c.io.lutReadAddress)
708	
, 709	step(1)
710	poke(c.io.inChannels(0).flit, zeroFlit)
711	poke(c.io.inChannels(0).flitValid, 0)
, 712	r (, , , , , , , , , , , , , , , , , ,
, 713	peek(c.io.lutReadData)
714	peek(c.io.lutReadAddress)
715	I to contract the total of total of the total of total of the total of the total of
716	step(1)
717	poke(c.io.inChannels(0).flitValid. 1)
718	$poke(c io in Channels(0) flit, mvHeadFlit_{\lambda})$
710	pone(chomionalinois(o),int, myrroadi ne_4)
720	peek(c io lutReadData)
721	peek(c io lutReadAddress)
722	peer(enonumental address)
722	sten(1)
724	noke(c io inChannels(0) flitValid 1)
/44 795	poke(c io in Channels(0) flit myBodyFlit 4)
726	poke(e.io.inenamicis(o).int, mybodyr nt_4)
720	neek(c io lutReadData)
/2/	pock(c.io.lutReadAddross)
720	pur
/49	stop(1)
/30	sicp(1) poko(a jo in Channols(0) flit zoroElit)
/31	poke(c.io.inChannels(c).iiit, zeroriit)
/32	poke(c.io.inchannels(0).intvalid, 0)
/33	peak(a is lut Pead Data)
734	peek(c.io.lutReadData)
/35	peek(c.io.iutkeadAddress)
736	

737	step(1)
738	poke(c.io.inChannels(0).flitValid, 1)
739	poke(c.io.inChannels(0).flit, myHeadFlit_5)
740	
741	peek(c.io.lutReadData)
742	peek(c.10.10tReadAddress)
743	stop(1)
744	step(1)
745	poke(c.io.inChannels(o).intvalid, 1)
740	poke(c.io.inchannels(o).int, myBodyFit_5)
747	pook(a jo lutPoodData)
/40	peck(c.io.iutReadData)
/49	peek(c.io.iutkeauAuuress)
/50	stop(1)
/51	step(1) poko(a jo inChannols(0) flit zoroElit)
/52	poke(c.io.inChannels(0).int, zeroritt)
/53	
/54 755	neek(c io lutReadData)
/00	pock(c.io.lutReadAddross)
/50	peek(c.io.iutkeauAuuress)
/5/	sten(1)
750	noke(c io in Channels(0) flitValid 1)
759	poke(c.io.inChannels(0).intvand, 1)
761	poke(e.io.inenanieis(o).int, inyrreaurin_o)
762	peek(c io lutReadData)
762	peek(c io lutReadAddress)
764	peex(e.io.iui.reauriauress)
765	sten(1)
766	poke(c io inChannels(0) flitValid 1)
767	poke(c io in Channels(0) flit myBodyFlit 6)
768	pone(enomienamions(o), mi, my body i m_o)
769	peek(c.io.lutReadData)
770	peek(c.io.lutReadAddress)
771	poon(ononanicalinations)
772	step(1)
773	poke(c.io.inChannels(0).flit, zeroFlit)
774	poke(c.io.inChannels(0).flitValid, 0)
775	
776	peek(c.io.lutReadData)
777	peek(c.io.lutReadAddress)
778	
779	
780	step(1)
781	poke(c.io.inChannels(0).flitValid, 1)
782	poke(c.io.inChannels(0).flit, myHeadFlit_7)
783	-
784	peek(c.io.lutReadData)
785	peek(c.io.lutReadAddress)
786	
787	step(1)
788	poke(c.io.inChannels(0).flitValid, 1)
789	poke(c.io.inChannels(0).flit, myBodyFlit_7)
790	
791	peek(c.io.lutReadData)
792	peek(c.io.lutReadAddress)
793	

794	step(1)
795	poke(c.io.inChannels(0).flit, zeroFlit)
796	poke(c.io.inChannels(0).flitValid, 0)
797	
798	peek(c.io.lutReadData)
799	peek(c.io.lutReadAddress)
800	
801	
802	step(1)
803	poke(c.io.inChannels(0).flitValid, 1)
804	poke(c.io.inChannels(0).flit, myHeadFlit_8)
805	
806	peek(c.io.lutReadData)
807	peek(c.io.lutReadAddress)
808	-
809	step(1)
810	poke(c.io.inChannels(0).flitValid, 1)
811	poke(c.io.inChannels(0).flit, myBodyFlit 8)
812	
813	peek(c.io.lutReadData)
814	peek(c.jo.lutReadAddress)
815	F(
816	sten(1)
817	poke(c.io.inChannels(o).flit.zeroFlit)
818	poke(c.io.inChannels(0).flitValid, 0)
810	pono(ononinenamois(o)interana, o)
820	peek(c io lutReadData)
821	peek(c io lutReadAddress)
822	peex(cionumentumess)
822	sten(router1 atencyInClks-5)
824	expect(c io outChannels(7) flit_myHeadFlit_1)
825	expect(c io outChannels(6) flit myHeadFlit 1)
826	expect(c io outChannels(5) flit myHeadFlit 1)
827	expect(c io outChannels(4) flit myHeadFlit 1)
828	expect(c io outChannels(2) flit myHeadFlit 1)
820	expect(c io outChannels(2) flit myHeadFlit 1)
820	expect(c io outChannels(1) flit myHeadFlit 1)
030 801	expect(c.io.outChannels(0) flit myHeadFlit 1)
031 800	expect(c.io.outchannels(0).int, inyffeaufit_1)
032 800	stop(1)
033 894	ovpost(s is outChannals(7) flit myBodyElit 1)
895 895	expect(c.io.outChannels(6) flit_myBodyFlit_1)
035 896	expect(c.io.outChannels(c).flit, myBodyFlit_1)
030 90 7	expect(c.io.outChannels(3).iiit, iiiyDodyFiit_1)
03/	expect(c.io.outChannels(a) flit myBodyFlit_1)
030	expect(c.io.outChannels(3).int, inybodyFit_1)
840	expect(c.io.outChannels(1) fit myPodyFit_1)
04U	expect(c.io.outChannels(1).iiit, iiiyDodyFiit_1)
041 940	expect(c.io.outChannels(0).int, inybodyFit_1)
042 940	stop(a)
043	step(2)
044 845	expect(c.io.outChannels(/).iiit, iiiyHeadFlit_2_VCMOd)
045	expect(c.io.outChannels(c).iiit, iiiyHeadFiit_2_VCMOd)
040	expect(c.io.outChannels(5).III, IIIyHeadFill_2_VCMOd)
04 ⁷ /	expect(c.io.outChannels(a).flit myHeadFlit a yeared)
040 840	expect(c.io.outChannels(3).iiit, iiiyHeadFlit_2_VCM0d)
049 850	expect(0.10.000001011111115(2).1110, 1119HeadFill_2_VCM00)
050	expect(c.io.outchannels(1).int, inyHeadFift_2_vcmod)

851	expect(c.io.outChannels(0).flit, myHeadFlit_2_vcmod)
step(1)	
854	expect(c.io.outChannels(7).flit, myBodyFlit_2_vcmod)
855	expect(c.io.outChannels(6).flit, myBodyFlit_2_vcmod)
856	expect(c.io.outChannels(5).flit, myBodyFlit_2_vcmod)
857	expect(c.io.outChannels(4).flit, myBodyFlit_2_vcmod)
858	expect(c.io.outChannels(3).flit, myBodyFlit_2_vcmod)
859	expect(c.io.outChannels(2).flit, myBodyFlit_2_vcmod)
860	expect(c.io.outChannels(1).flit, myBodyFlit_2_vcmod)
861	expect(c.io.outChannels(0).flit, myBodyFlit 2 vcmod)
862	
863	
864	step(2)
865	expect(c.io.outChannels(7).flit, myHeadFlit 3 vcmod)
866	expect(c.io.outChannels(6).flit, mvHeadFlit_3_vcmod)
867	expect(c.io.outChannels(5).flit, mvHeadFlit_3_vcmod)
868	expect(c.io.outChannels(4).flit, mvHeadFlit_3_vcmod)
869	expect(c.io.outChannels(3).flit, myHeadFlit_3_vcmod)
870	expect(c.io.outChannels(2).flit, myHeadFlit_3_vcmod)
871	expect(c io outChannels(1) flit, myHeadFlit 3 ycmod)
872	expect(c io outChannels(o) flit myHeadFlit 2 vcmod)
872	expect(c.io.outonumeis(o).int, myricuut nt_j_teniou)
874	sten(1)
875	expect(c io outChannels(7) flit_myBodyFlit_2_vemod)
876	expect(c io outChannels(6) flit_myBodyFlit_2_vcmod)
877	expect(c io outChannels(5) flit_myBodyFlit_2_vemod)
878	expect(c io outChannels(4) flit myBodyFlit_2 vemod)
870	expect(c io outChannels(2) flit myBodyFlit 2 vemod)
880	avport(c.io.outChannels(3).int, inyBodyFit_3_vemod)
000 QQ1	expect(c.io.outChannels(1) flit_myBodyFlit_3_vclifod)
001	expect(c.io.outChannels(1).int, myBodyFlit_3_vcmod)
002	expect(c.io.outchannels(0).int, mybodyFitt_3_vcmod)
003 001	
004 00-	$\operatorname{stop}(\mathbf{a})$
005 004	supp(2)
000	expect(c.io.outChannels(/).iiit, iiiyHeadFlit_4_vciiiod)
887	expect(c.io.outChannels(5).iiit, myHeadFiit_4_vcinod)
000	expect(c.io.outChannels(5).int, inyHeadFitt_4_vcinou)
889	expect(c.io.outChannels(4).filt, myHeadFilt_4_vcmod)
890	expect(c.io.outChannels(3).iiit, myHeadFlit_4_vcmod)
891	expect(c.io.outChannels(2).int, myHeadFit_4_vcmod)
892	expect(c.io.outChannels(1).iiit, myHeadFiit_4_vcmod)
893	expect(c.io.outChannels(0).filt, myHeadFilt_4_vcmod)
894	
895	step(1)
896	expect(c.io.outChannels(7).flit, myBodyFlit_4_vcmod)
897	expect(c.io.outChannels(6).flit, myBodyFlit_4_vcmod)
898	expect(c.io.outChannels(5).flit, myBodyFlit_4_vcmod)
899	expect(c.io.outChannels(4).flit, myBodyFlit_4_vcmod)
900	expect(c.io.outChannels(3).flit, myBodyFlit_4_vcmod)
901	expect(c.io.outChannels(2).flit, myBodyFlit_4_vcmod)
902	expect(c.io.outChannels(1).flit, myBodyFlit_4_vcmod)
903	expect(c.io.outChannels(0).flit, myBodyFlit_4_vcmod)
904	
905	step(2)
906	expect(c.io.outChannels(7).flit, myHeadFlit_5_vcmod)
907	expect(c.io.outChannels(6).flit, myHeadFlit_5_vcmod)
908	expect(c.io.outChannels(5).flit, myHeadFlit_5_vcmod)

909	expect(c.io.outChannels(4).flit, myHeadFlit_5_vcmod)
910	expect(c.io.outChannels(3).flit, myHeadFlit_5_vcmod)
911	expect(c.io.outChannels(2).flit, myHeadFlit_5_vcmod)
912	expect(c.io.outChannels(1).flit, myHeadFlit_5_vcmod)
913	expect(c.io.outChannels(0).flit, myHeadFlit_5_vcmod)
914	
915	step(1)
916	expect(c.io.outChannels(7).flit, myBodyFlit_5_vcmod)
917	expect(c.io.outChannels(6).flit, myBodyFlit_5_vcmod)
918	expect(c.io.outChannels(5).flit, myBodyFlit_5_vcmod)
919	expect(c.io.outChannels(4).flit, myBodyFlit 5 vcmod)
920	expect(c.io.outChannels(3).flit, myBodyFlit 5 vcmod)
921	expect(c.io.outChannels(2).flit, myBodyFlit 5 vcmod)
922	expect(c.io.outChannels(1).flit, myBodyFlit 5 vcmod)
923	expect(c.io.outChannels(0).flit, mvBodyFlit 5 vcmod)
924	
925	step(2)
926	expect(c.io.outChannels(7).flit. mvHeadFlit 6 vcmod)
927	expect(c.jo.outChannels(6),flit, myHeadFlit_6_vcmod)
028	expect(c io outChannels(5) flit myHeadFlit 6 vcmod)
020	expect(c io outChannels(4) flit, myHeadFlit, 6, ycmod)
030	expect(c io outChannels(3) flit_myHeadFlit_6_vcmod)
930	expect(c io outChannels(2) flit myHeadFlit 6 ycmod)
931 022	expect(c io outChannels(1) flit myHeadFlit 6 ycmod)
932	expect(c io outChannels(0) flit myHeadFlit 6 ycmod)
933	expect(c.io.outonanneis(0).int; injiteaut in_0_veniou)
934	sten(1)
935	expect(c io outChannels(7) flit_myBodyFlit_6_ycmod)
930	expect(c.io.outChannels(/).int, myBodyFit_0_vemod)
93/	expect(c.io.outChannels(c).int, myBodyFit_6_vemod)
930	expect(c.io.outChannels(3).int, myBodyFlit_6_vemod)
939	expect(c.io.outChannels(2).flit, myBodyFlit_6_vemod)
940	expect(c.io.outChannels(2).int, inyBodyFit_0_vemod)
941	avpact(a io outChannels(1) flit myBodyFlit_6_vemod)
942	expect(c.io.outChannels(1).int, inyBodyFit_0_vcinod)
943	expect(c.io.outchannels(0).int, inybodyFit_0_vcinod)
944	stan(a)
945	step(2)
940	expect(c.io.outChannels(/).iiit, myHeadFlitvcinod)
94/	expect(c.io.outChannels(c).int, myHeadFlit_7_vcmod)
940	expect(c.io.outChannels(5).int, invitedurit_/_vcinou)
949	expect(c.io.outChannels(4).iiit, iiiyHeadFlit_7_vciiiod)
950	expect(c.io.outChannels(3).iiit, myHeadFlitvcinou)
951	expect(c.io.outChannels(2).int, inyHeadFlitvcinou)
952	expect(c.io.outChannels(1).int, invHeadFitvcinod)
953	expect(c.io.outchannels(0).int, myneaurit_/_vcmou)
954	stop(1)
955	step(1)
956	expect(c.io.outChannels(7).fiit, myBodyFlit_7_vcmod)
957	expect(c.io.outChannels(b).filt, myBodyFilt_7_Vcmod)
958	expect(c.io.outChannels(5).filt, myBodyFilt_7_vcmod)
959	expect(c.io.outChannels(4).filt, myBodyFilt_7_vcmod)
960	expect(c.io.outUnanneis(3).IIIt, myBodyFilt_7_vcmod)
961	expect(c.io.outChannels(2).flit, myBodyFlit_7_vcmod)
962	expect(c.io.outChannels(1).flit, myBodyFlit_7_vcmod)
963	expect(c.io.outChannels(0).flit, myBodyFlit_7_vcmod)
964	
965	

966	
967	step(2)
968	expect(c.io.outChannels(7).flit, myHeadFlit_8_vcmod)
969	expect(c.io.outChannels(6).flit, myHeadFlit_8_vcmod)
970	expect(c.io.outChannels(5).flit, myHeadFlit_8_vcmod)
971	expect(c.io.outChannels(4).flit, myHeadFlit_8_vcmod)
972	expect(c.io.outChannels(3).flit, myHeadFlit_8_vcmod)
973	expect(c.io.outChannels(2).flit, myHeadFlit_8_vcmod)
974	expect(c.io.outChannels(1).flit, myHeadFlit_8_vcmod)
975	expect(c.io.outChannels(0).flit, myHeadFlit_8_vcmod)
976	
977	step(1)
978	expect(c.io.outChannels(7).flit, myBodyFlit_8_vcmod)
979	expect(c.io.outChannels(6).flit, myBodyFlit_8_vcmod)
980	expect(c.io.outChannels(5).flit, myBodyFlit_8_vcmod)
981	expect(c.io.outChannels(4).flit, myBodyFlit_8_vcmod)
982	expect(c.io.outChannels(3).flit, myBodyFlit_8_vcmod)
983	expect(c.io.outChannels(2).flit, myBodyFlit_8_vcmod)
984	expect(c.io.outChannels(1).flit, myBodyFlit_8_vcmod)
985	expect(c.io.outChannels(0).flit, myBodyFlit_8_vcmod)
986	
987	printf ("\n\n")

Modification in main.scala:

case "SimpleVCRouterTester" => (chiselMainTest(myargs, moduleToTest) { c => new SimpleVCRouterTester(c.asInstanceOf[SimpleVCRouterTestWrapper]) }