Energy Efficient Links and Routers for Multi-Processor Computer Systems

by

Imran Shamim

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Science

in Electrical Engineering and Computer Science

September 2009

Department of Electrical Engineering and Computer Science

Cambridge, MA

Author........[illegible]........
Department of Electrical Engineering and Computer Science
August 18, 2009

Certified by........[illegible]........
Vladimir Stojanović
Assistant Professor of Electrical Engineering
MIT Thesis Supervisor

Accepted by........[illegible]........
Terry P. Orlando
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Energy Efficient Links and Routers for Multi-Processor Computer Systems

by

Imran Shamim

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science,
at the Massachusetts Institute of Technology
September 2009

Abstract

As multi-processor computer systems become more prevalent in today's computer industry, it is clear that routers and interconnection networks are critical components of these multi-processor systems. Therefore, there is a need to obtain accurate area and power models for these critical components so that we can better understand the area and power tradeoffs as we balance the on-chip and off-chip communication energy given a fixed energy budget.

In this thesis, we propose an alternative method to understanding the power and area tradeoffs for routers by not solely relying on analytical models, on which most current studies done on this topic are based. Instead, in this thesis we propose analyzing the area versus power tradeoff for these routers and interconnection networks using an Application Specific Integrated Circuit (ASIC) flow in a commercially available IBM 90nm process technology.

This thesis shows that multiplexer routers are more area and power efficient compared to matrix routers since matrix routers quickly exhibit a quadratic-like increase in area and power as the number of ports and port width increases. In addition, we show that there is a real gain in area when the router is shared among 4 or more cores. The savings by sharing the same router among multiple cores does not continue indefinitely, since after a certain port number and port width size, the increase in the crossbar size can no longer be compensated by sharing the router. So for a fixed port-width, there is always a sweet spot for the number of ports where a local minimum can be found for the Router Area Overheard per Core. By examining and analyzing the router design space, we show that for maximum area and power efficiency, it is much better to use a multiplexer router with 8-ports as opposed to matrix routers. Moreover, keeping the flit size to 32-bits or 64-bits results in a larger Router Area Overheard per Core savings as opposed to using a flit size of 128-bits. Even in situations where the core manipulates 128-bits of data, using two 64-bit routers running in parallel at the core frequency will result a larger area and power savings.

Most importantly, we show that by looking at the costs and benefits of aggregation, we see that aggregation is only useful for narrow channel routers. This is because successful aggregation is a function of the crossbar complexity versus the bit/port width. By switching from a NoC with 5-port routers to higher radix routers while keeping the network bisection bandwidth approximately constant, the savings in the Router Area Overheard per Core can be up to 63%.

The results of this work will allow us to calibrate our existing Orion 1.0 analytical models. We show that the source of the largest discrepancy between the synthesized results and the analytical models is the buffer and not the crossbar as we expected. The crossbar can be further optimized by designing and physically laying it out manually. The buffers were obtained using the Artisan SRAM Register File Memory Generator software, and hence they are expected to be fully optimized for power and area efficiency. Therefore, it appears that our analytical models for the buffer might not be accurate. In addition, the results of this study will be used to narrow down the microarchitecture of a router to be used in the Integrated Photonics Network project at the Integrated Systems Group (ISG) at the Research Laboratory for Electronics.

Thesis Supervisor: Vladimir Stojanović
Title: Assistant Professor of Electrical Engineering

Acknowledgement

I would like to gratefully acknowledge the contribution of my advisor, Professor Vladimir Stojanović, for his constant mentorship, support, and oversight of this research project. I can certainly affirm that I learned a lot more by working with Professor Stojanović than taking any course, or teaching any course at MIT. His strong persistence in helping me solve various tool and research issues brought this project to the point of completion. Professor Stojanović's diversity, breadth of knowledge, and understanding has been a true source of inspiration. I can never thank Prof. Stojanović enough for giving me the opportunity to participate in his research group.

I would also like to thank our post-doc Ajay Joshi for his assistance and advice throughout the project. In particular, I would like to thank Ajay for providing me with the Orion 1.0 1.0 Analytical Power estimates for the router components. A special thanks to Fred Chen for helping me understand and solve various tool issues I encountered during this project. Ajay and Fred's knowledge of tools has been an inspiration and encouragement for me. I would also like to thank everyone else at the Integrated Systems Group: Wei, Mike, Byungsub, Ben, Sanquan, Ranko, Jonathan, Hossein and Yan, along with our summer interns, Thanh, Sergio, and Ankit, for providing a friendly and fun environment to work in.

Finally, I would like to especially thank my wife, my parents, and my family for supporting me and helping me bear with the ups and downs of life.

Contents

List of Figures

List of Tables

1. INTRODUCTION AND MOTIVATION

The advantages of technology scaling have been diminishing since even though CMOS gates in single-processor systems are getting smaller, higher clock frequencies and larger wire resistances have dramatically increased their power dissipation. Therefore, the computer hardware industry has shifted its focus to parallelism, using multi-core processors. Technology scaling has made it possible to integrate a large number of processor cores onto a single chip. In addition, chip bandwidth has been continuously increasing over the past two decades. Traditionally, most implementations increase the on-chip and off-chip bandwidth by simply increasing the bandwidth of the router ports and the mesh interconnection network links [1]. However, as the number of processor cores increases, increasing the bandwidth of the router ports and interconnection links makes them critical components of the system, which affects not only performance but also the power consumption and area overhead of the entire network [2].

Since routers and interconnection networks are crucial components in a multi-processor computer system, there is a need to obtain accurate area and power models for these components that are not based solely on analytical models or numerical simulations. Moreover, since energy consumption is a key metric in today's computer hardware industry, it is important to understand the different tradeoffs that result when we balance the energy of on-chip and off-chip communication for a given energy budget. A better understanding of the area and power tradeoff will enable the design of more energy and area efficient router architectures, as well as interconnection networks. In this thesis, we propose an analysis for the power and area tradeoff for routers and interconnection links by varying the router microarchitecture, the router ports, and the interconnection link widths. Instead of using analytical models for different components, we synthesize different router designs using a commercially available IBM 90nm process technology, and use these synthesized results to obtain area and power consumption values. The results we obtain in this thesis will be used not only to develop better energy/area models and calibrate existing analytical models, but also to better understand the bottlenecks and suggest directions for improvement of router microarchitecture.

1.2. PREVIOUS WORK

The performance of multi-processor computer systems is highly dependent on the interconnection network that is used to connect processors, router switches, memories, and the I/O devices. As the number of processor cores increases, the number of wires that makes up the interconnection network becomes a significant fraction of the entire chip area [3]. More wires imply higher bandwidth, therefore we have seen a steady increase of an order magnitude every five years in the bandwidth per router port [1].

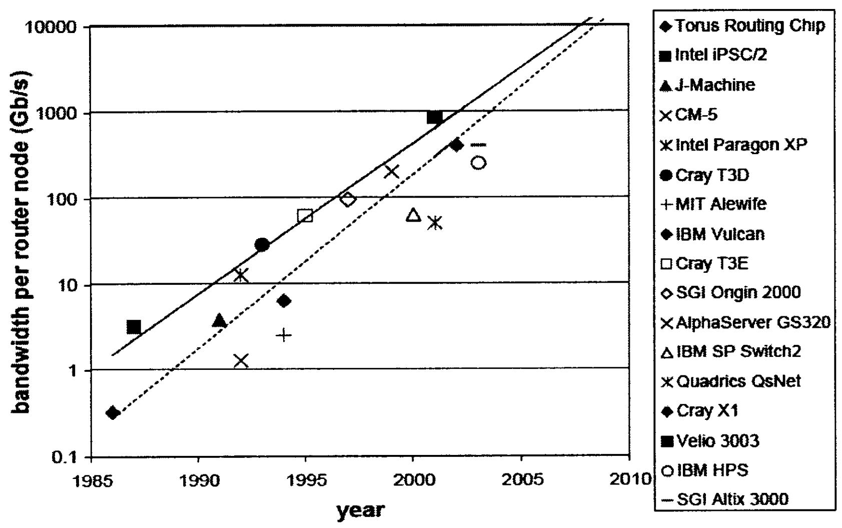

Figure 1: Bandwidth per router port plotted versus time. The dotted line represents the curve fit, while the solid line depicts the best performance router for each period [1].

Figure 1 depicts the increase in bandwidth per router port as a function of time, and it is observed that the bandwidth per port has increased by several magnitudes over the past two decades. The steady increase in bandwidth is a result of not only an increase in the number of wires, but also due to faster clocking speeds. Also shown in Figure 1 is a dotted line representing the curve fit, and a solid line that represents the best performance router for each technology period.

Most implementations exploit the increase in bandwidth per router port simply by increasing the width of the router links and the interconnection links. Recently, it has been suggested that taking advantage of the increase in bandwidth by increasing the number of router ports not only reduces the latency by reducing the hop-count, but it also results in a lower area cost [1]. However, previous studies done on multi-port (high radix) routers and link widths rely heavily on analytical models and numerical simulations. Moreover, these studies look at the router microarchitecture primarily from a performance perspective by proposing different circuit, architectural, and algorithmic techniques to reduce latency and increase throughput [1]. The results of these numerical simulations are only as accurate as the analytical models used for the router components and technology parasitics. Recently, it has been shown that these analytical models can be off from reality by an order of magnitude or even more [7, 8].

In addition, these studies give very little information on the router's power consumption and its area overhead as a function of the number of router ports, as well as the right cycle-time to properly balance the static and dynamic power consumption in the router network, for best energy-efficiency.

1.3. ASIC DESIGN METHODOLOGY

Since power consumption and circuit footprint are key metrics in today's computer hardware industry, in this project we evaluate different router architecture designs by using a real commercially available IBM process. Therefore, instead of relying solely on analytical models and numerical network simulations, we setup an ASIC (Application Specific Integrated Circuits) toolflow to synthesize various router designs using the IBM CMOS9SF 90nm process technology. We then analyze the tradeoff between area and power consumption for synthesized routers and use these results to narrow down the router microarchitecture design space.

Our ASIC design flow starts with a Verilog file that contains the behavioral description of the router modules. The Verilog file is then simulated using Modelsim to ensure correct functionality of the design. Next, the Verilog file is synthesized into a gate level netlist using Cadence RTL Compiler. The synthesized design is then floor-planned and placed-and-routed using Cadence Encounter to generate the physical layout of the design. The physical layout gives us an accurate area number for the chip. Finally, the physical layout file is imported into Cadence Virtuoso, and converted into an HSPICE netlist containing a description of the entire circuit and device parasitic. The HPSICE netlist is then simulated in Synopsys Nanosim to verify the logical functionality of the synthesized, placed-and-routed design, and to give a power consumption estimate for the chip. Our entire ASIC design flow is presented in a flowchart format in Figure 2.

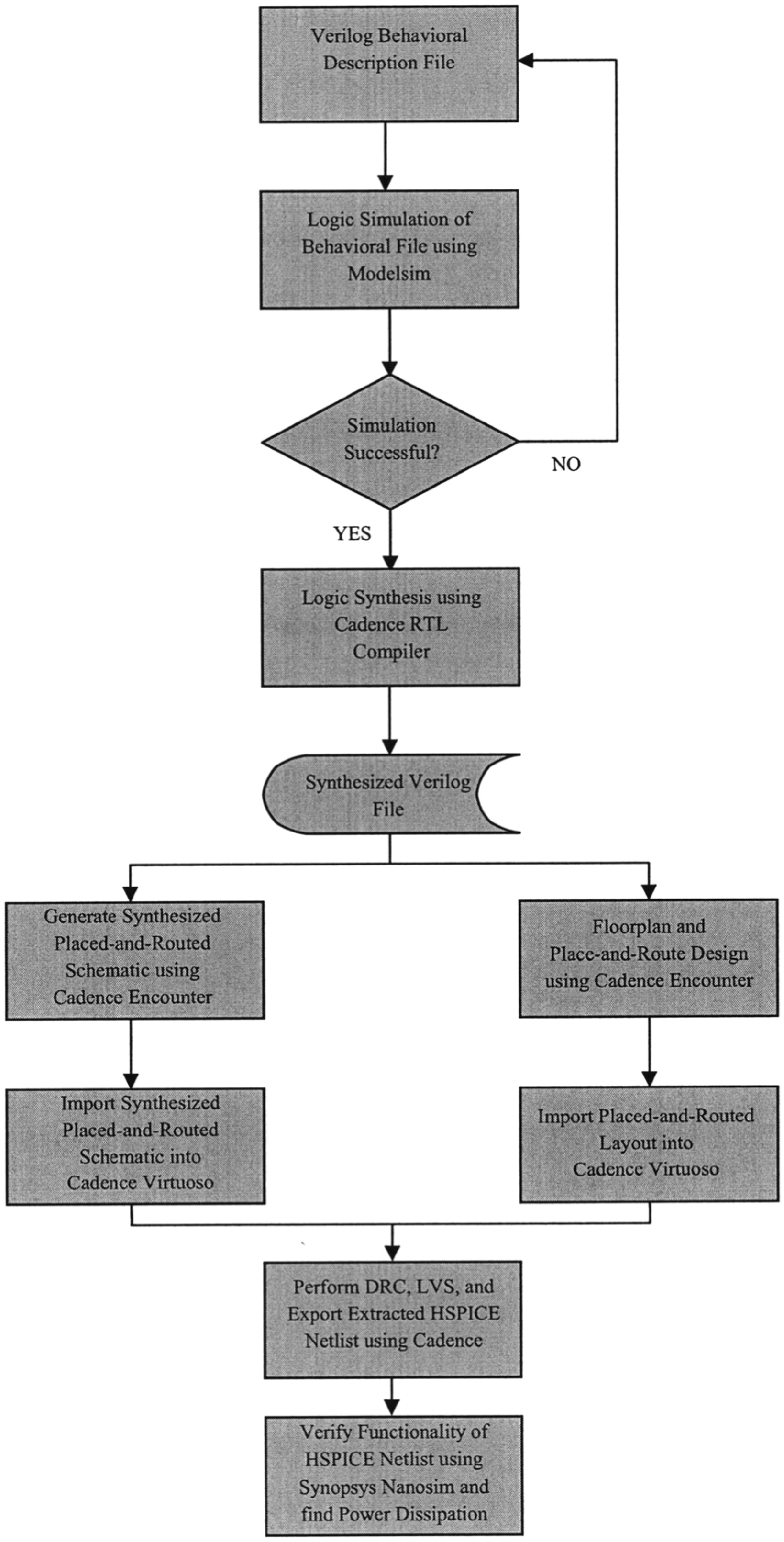
Verilog Behavioral Description File
Logic Simulation of Behavioral File using Modelsim
Simulation Successful?
NO
YES
Logic Synthesis using Cadence RTL Compiler
Synthesized Verilog File
Generate Synthesized Placed-and-Routed Schematic using Cadence Encounter
Floorplan and Place-and-Route Design using Cadence Encounter
Import Synthesized Placed-and-Routed Schematic into Cadence Virtuoso
Import Placed-and-Routed Layout into Cadence Virtuoso
Perform DRC, LVS, and Export Extracted HSPICE Netlist using Cadence
Verify Functionality of HSPICE Netlist using Synopsys Nanosim and find Power Dissipation

Figure 2: ASIC design flowchart.

1.4. ROUTER BASICS

Advanced fabrication technologies and scaling have made it possible to integrate a large number of processor cores onto a single die, allowing us to obtain an entire Network-on-Chip (NoC). An example of a NoC with DRAM memory controllers and Dual Inline Memory Modules (DIMM) is shown in Figure 3, where a system consisting of 16 cores is shown [4, 5]. The trend to increase the number of processor cores on a die is expected to continue and reach up to 256 cores in 22nm technology [4]. Data is switched and transferred in these many-core systems using routers, and the microarchitecture of a typical router is shown in Figure 3.

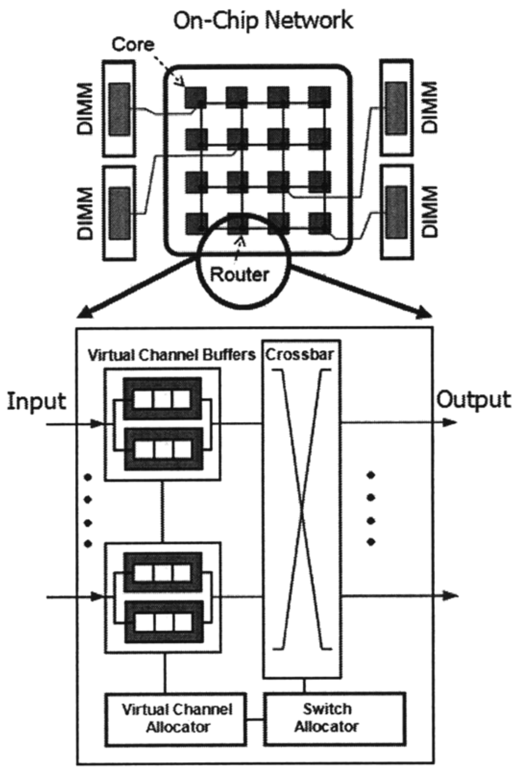

Figure 3: An example of a Network-on-Chip with routers to switch and transfer data. A zoom-in on the router reveals its microarchitecture [4, 5].

The router microarchitecture consists of a connection of input Virtual Channel (VC) buffers, a crossbar switch, and allocators (for VCs and crossbar switch). Arriving data is stored at the input buffers, which are divided into Virtual Channels (VC) to prevent deadlock and increase throughput [5]. The VC allocator selects one VC for each input, and the switch allocator (arbiter) decides where each input will be routed to at the output port. The arbiter will be discussed in much more detail in the next chapter. The switch is usually implemented as a matrix crossbar shown in Figure 4. The matrix crossbar consists of several horizontal input wires over vertical output wires, with tri-state buffers at the cross-points to allow each input port to be electrically connected to any output port [1, 5].

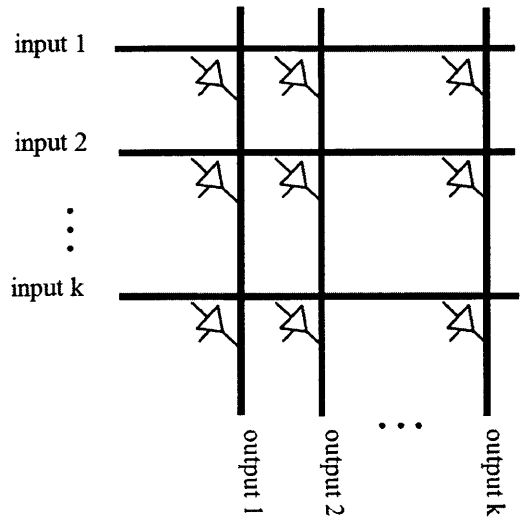

Figure 4: Matrix crossbar switch with tri-state buffers at the cross-points [1, 5].

By examining the microarchitecture of the router, it becomes clear that the crossbar and input buffers will be the most critical components of the router. As we increase the number of ports or the link widths, we expect the input buffer size to grow linearly, however the price will be paid by a quadratic increase in the size of the crossbar.

1.5. PRELIMINARY ANALYSIS

To understand the trend of how the router and the interconnection mesh dynamics change with link width, we first conducted a preliminary analysis by looking at the energy per cycle versus the link width using a numerical network simulation based on the Orion 1.0 1.0 analytical models, adjusted to the 22nm technology [4, 6]. We consider an energy constrained NoC system with a clock frequency of 2.5GHz and with $N_r = 256$ cores/routers [4]. The results from this numerical simulation are plotted in Figure 5.

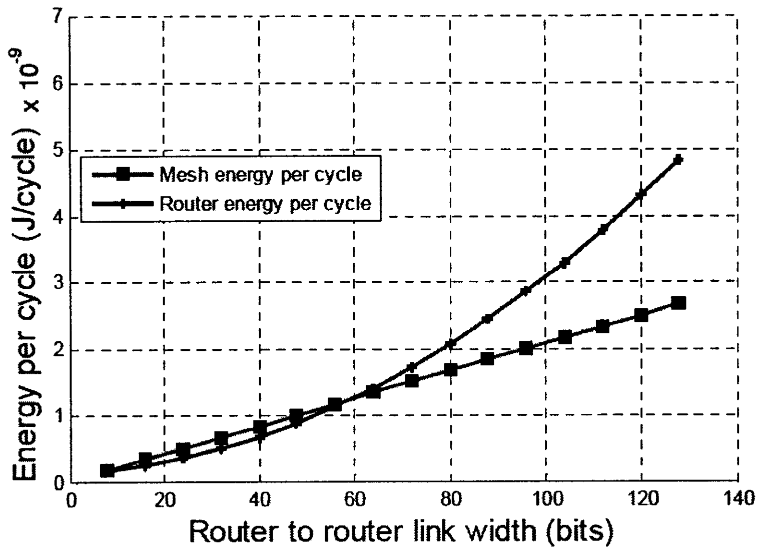

Figure 5: Energy per cycle versus link width for different components of the baseline NoC system [4].

In Figure 5, we have shown the router, interconnection mesh, I/O, and the total energy per cycle versus the link widths, for our baseline NoC system described above. We observe an approximately quadratic increase in the router energy per cycle as the link width increases. This is consistent with our previous observation that as the number of wires in the crossbar grows; its power and area dominate compared to the other router components and increase quadratically. Also, as the router link width increases, we have to correspondingly increase the link width of the interconnection mesh, which leaves less energy available for the I/O when the total energy budget for on-chip and off-chip communication is fixed. Therefore, we see a linear rise in the mesh channel energy per cycle [4]. Also, it is interesting to note that the router energy is comparable to the mesh energy, confirming our belief that the router is a critical component of the system.

Finally, to understand the trend of how the crossbar size and the input buffer size grows with the number of ports, we obtained an estimate of the total gate area for the crossbars by synthesizing them using Cadence RTL Compiler in 90nm technology. We emphasize that for this preliminary analysis, the crossbar designs that we generated were not placed-and-routed since we just wanted a quick estimate of the area. The actual placed-and-routed area numbers are expected to be larger. Furthermore, we also synthesized various SRAM Register File memory modules for the input buffers also generated using a 90nm technology. These memory modules were generated using a commercially available Artisan memory generator software which will be discussed in the next chapter. The area numbers for the memory modules correspond to placed-and-routed designs, assuming that 5-flits are stored at each input port. The area versus number of ports plot for these designs are shown in Figure 6. We would like to emphasize that this 5-flit buffer and crossbar designs are just a preliminary setup to illustrate the relative size of the router components.

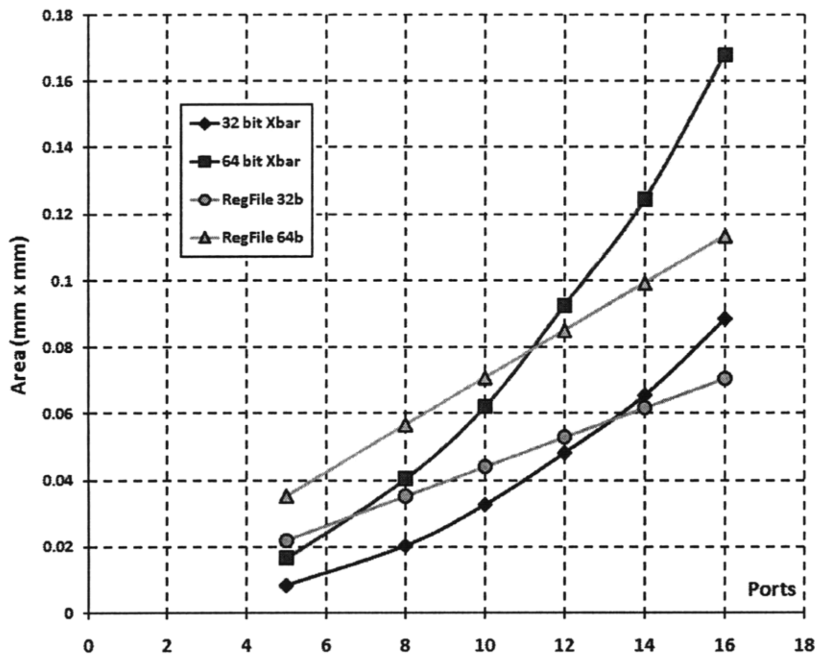

Figure 6: Areas versus number of ports for matrix crossbars (Xbars) and 5-words SRAM register file (RegFile) memory modules in 90nm technology. Results for both 32-bit and 64-bit flits are shown.

By analyzing the trend of Figure 6, it is clear that the area of the crossbar increases approximately quadratically as the number of ports or the link width increases. The area of the register file memory modules increases linearly as expected. In addition, we note that the crossbar area clearly dominates, and we remind the reader that area numbers for placed-and-routed crossbars are expected to be larger than the estimates shown in Figure 6.

2. ROUTER MICROARCHITECTURE

In this section we will discuss the details of the router microarchitecture used for this research project. To keep the study simple and to conform to our analytical Orion 1.0 1.0 models that we have used to predict power [4], we have omitted the use of Virtual Channels (VCs). Including VCs would just add another layer of arbitration at the input ports of the router. As we will show later in this thesis, the power dissipation of the arbiter is relatively small and negligible compared to the crossbar power and overall router power dissipation.

Figure 7 depicts the block diagram of the router microarchitecture. In this block diagram, the letter p corresponds to the number of ports (in this case $p = 5$), and the letter w refers to the port width which is also the flit size. To analyze the power and area trade-off for our routers and to explore the router design space, we will be varying the parameters p and w.

The first stage of the router consists of an input buffer to store the incoming flits that make up the messages. The header bits of each flit are supplied to the State module which determines the type of the flit (head flit, body flit, or tail flit), and the output destination of the message. Once the output port has been determined using a lookup table, a request signal (*req*) is sent to the Round Robin Arbiter. After arbitration, the arbiter sends a *grant* signal to the priority encoder, which connects the corresponding input and output ports of the crossbar so that the flit can traverse through the crossbar and reach its destination. At the same time, the arbiter also sends a *credit* signal to the corresponding input port to release one flit in the buffer so that a new flit can take its place.

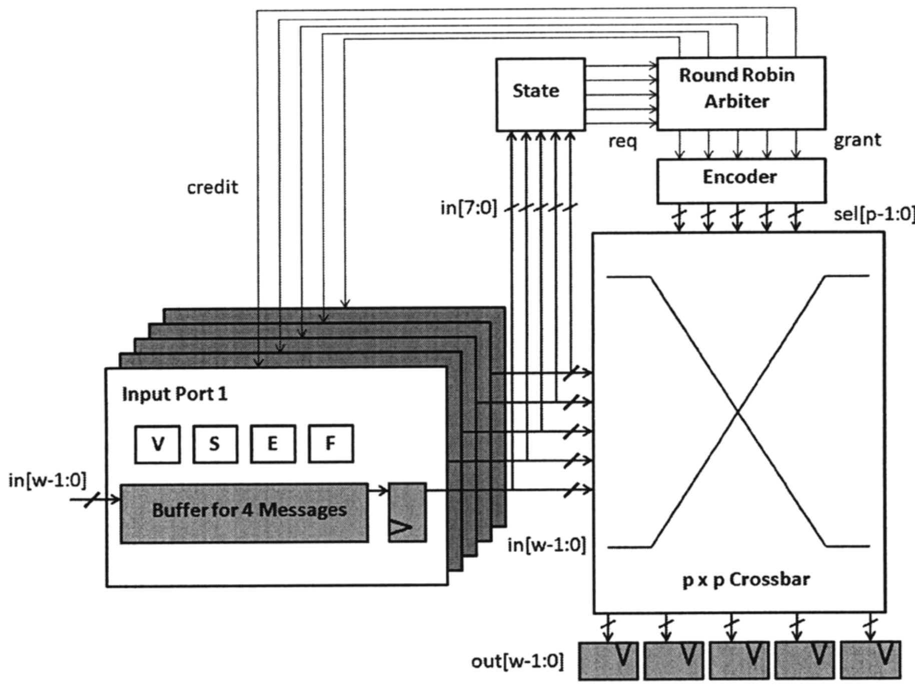

Figure 7: Router microarchitecture for this study. The letter p corresponds to the number of ports and w represents the port width. In this diagram p = 5.

Before we discuss each of the blocks described in Figure 7 individually and in more detail, we are first going to go over the flit partition that we used throughout this project. We assume that each input port can store a total of four 512-bit messages and the flit size *w* will be varied. For the system we will consider, we assume that we have a network consisting of 64 individual cores. Thus, we require 6-bits to represent the address of each core. In addition, we need 2-bits to represent the incoming flit identifier: head, body, or tail. Therefore, for a flit size of *w*, the first 8-bits are used to indicate the flit type and the address of the destination core (represented as in[7:0] in Figure 7). The partition of the flit is shown in the diagram of Figure 8.

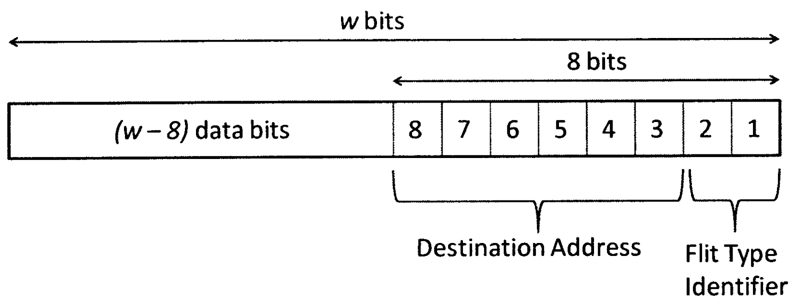

Figure 8: Flit partition. 2-bits are used for the flit identifier and 6-bits for the destination address of the flit.

In Table 1, we have shown the different flit identifiers we have used to code the flit types. We will now discuss the implementation of the various router components in Figure 7 in more detail.

Flit Type	**Flit Identifier**
Head	11
Body	10
Tail	01
Not-Used	00

Table 1: Flit Identifier types and codes.

2.1 INPUT BUFFERS

The input buffers for this router were generated using the Artisan Register File Memory Generator (rf_2p_adv) for the IBM CMOS9SF 90nm process technology. The Artisan memory generator essentially produces an array of 2-port SRAM cells with separate read and write lines so that data can be read and written to different addresses concurrently, as shown in Figure 9. A separate read and write wordline (and address bits) are used to allow concurrent read/write operations. For each input port of the router, we will have an array of SRAM cells consisting of w columns and b rows. The number of columns w is the flit size, which is the same as the port width w, and will be varied as we investigate different designs. The input buffer is designed so that it can store four 512-bit messages, thus a total of 2048-bits. The number of rows *b* varies depending on the flit size (port width) *w*. For example, if the flit size *w* = 32-bits, then *b* = 2048/32 = 64. Therefore, we would require a 64-word 32-bit SRAM at each input port of the router to store four messages if the flit size is 32-bits.

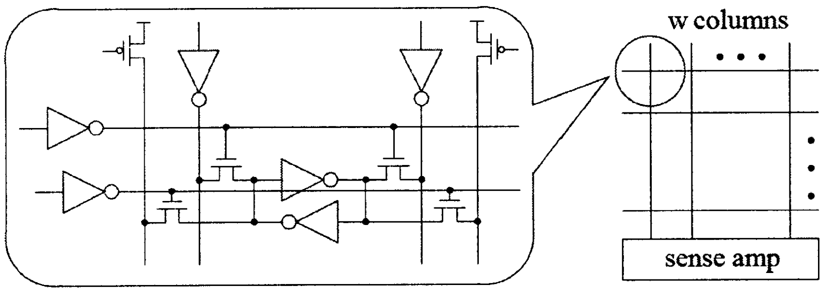

Figure 9: Schematic of a 2-port SRAM cell used for the router's input buffers [7].

As shown in Figure 9, data is written on the write ports using drivers, whereas for read operations, the read ports are first precharged and then a sense amp is used to detect the value stored in the SRAM cell.

In addition to storing the incoming flits that make up the four messages, the input buffers also contain a few registers to account for the status of each input port. The status is accounted for using the four variables shown in the input port of Figure 7, namely *V*, *S*, *E*, and *F*. The first variable *V* refers to a 1-bit Valid register, which asserts that the incoming input flit is valid. The variable *S* represents the available Space (number of words) in the input buffer SRAM array. Its length varies depending on the number of words used. The Space variable *S* is decremented by one each time a flit is written into the buffer, and it is incremented by one each time the buffer receives a *credit* signal from the Arbiter (Figure 7). The letter *E* represents the Empty 1-bit register, which is set to a logical "1" if the input buffer is empty. And finally, the letter *F* corresponds to the Full 1-bit register, which is set to a logical "1" if the input buffer is full.

The pin structure of the Artisan SRAM memory array is shown in Figure 10. The memory consists of a total of 8-pins, including 4-buses. The pins CLKA, CENA, AA, and QA control the read operation of the memory, whereas CLKB, CENB, AB, and DB control the write operation (Figure 10).

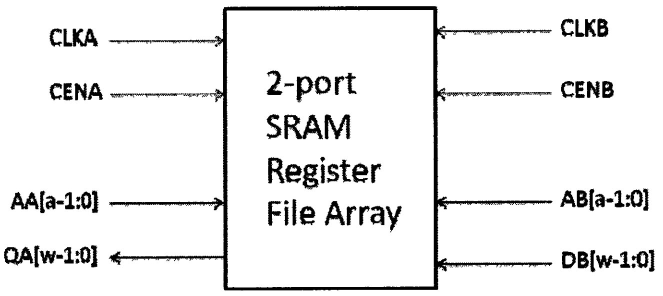

Figure 10: SRAM memory pin and bus diagram.

Access to the SRAM Register File array is synchronous and triggered by the rising edge of the clock signals CLKA and CLKB. AA is the read word line address, while AB is the write word line address. Although these ports are fully independent, it is not possible to read and write the same address at the same time. In the event of a read/write collision, the write operation is guaranteed while the read is undefined. CENA and CENB are the read enable and write enable lines respectively. QA corresponds to the bus that reads data out of the SRAM, whereas DB is the bus that writes data into the SRAM. The signals CENA, AA, QA, and the signals CENB, AB, DB, are latched at the rising edge of the clocks CLKA and CLKB respectively.

A read operation begins when the read word line enable signal CENA is a logical "0". When this happens data is read from the memory location specified by the read address bus AA[*a-1:0*] and the data read out of the SRAM appears on the read port QA[*w-1:0*]. When CENA is at a logical "1" value, the read port of the SRAM enters standby mode. During this mode the address is disabled and the data stored in the memory is retained.

Similarly, a write operation begins when the write word line enable signal CENB is at a logical "0", after which the data on the bus DB[*w-1:0*] is written into the SRAM cells whose address is specified by the write address bus AB[*a-1:0*]. When CENB is a logical "1", the write port of the SRAM enters standby mode. During this mode the address AB and the data input DB lines are disabled, and the data stored in the memory is retained.

The SRAM read and write enable signals CENA and CENB are controlled using the Valid V, Empty E, and Full F registers. Since both CENA and CENB are active low signals, they are given by:

$$\text{CENA} = !\,(\,(V \mid F)\ \&\ !(E)\,)$$

$$\text{CENB} = !\,(\,V\ \&\ !(F)\,)$$

The flits read out of each read port are latched to a register at the rising edge of the clock cycle as shown in the input port of Figure 7. The first 8-bits of the flit are then fed to the State module which will be discussed in the next section.

2.2 STATE MODULE

The State module has 4 main functions. First, it evaluates the flit identifier to find out whether the incoming flit is a head, body, or tail flit. Next, it uses the 6-bit destination address in the flit to determine which router output port the flit/message should be sent to using a lookup table. The output port address for each flit/message is stored in the State module. Then the State module sends a request (*req*) signal to the arbiter to allow the flits to traverse the crossbar. And finally, the State module reserves the output port so that other input ports cannot use the same output port until it is released.

To illustrate an example, let's assume the router shown in Figure 7 where the number of ports $p =$ 5, therefore we have a 5x5 (5 inputs and 5 outputs) router. Using a lookup table, the flit's destination address is converted into a 3-bit code used to indicate the address of one of the 5 output ports of the router. An x-y routing algorithm is assumed. In addition to the 3-bit code for the output port, the State module contains a 1-bit store for Head flit detection (H), 1-bit for Tail flit detection (T), and 1-bit to

Reserve (*R*) the output port so that it cannot be taken by another input port (Figure 11). Therefore, the State module will contain a 6-bit entry for each input port assuming a 5x5 router as shown in Figure 11. The request signal (*req*) for each input port is given by the Boolean operation:

$$req = ((H \mid T) \,\&\, R)$$

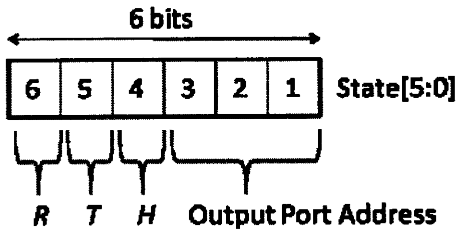

Figure 11: Illustration for the State module for a 5x5 router. H stand for Head bit detection, T for Tail bit detection, and R for Reserving the crossbar output port.

2.3 ROUND ROBIN ARBITER

We chose a simple Round Robin arbiter for our router [5]. A Round Robin arbiter was chosen since it is simple to implement and because it exhibits strong fairness. The Round Robin arbiter operates by assigning a low priority to a request that has just been granted access to the crossbar. All other pending requests will be serviced before the priority rotates around in a circular fashion [5]. Once a request is granted access to the crossbar, the arbiter sends a *grant* signal to the encoder which is described next.

2.4 ENCODER

The Encoder accepts the *grant* signal from the Round Robin arbiter, and using the Output Port Address (S[0:2] in Figure 11) stored in the State module, the Encoder generates a *p*-bit select line signal for each output port, where *p* is the number of input ports. The select signal is issued such that only one of the *p*-bits is high while the remaining bits are low. This ensures that at any point in time, only one input port can drive an output port at the crossbar.

2.5 MATRIX CROSSBAR

The matrix crossbar was introduced and presented in the Router Basics section of the Introduction chapter. A schematic of the matrix crossbar is presented below in Figure 12, where the switches represent tri-state buffers. The matrix crossbar (or distributed multiplexer) essentially consists of several horizontal input wires over vertical output wires, with tri-state buffers at the cross-points to allow each input port to be electrically connected to an output port. As shown in Figure 7, the matrix crossbar takes as input the select (*sel[p-1:0]*) vectors from the Encoder and connects the corresponding input to the appropriate output line as determined by the State module. Each output line on the matrix crossbar is assigned its own select vector, and at any point in time, only one of the *p*-bits in the select vector can be a logic high, activating the corresponding tri-state buffer (Figure 12). Thus, only one input port can drive an output port at any given time. In the 5x5 matrix crossbar example of Figure 12, inputs {in1, in2, in3, in4, in5} are routed to the following outputs {out2, out1, out5, out4, out3}.

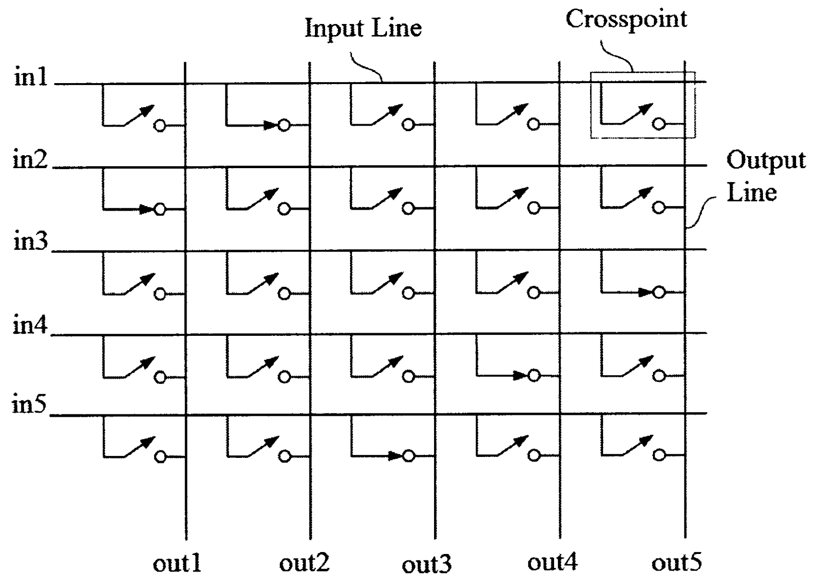

Figure 12: A 5x5 Matrix Crossbar where the switches represent tri-state buffers. The inputs {in1, in2, in3, in4, in5} are routed to the outputs {out2, out1, out5, out4, out3}.

The output ports of the crossbar are latched to a register as shown in Figure 7. The width of each input and output port depends on the flit size *w*, and will be varied as we investigate different designs.

2.6 MULTIPLEXER CROSSBAR

Crossbars can come in different flavors, and another type of crossbar we will be looking into is the multiplexer crossbar (or centralized multiplexer) shown in Figure 13.

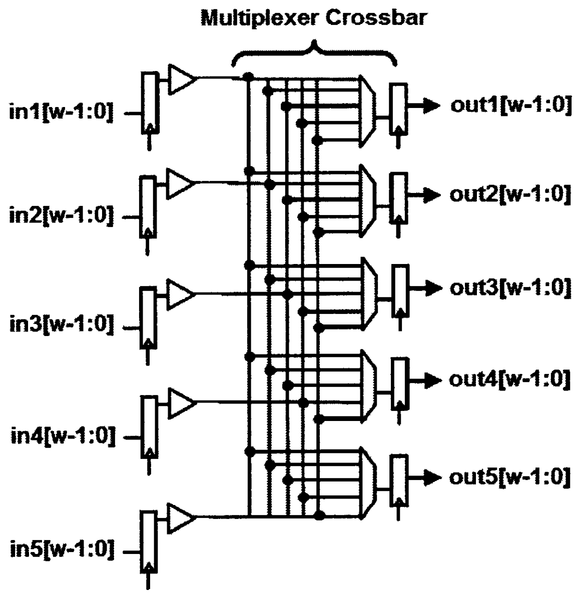

Figure 13: A 5x5 Multiplexer Crossbar. The input and output registers that latch the data are also shown [9].

The multiplexer crossbar operates in a manner completely analogous to the matrix crossbar, in the sense that it uses the same select line (*sel[p-1:0]*) vectors from the Encoder and connects the corresponding input to the appropriate output line as determined by the State module. In this case a multiplexer is used for switching. Each input port connects to a multiplexer at the output ports, and the select line is used to choose the appropriate input port for each output. The input and output registers shown in Figure 13 are the same registers that were shown in the router microarchitecture diagram of Figure 7.

3. EVALUATION METHODOLOGY

In order to understand the power versus area tradeoff and explore the router microarchitecture design space, we will be looking at several different router designs. In this chapter we will first review our system under investigation, define our design constraints, present the variables that we used to explore the design space, and discuss our ASIC design methodology used to place-and-route each design. Finally, we will specify the floorplans used for the various router designs synthesized using the IBM CMOS9SF 90nm process technology.

Our system under investigation consists of a collection of an 8x8 array of 64-cores connected to each other with a mesh interconnection network, as shown Figure 14. Each core in Figure 14 is shown with its own local router, represented as a black dot. To analyze the router design space, we have chosen a router close to the network bisection. This router under consideration has been circled in red on Figure 14.

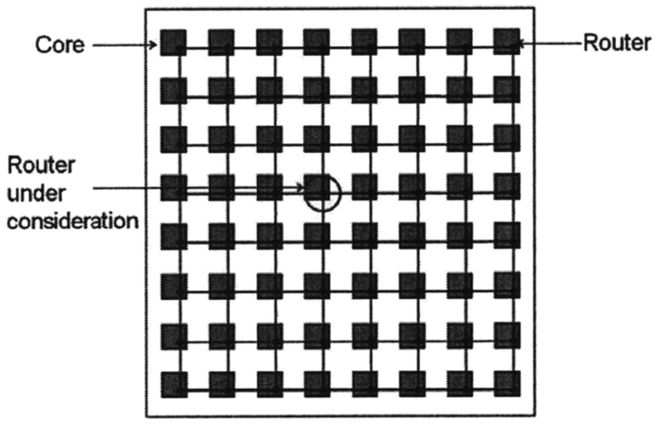

Figure 14: The 64-core system under investigation. The router under consideration is circled in red.

Since the network shown in Figure 14 consists of 64-cores, we need a 6 digit binary number to represent the address of each core. The exact number of routers will vary depending on the number of ports used, however Figure 14 illustrates a router with 5-ports, one port for each of the directions North, South, East, and West, and one port for the connection to the core. In addition, we will assume that an x-y routing algorithm is used to transmit the flits and messages. Finally, we assume that the aspect ratio for our routers is fixed to 1, and that each input port of the router buffers four 512-bit messages.

To analyze the router design space, we are going to look at router designs with three different numbers of input and output ports, which we defined earlier as the parameter *p*. More specifically, we will look at 5x5, 8x8, and 12x12 routers (*p* = 5, 8, and 12 respectively). In addition to the variation in the number of ports, we are also going to vary the port width *w* for each router. The port width *w* will define the flit size of the message. We will be looking at designs with a port width of 32-bits, 64-bits, and 128-bits. This will give us 9 different router designs. Finally, as pointed out in the previous chapter, we are interested in analyzing the power and area tradeoff for routers made with two different kinds of crossbars: matrix crossbars and multiplexer crossbars. This will give us a total of 18 different router designs that we will evaluate. The different router designs that we will investigate have been summarized in Table 2 below. Each design is given an identification label, for example the label 5p32bit-mat means "5 ports 32-bits matrix" router, whereas 5p64b-mux means "5 ports 64-bits multiplexer" router.

Each router given in Table 2 will be run at a frequency of 1GHz, with 128 randomly generated input data samples arriving at each input port. Moreover, we will be looking at two different operational regimes of the routers. The first regime will allow us to get the maximum power dissipation of the router by switching each input port of the crossbar at every clock cycle; we will call this the Deterministic No Contention (DNC) mode. For this regime, the output destination port for each input port is fixed for the entire duration. The second regime will be to measure the power dissipation with contention under uniform random traffic (i.e. port selection), and we will call this the Uniform Random (UR) data mode.

DESIGN LABEL	PORTS *p*	PORT WIDTH *w* (bits)	CROSSBAR TYPE
5p32b-mat	5	32	Matrix
5p64b-mat	5	64	Matrix
5p128b-mat	5	128	Matrix
5p32b-mux	5	32	Multiplexer
5p64b-mux	5	64	Multiplexer
5p128b-mux	5	128	Multiplexer
8p32b-mat	8	32	Matrix
8p64b-mat	8	64	Matrix
8p128b-mat	8	128	Matrix
8p32b-mux	8	32	Multiplexer
8p64b-mux	8	64	Multiplexer
8p128b-mux	8	128	Multiplexer
12p32b-mat	12	32	Matrix
12p64b-mat	12	64	Matrix
12p128b-mat	12	128	Matrix
12p32b-mux	12	32	Multiplexer
12p64b-mux	12	64	Multiplexer
12p128b-mux	12	128	Multiplexer

Table 2: The different router designs analyzed to explore the router design space.

3.1 ASIC DESIGN METHODOLOGY

Although we presented a flowchart of our ASIC toolflow in Chapter 1, we are now going to describe how we went about to generate each router design. To illustrate our design methodology, consider the example of a 5x5 128-bit matrix router (design 5p128b-mat).

After writing and testing the behavioral verilog code for the 5p128b-mat router, the router design was synthesized using Cadence RTL Compiler, which converts the verilog behavioral description into a verilog gate-level netlist. The only component that is not generated by the RTL compiler is the SRAM memory buffers. As pointed out in the previous Chapter, we used the Artisan SRAM Register File

Memory Generator software to obtain our input buffers. The Artisan software produces a *gds* file which contains the physical layout of the SRAM buffers, as well as a *vclef* file that contains the abstract and pin location information for the SRAM buffers. Since the Artisan software already gives us an optimized physical layout of the memory buffers in *gds* format, it does not need to be synthesized by the RTL Compiler and Encounter. The *gds* file can be imported directly into Cadence Virtuoso according to the directions given in Appendix A.

Therefore, in the verilog behavioral description of the 5p128b-mat router, the module that describes the SRAM input buffer is left empty; only its input and output pins are defined in the module description. When RTL Compiler sees an empty module in a verilog code, it treats that module as a Hard Macro by default. The empty module code for the memory buffers (module *rf_2p_adv*) can be seen in the verilog code given in Appendix C.

Once we have the synthesized gate-level netlist of our router from RTL Compiler, the next step is to floorplan the design. To floorplan the design using our automated scripts, we first run the scripts and specify a floorplan density of 50%. This provides us an estimate of the size of the chip. Next, we use this information along with the size of the SRAM memory buffers to deduce the width and length of the chip. Since the SRAM modules were defined as Hard Macros by the RTL Compiler, by importing the *vclef* file of the memory modules in Encounter, the SRAM modules appear as Hard Macros in the floorplan chip area. To move and rotate these SRAM modules around the chip, we used the following Encounter commands:

setObjFPlanBox Instance *instance_name llx lly urx ury*

orientateInst *instance_name* [R0 R90 R180 R270]

The first line above places the SRAM memory module inside the chip area. This command uses the name of the instance (*instance_name*) and the lower-left x (*llx*), lower-left y (*lly*), upper-right x (*urx*), and upper-right y (*ury*) coordinates to move the SRAM modules within the chip area. The second line allows us to rotate the instance by a certain fixed number of angles: R0 means a 0° rotation, R90 is a 90° rotation, R180 is 180°, and R270 is a 270° rotation. The final floorplan for our 5p128b-mat router is shown in Figure 15.

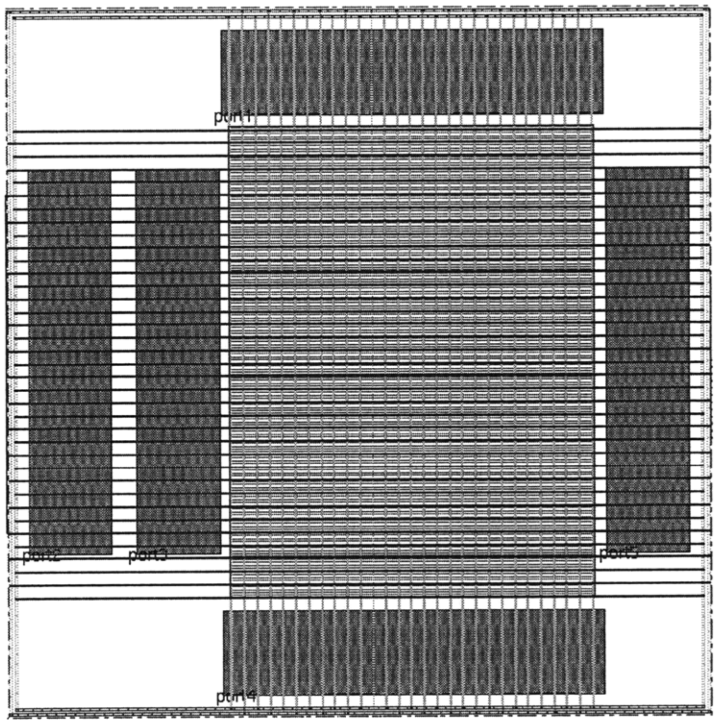

Figure 15: Floorplan for 5x5 128-bit matrix router.

In Figure 15, the outer ring is the VSS power ring, and the inner ring is the VDD power ring. The dark green rectangles in Figure 15 are the Hard Macros SRAM memory modules.

Once we have the floorplan we run Cadence Encounter again, which now does placement and routing for the whole chip. The Amoeba view of the final design is shown in Figure 16, which basically shows the placement of the standard library cells within the core area. It can be seen that the core area in Figure 16 has been utilized as much as possible to minimize the chip area. The SRAM memory Hard Macros appear in dark green in Figure 16. Finally, the final placed-and-routed chip design is shown in Figure 17. The steps used for importing the router design into Cadence Virtuoso and obtaining the post-layout extracted parasitic netlist have been given in Appendix A.

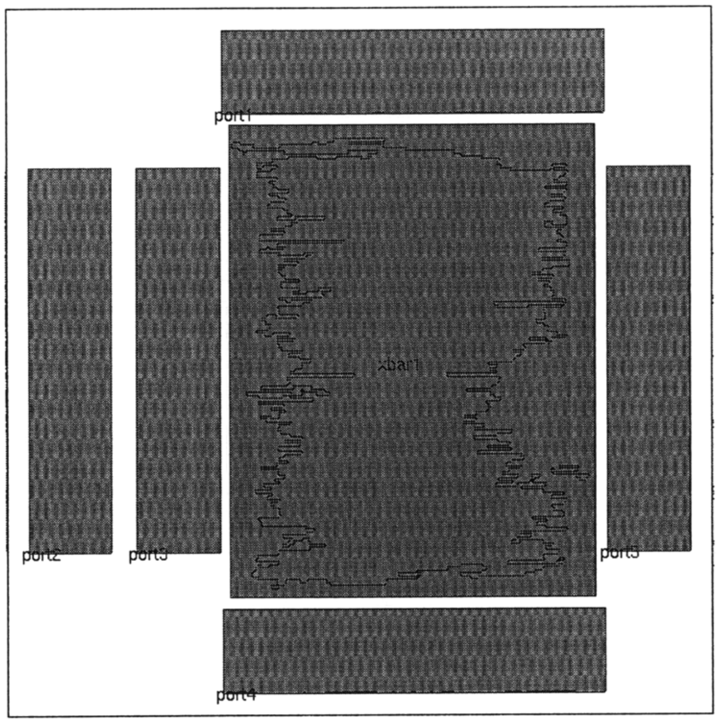

Figure 16: Amoeba view of the 5x5 128-bit matrix router. The red region inside shows the placement of the standard library cells.

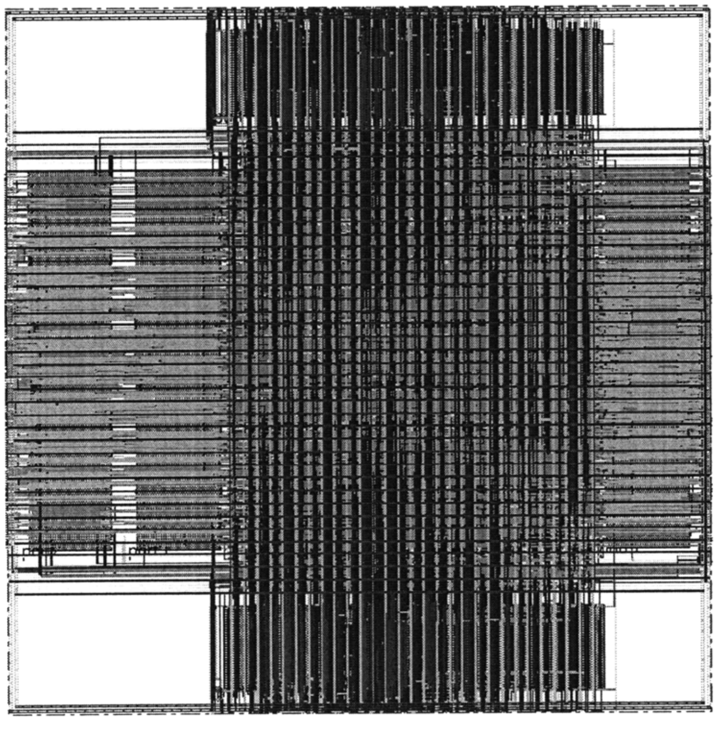

Figure 17: Final placed-and-routed design for 5x5 128-bit matrix router.

3.2 FLOORPLAN FOR 5x5 ROUTERS

In this section we will discuss the core partitions, as well as the floorplan used for the 5x5 router designs. In Figure 18, we have shown the array of 64-cores that make up our network. Each core is given an address from 0 to 63, thus we use a 6-bit number to denote the address of each core. For a 5x5 router, each core has its own local router denoted by the colored partition in Figure 18. The router associated with core 27 will be our baseline router under investigation.

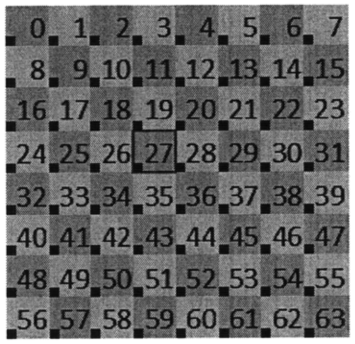

Figure 18: A network of 64-cores. Each core is given a 6-bit address value and delimited by a color. Each router is represented by a black square. The router at core 27 is the design under investigation.

The floorplan for the 5x5 32-bit matrix and multiplexer crossbar routers (designs 5p32b-mat and 5p32b-mux in Table 2) is given below in Figure 19. As we specified earlier, we have constrained our design to have an aspect ratio of one. Also shown in Figure 19 are the VSS and VDD supply power rings, made up with the highest (global) metal layers in the IBM9SF process, Metal layers 7 and 8. The vertical I/O pins are created in Metal 6 (shown in orange), while the horizontal I/O pins are created in Metal 5 (shown in purple). The 5 ports of the router correspond to the North, South, East, and West cores, and the fifth port connects to core 27. For a router with a flit size of 32-bits, a 64-word SRAM buffer is required to hold four 512-bit messages at each input port.

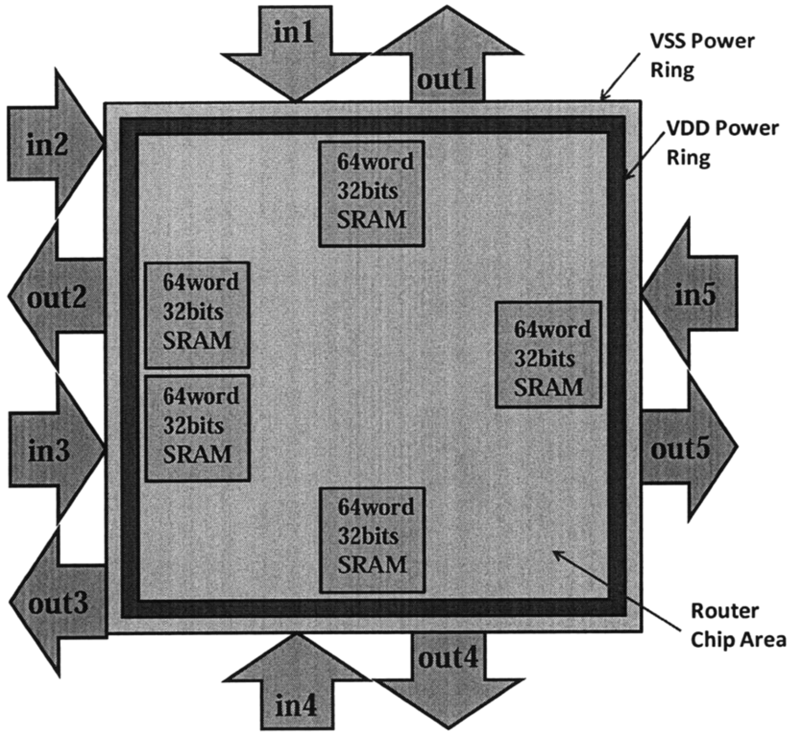

Figure 19: Floorplan used for the 5x5 32-bit matrix and multiplexer routers. The vertical pins are created using Metal 6 (orange), while the horizontal pins are created in Metal 5 (purple).

Figure 20 portrays the floorplan used for the 5x5 64-bit matrix crossbar and multiplexer crossbar routers (designs 5p64b-mat and 5p64b-mux in Table 2). The legend used for Figure 20 is exactly the same as in Figure 19. The only difference is that the I/O port width is 64-bits now, and to store four 512-bit messages, a 32-word 64-bit SRAM buffer has been added at each input port.

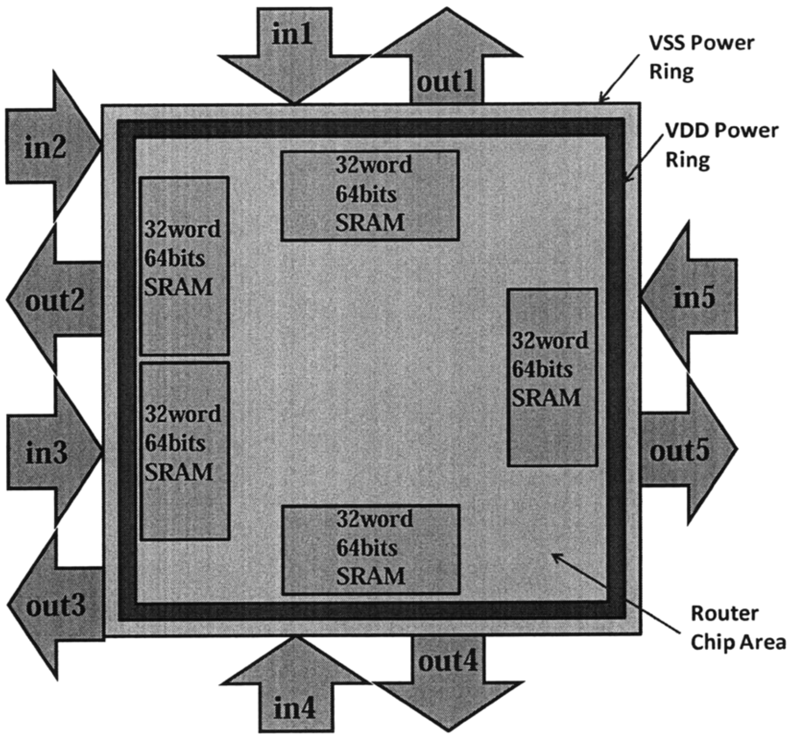

Figure 20: Floorplan used for the 5x5 64-bit matrix and multiplexer routers. The vertical pins are created using Metal 6 (orange), while the horizontal pins are created in Metal 5 (purple). A 32-word 64-bit SRAM input buffer is used at each port to store 4 messages.

Figure 21 shows the floorplan for the 5x5 128-bit matrix crossbar and multiplexer crossbar routers (designs 5p128b-mat and 5p128b-mux in Table 2). Each input buffer is now made up of a 16-word 128-bit SRAM to accommodate the 128-bit flit size. The irregular arrangement for the SRAM input buffers for ports 2 and 3 was chosen to minimize the area since we fixed the aspect ratio of the chip to 1. Finally, for the sake of clarity, we will omit the VDD/VSS power ring and Router Chip Area labels in the floorplans for higher-radix routers to reduce the clutter on the figures.

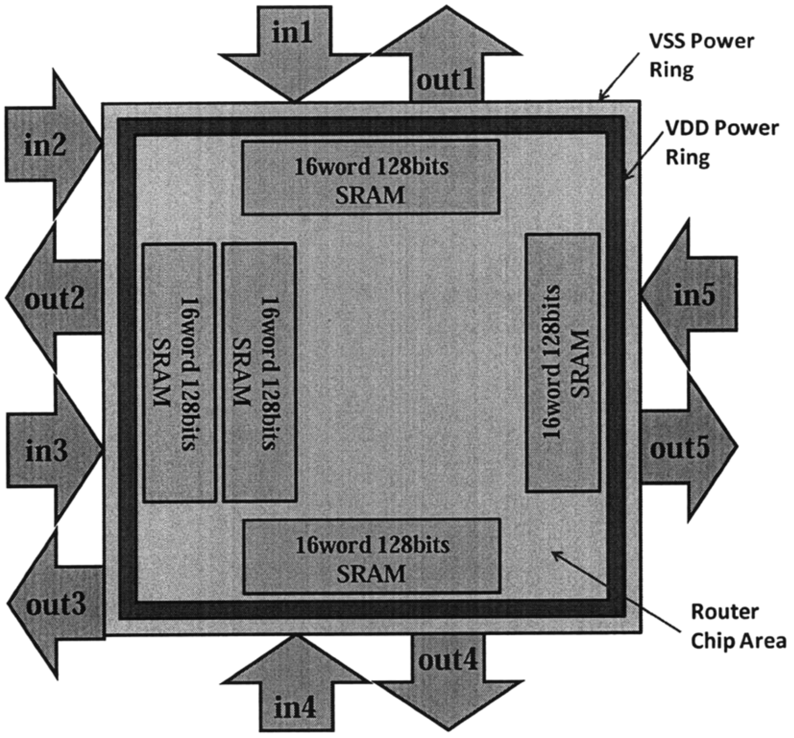

Figure 21: Floorplan used for the 5x5 128-bit matrix and multiplexer routers. The peculiar arrangement for the input buffers for ports 2 and 3 was chosen to minimize the area since the aspect ratio is constrained to 1.

3.3 FLOORPLAN FOR 8x8 ROUTERS

Now we will look at the core partitions and the floorplan designs for the 8x8 router designs. The core partition for this case has been shown in Figure 22. The colored regions shown in Figure 22 represent the cores that are sharing the same router. In this case, we have 16-groups of 4-cores and each group has its own router.

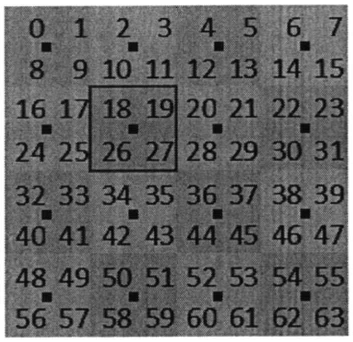

Figure 22: Core partition for 8x8 routers. Each colored region depicts the cores that share the same router. In this configuration, 16 routers are shared among 64-cores, with each router connected to 4-cores. The router under consideration is shared among cores 18, 19, 26, and 27.

Every router in each group connects to four neighboring cores, in addition to the usual North, South, East, and West connections thus giving a total of 8 ports.

Figure 23 portrays the floorplan used for the 8x8 32-bit matrix and multiplexer crossbar routers (designs 8p32b-mat and 8p32b-mux in Table 2). The legend in Figure 23 is similar to the one in Figure 20 through Figure 22, where the outer light green ring represents the VSS Power Ring, and the inner dark green ring shows the VDD power ring. Due to the symmetry of the 8x8 router, each side of the router is assigned 2 sets of I/O ports. Once again, the vertical pins are created using Metal layer 6 (orange), while the horizontal pins are created using Metal layer 5 (purple). The floorplans for the 64-bit 8x8 (designs 8p64b-mat and 8p64b-mux) and 128-bit 8x8 (designs 8p128b-mat and 8p128b-mux) routers are shown in Figure 24 and Figure 25 respectively. In each case, the width of the I/O ports and the attributes of the input buffer SRAMs are adjusted to comply with the flit size and port widths.

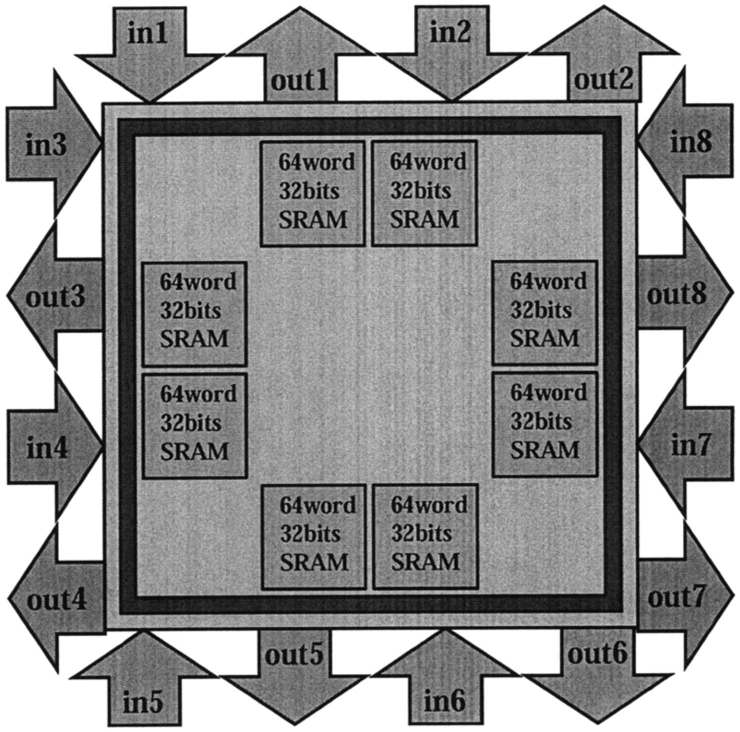

Figure 23: Floorplan used for the 8x8 32-bit matrix crossbar and multiplexer crossbar routers.

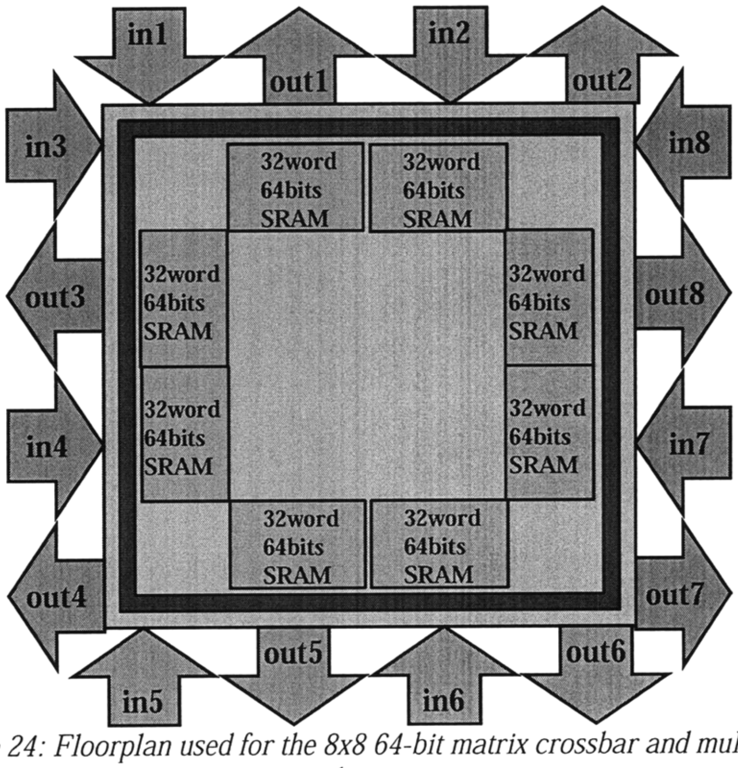

Figure 24: Floorplan used for the 8x8 64-bit matrix crossbar and multiplexer crossbar routers.

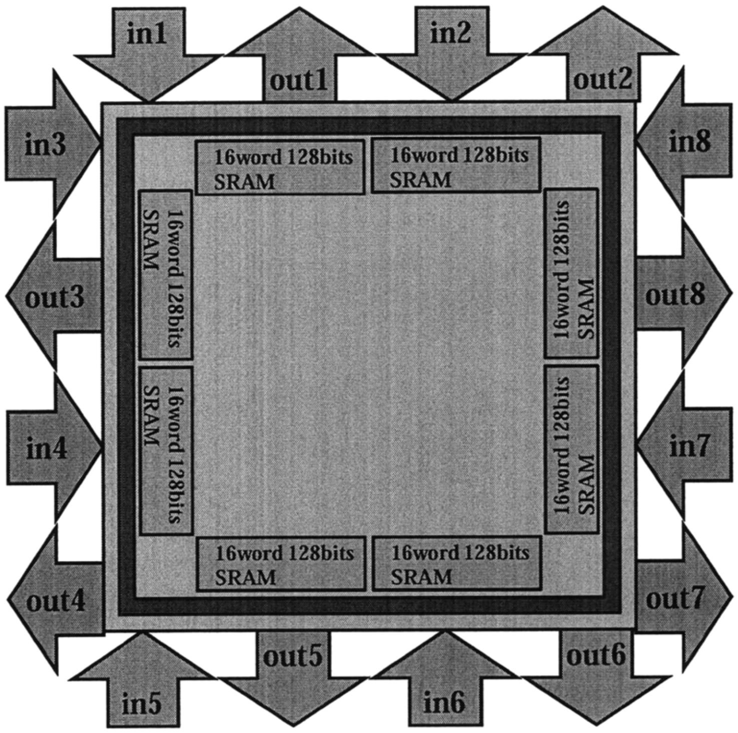

Figure 25: Floorplan used for the 8x8 128-bit matrix crossbar and multiplexer crossbar routers.

3.4 FLOORPLAN FOR 12x12 ROUTERS

Finally we will look at the core partitions and the floorplan designs for the 12x12 routers. The core partition for this case has been shown in Figure 26. The colored regions shown in Figure 26 represent the cores that are shared among the same router. In this case, we have 8-groups of 8-cores and each group has its own router. Every router in each group connects to eight neighboring cores, in addition to the usual North, South, East, and West connections thus giving a total of 12 ports.

Figure 27 shows the floorplan used for the 12x12 32-bit matrix and multiplexor crossbar routers (designs 12p32b-mat and 12p32b-mux in Table 2). Because of the symmetry of the 12x12 router, each side of the router is assigned 3 sets of I/O ports and input buffers. The floorplans for the 64-bit 12x12 (designs 12p64b-mat and 12p64b-mux) and 128-bit 12x12 (designs 12p128b-mat and 12p128b-mux) routers are shown in Figure 28 and Figure 29 respectively. Once again the width of the I/O ports and the attributes of the input buffer SRAMs are adjusted to comply with the flit size. For Figure 29, the buffers could not be placed the way we did for the 5x5 128-bit router to save area, since the 12x12 radix router needed more area to accommodate the crossbar.

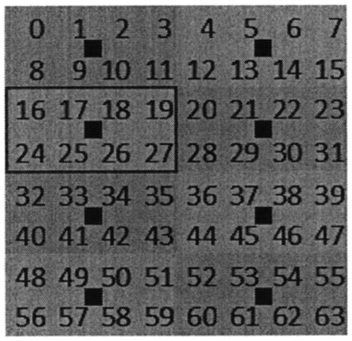

Figure 26: Core partition for 12x12 routers. Each colored regions depicts the cores that share the same router. In this configuration, 8 routers are shared among 64-cores, with each router connected to 8-cores. The router connected to cores 16, 17, 18, 19, 24, 25, 26, and 27 is the one we will evaluate.

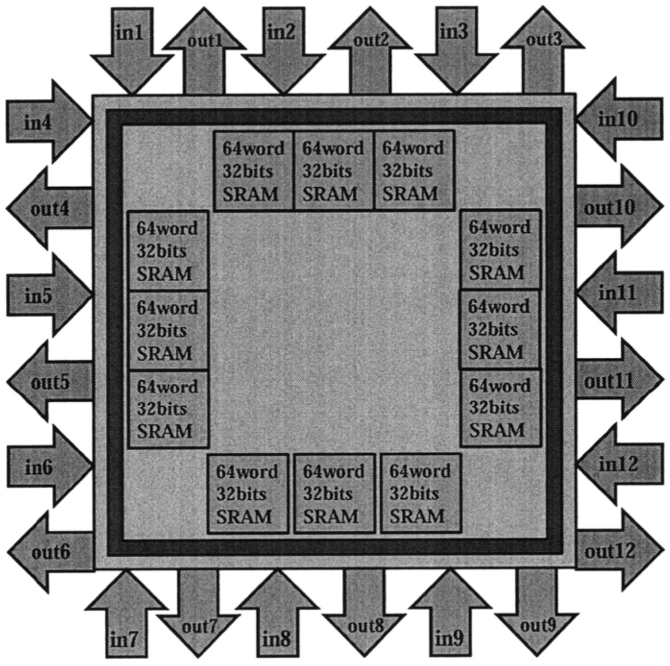

Figure 27: Floorplan used for the 12x12 32-bit matrix crossbar and multiplexer crossbar routers.

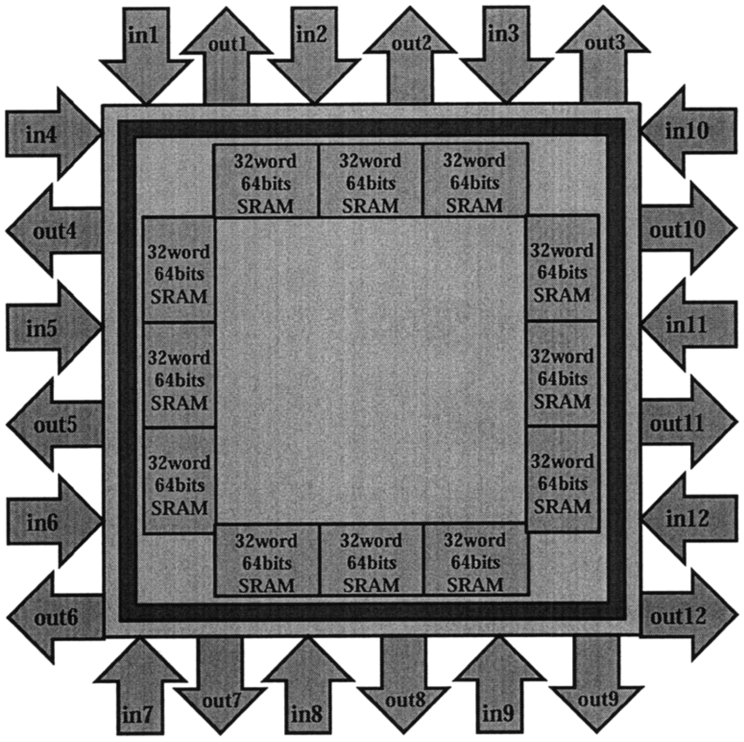

Figure 28: Floorplan used for the 12x12 64-bit matrix crossbar and multiplexer crossbar routers.

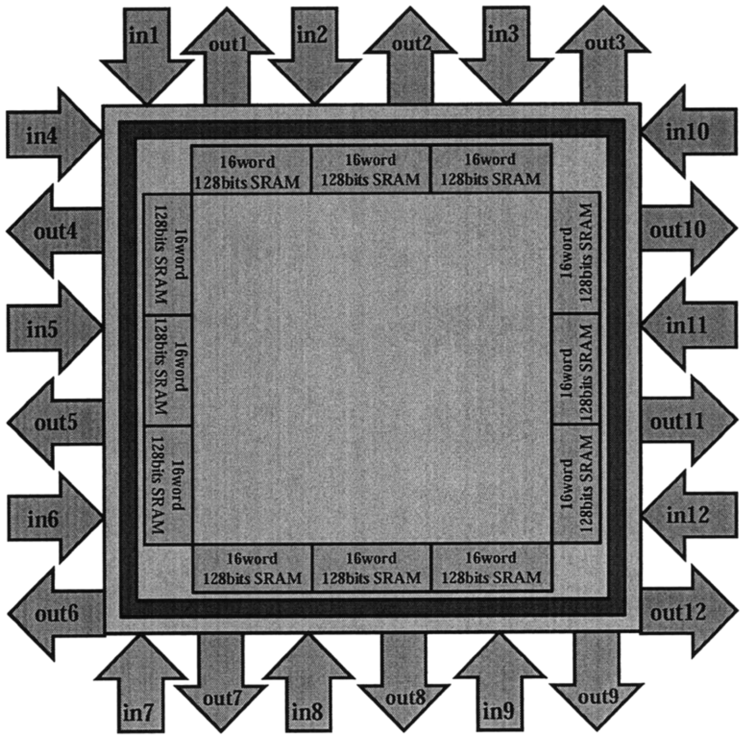

Figure 29: Floorplan used for the 12x12 128-bit matrix crossbar and multiplexer crossbar routers.

4. RESULTS AND ANALYSIS

We will now present the results for all the router designs given in Table 2. Each design was synthesized using an IBM CMOS9SF 90nm process technology using the described ASIC methodology. After each design was synthesized, it was placed-and-routed using a 60% density and its area was measured. Next a post-layout parasitic HSPICE netlist of the routers was extracted and simulated in Nanosim with a 1.2V power supply to verify the timing, functionality of each router and to get an estimate of the power dissipation. The numerical results gathered for each router have been tabulated in Appendix A.

4.1 AREA AND POWER TRADEOFF

The first plot in Figure 30 shows the Area versus Port Width variation for each router. The solid dark colored lines correspond to the curves for the matrix routers, whereas the dashed light colored lines represent the curves for the multiplexer routers. Looking at these curves, it becomes clear that as the port width and the number of ports increases; the area of the matrix router grows substantially faster than the area of the multiplexer router. Furthermore, we would like to point out that for the 32-bit 5x5, 8x8, and 12x12 multiplexer routers (designs 5p32b-mux, 8p32b-mux, and 12p32b-mux), the area could have been reduced further but was constrained by our fixed aspect ratio of 1 and the size of the SRAM input buffers. With that consideration in mind, it appears that the matrix routers exhibit an approximate quadratic increase in area as the port width and number of ports increases, whereas the area of the multiplexer routers tend to grow somewhat linearly with the number of ports and port width for this narrow port range. Clearly these results reinforce our preliminary analysis where we deduced that the matrix crossbars would be the most dominating component of the system as the number of ports and port width grew.

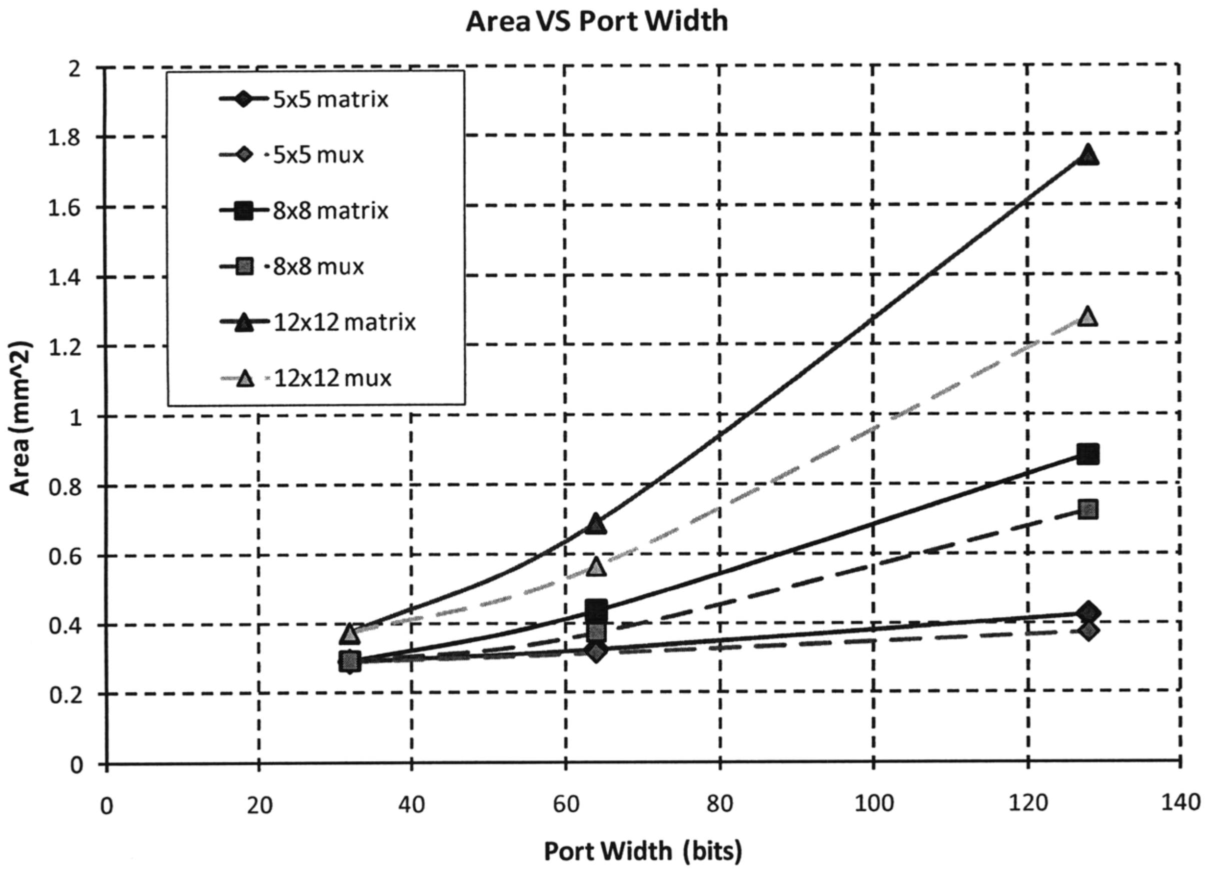

Figure 30: Area variation versus the port width for the various router designs we evaluated. The matrix routers tend to exhibit a quadratic-like increase in area whereas the increase for the multiplexer router is less significant.

The power versus port width plots for our routers are shown in Figure 31. In this plot, the dark colored lines represent the curves for the matrix routers, whereas the light colored lines correspond to the multiplexer routers. In addition, the solid lines correspond to the power dissipation in the DNC (Deterministic No Contention) regime defined in the previous chapter, and the dashed lines represent the UR (Uniform Random) regime. Once again, we see that the matrix router clearly dominates and shows an approximate quadratic-like behavior, whereas the multiplexer routers' power increases somewhat linearly. This was expected since the power dissipation is proportional to the area and tends to exhibit a similar trend. Although this fact seems apparent for the 12x12 router case, it is harder to see for the 8x8 and 5x5 routers due to the scaling of the vertical axis in Figure 31.

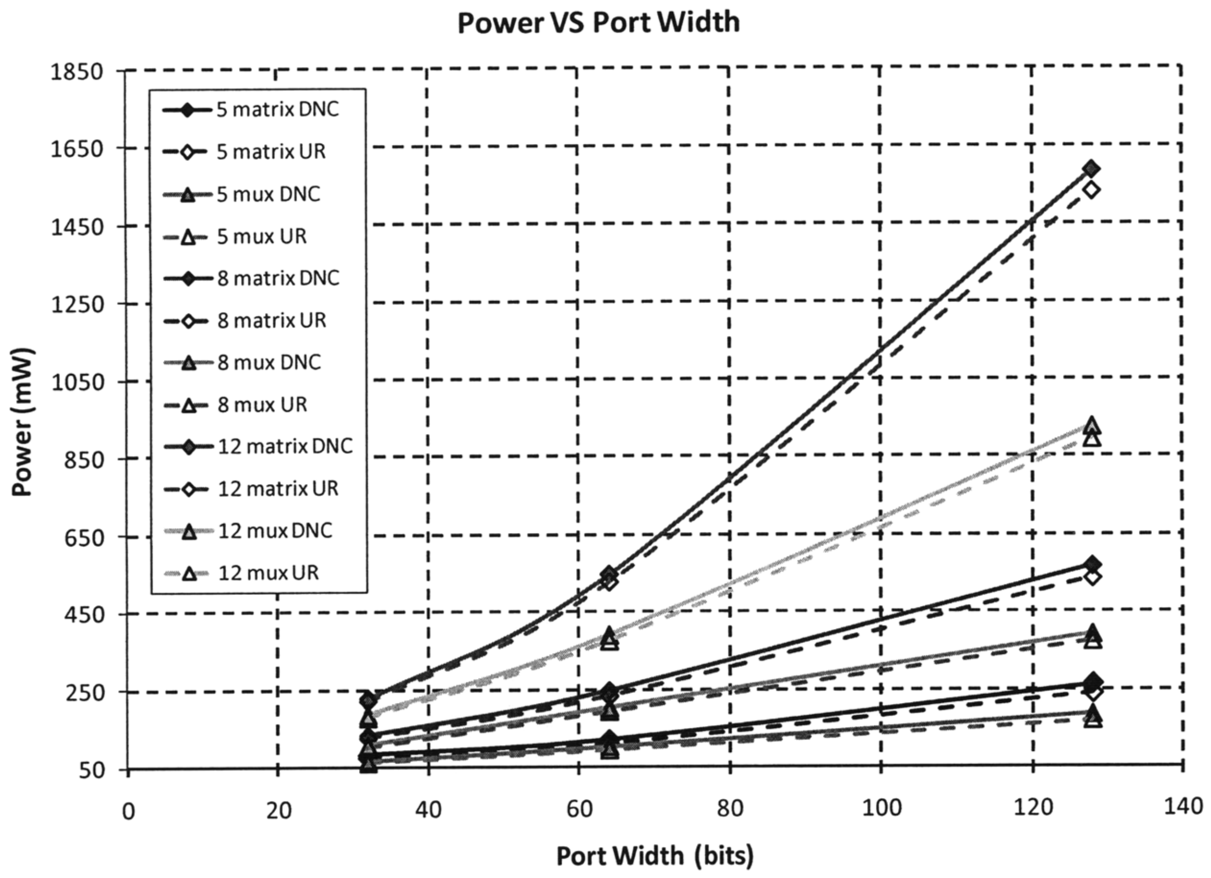

Figure 31: Power variation versus the port width for the various router designs we evaluated. The solid lines correspond to the DNC (Deterministic, No Contention) regime whereas the dashed lines represent the UR (Uniform Random) regime. The DNC regime corresponds to the case where each input is assigned a fixed output so that the crossbar switches every clock cycle. In the UR case, contention is present therefore some ports remain idle for certain clock cycles.

The most important lesson that we can derive from these area and power results can be obtained by looking at the costs and benefits of aggregation, by keeping the network bisection bandwidth of these different NoCs approximately the same. Table 3 lists this information for the Matrix Router designs, whereas Table 4 represents the power and area aggregation numbers for the Multiplexer Router designs.

Matrix Design	Area (mm^2)	Power (mW)
4 x 5p32b-mat	1.1664	332.304
1 x 8p64b-mat	0.4356	246.3924
4 x 5p64b-mat	1.2996	484.4544
1 x 8p128b-mat	0.8836	568.2672
2 x 8p32b-mat	0.5832	264.6312
1 x 12p64b-mat	0.6889	546.8928
2 x 8p64b-mat	0.8712	492.7848
1 x 12p128b-mat	1.7424	1584.54
8 x 5p32b-mat	2.3328	664.608
1 x 12p128b-mat	1.7424	1584.54

Table 3: Aggregation numbers for Matrix Router designs.

Mux Design	Area (mm^2)	Power (mW)
4 x 5p32b-mux	1.1664	268.3056
1 x 8p64b-mux	0.3721	203.268
4 x 5p64b-mux	1.2544	410.5872
1 x 8p128b-mux	0.7225	391.0116
2 x 8p32b-mux	0.5832	215.8464
1 x 12p64b-mux	0.5625	389.5896
2 x 8p64b-mux	0.7442	406.536
1 x 12p128b-mux	1.2769	926.2188
8 x 5p32b-mux	2.3328	536.6112
1 x 12p128b-mux	1.2769	926.2188

Table 4: Aggregation numbers for Multiplexer Router designs.

Looking at the aggregation data in Tables 3 and 4, we see that aggregation in only useful for routers with narrow channels. Moreover, for the matrix router designs, aggregation is only useful for the first case where four 5p32b-mat routers are compared with one 8p64b-mat router. In this case, we see a reduction in both the area and power when we use the 8p64b-mat router. For the multiplexer router designs, we see that aggregation is successful for the first two cases where we compare four 5p32b-mux routers with one 8p64b-mux router, and four 5p64b-mux routers with one 8p128b-mux router. Looking at the aggregation data, we see that aggregation is successful only as a function of the complexity of the crossbar versus the bit/port width. Since the matrix crossbar exhibits a stronger quadratic-like behavior than the multiplexer router, aggregation is more beneficial for it on narrow channels.

To further clarify the power scaling of the matrix and multiplexer with the port width and number of ports, in Figure 32 we have plotted the power breakdown of the matrix and multiplexer routers in a bar graph. Before examining these two figures in more detail, we would like to remind the reader that the round robin arbiter does not scale with the port width since its operation depends only on the number of ports. Therefore, the power dissipation of the arbiter is insignificant compared to the power of the crossbars and buffers. In Figure 32, the arbiter's power can barely be seen since it is completely dwarfed by the power of the crossbars and buffers.

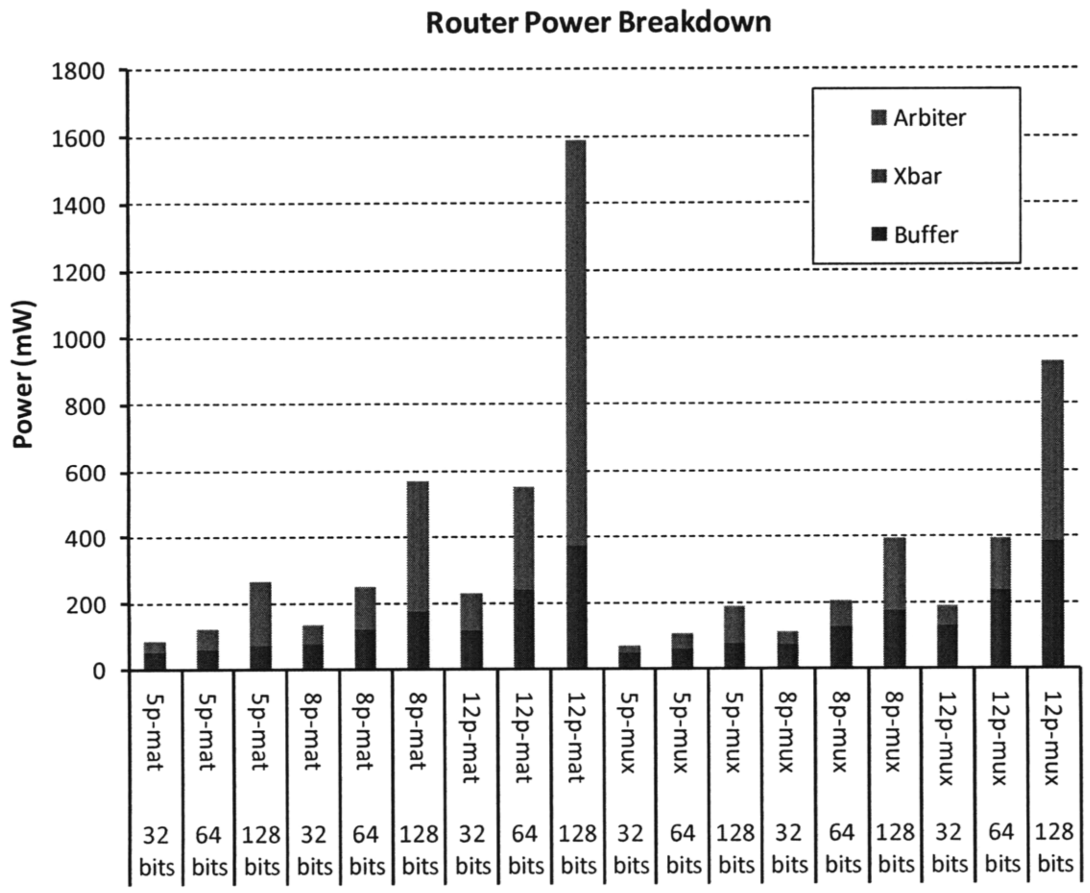

Figure 32: Power breakdown for the Matrix and Multiplexer routers.

By examining the bar graph of Figure 32, the quadratic increase in the power dissipation of the matrix crossbar becomes evident and clear. For the small 32-bit matrix crossbars, the power approximately doubles as the port width increases to 64-bits. However going from 64-bits to 128-bits more than doubles the power dissipation of the crossbar, and in the case of the 12x12 matrix router, it appears that the power dissipation almost quadruples when the port width doubles. This is clear evidence of the quadratic nature we observed in the previous results.

The power breakdown bar graphs for the multiplexer routers in Figure 32 do not exhibit a strong quadratic behavior like the matrix routers. Although the increase in the power as the port width doubles is not exactly linear, the power does not scale as quickly as the power for matrix crossbars. As the number of wires in the multiplexer router grows, the number or wires in the crossbar increase, thus the multiplexer crossbar tends to exhibit a similar trend as the matrix crossbar; however it is nowhere as significant as in the matrix crossbar case. For example, going from the 12x12 64-bit multiplexer crossbar to the 128-bit crossbar, we see the power approximately triples. However, this increase is still far more energy efficient than the matrix crossbar. From these results, we clearly see that the multiplexer routers are more power and area efficient than matrix routers. The matrix routers quickly exhibit a quadratic increase in area and power as the number of ports and port width increases. On the other hand, the multiplexer routers show a much lower increase.

Another interesting graph to look at is the Router Area Overheard per Core Versus the number of ports. We define this as the area of the router compared to the area of the cores that share the same router (Figure 33). The area of a single core in 90nm technology is estimated to be 6.25mm^2 using Orion 1.0 1.0 models adjusted to 90nm [4]. In Figure 33, the solid line curves represent the matrix routers, while the dashed curves depict the multiplexer routers. Looking at these curves, we deduce that as the number of ports increases from 5-ports, we see a reduction in the Router Area Overheard per Core since the area of the router is shared among multiple cores: 1 core for 5-ports, 4 cores for 8-ports, and 8 cores for 12-ports. However for the 12x12 128-bit matrix router case, we start seeing an increase in the Router Area Overheard per Core. This happens due to the fact that as we increase the number of ports and port width, the quadratic increase in the crossbar area (and hence the router area) starts to dominate and reduces the savings obtained by sharing the router among multiple cores. For this case, we observe that there is a sweet spot between 8 to 10 ports where a local minimum in the Router Area Overheard per Core is obtained.

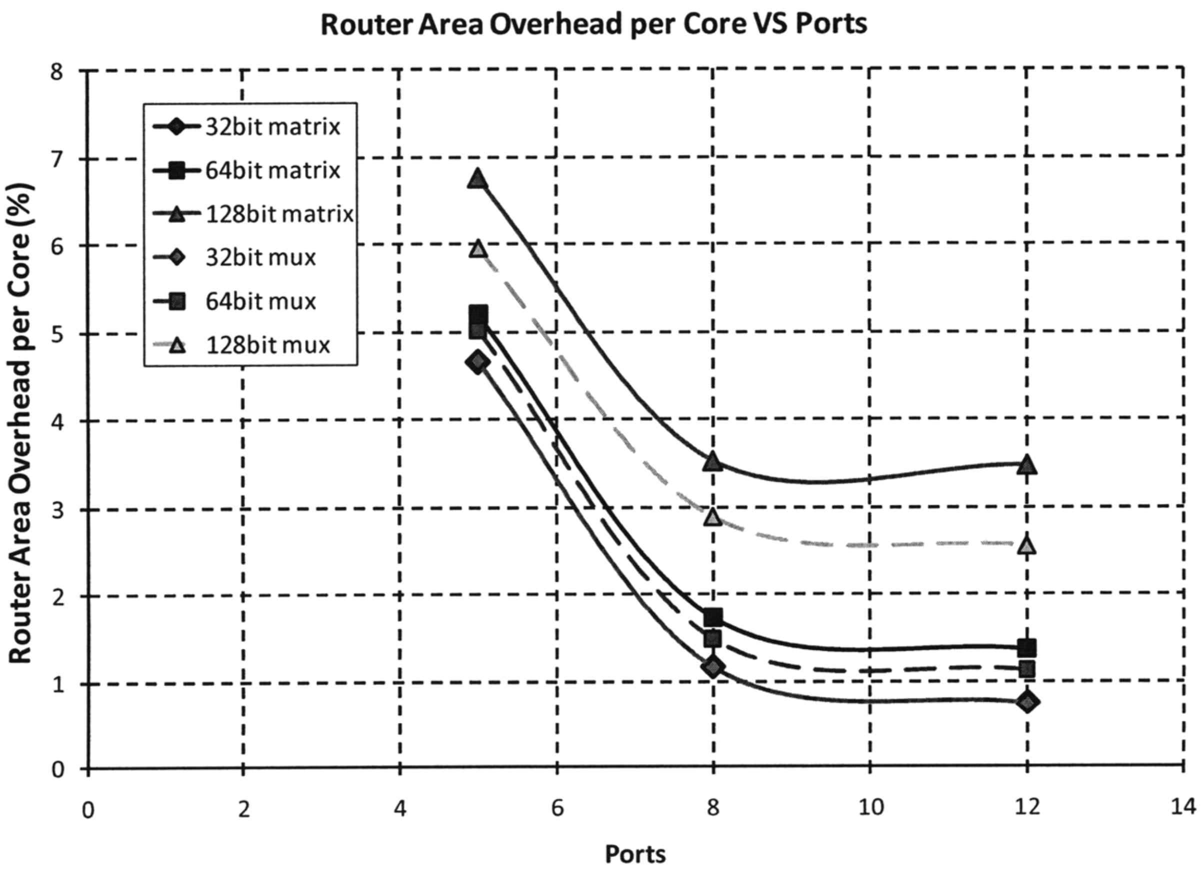

Figure 33: Router Area Overhead per Core versus Ports.

An interesting point to note is that the multiplexer routers consistently gives a lower Router Area Overheard per Core than the matrix routers. The 12x12 128-bit multiplexer router does not exhibit an increase in the Router Area Overheard per Core as the matrix router did. As stated previously, we would expect this to change if we increase the number of ports or port width beyond a certain critical value. Moreover, using the aggregation data in Tables 3 and 4, we see that the savings in the Router Area Overheard per Core by going from four 5-ports 32-bit routers to one 8-port 64-bit router is more than 63%, whereas going from eight 5-ports 32-bit to one 12-port 128-bit routers the savings is only 25%.

Also, it is worth noticing that using 32-bit or 64-bit routers results in a larger area savings when the ports of the router are increased so that the same router is shared among multiple cores. Therefore, even in a situation where we would want to have a core that processes 128-bits of data, it might be far more area and power efficient to use two 64-bit routers running in parallel at the core frequency than using a router that runs at the core frequency with a port width of 128-bits.

Most importantly, we see that by switching from a NoC with 5-port routers to higher radix routers while keeping the network bisection bandwidth approximately constant, the savings in the Router Area Overheard per Core can be up to 63%. Therefore, we see that using 8x8 routers in NoC systems is more power and area efficient than using 5x5 routers.

4.2 COMPARISON WITH ANALYTICAL POWER MODELS

We will now look at the comparison of the results obtained from this thesis, with results obtained using the Orion 1.0 Analytical Models for power estimation. The ratio of the power obtained for the matrix buffers and crossbar is compared to the power obtained for these components using Orion 1.0 in Figure 34. Looking at Figure 34, it is surprising to see that the component that shows the most deviation is actually the buffer, and not the crossbar. As stated previously, the buffers were generated using the Artisan software tool, therefore they are expected to be highly optimized. The crossbar was synthesized using Cadence Encounter, and designing the crossbar in the custom design flow can possibly reduce the power further, and hence diminish the discrepancy between the synthesized results and the analytical estimates. Moreover, the crossbar is inherently a much simpler circuit compared to the SRAM buffers. It could be possible that the Orion 1.0 models for the buffers are not accurate and need to be adjusted.

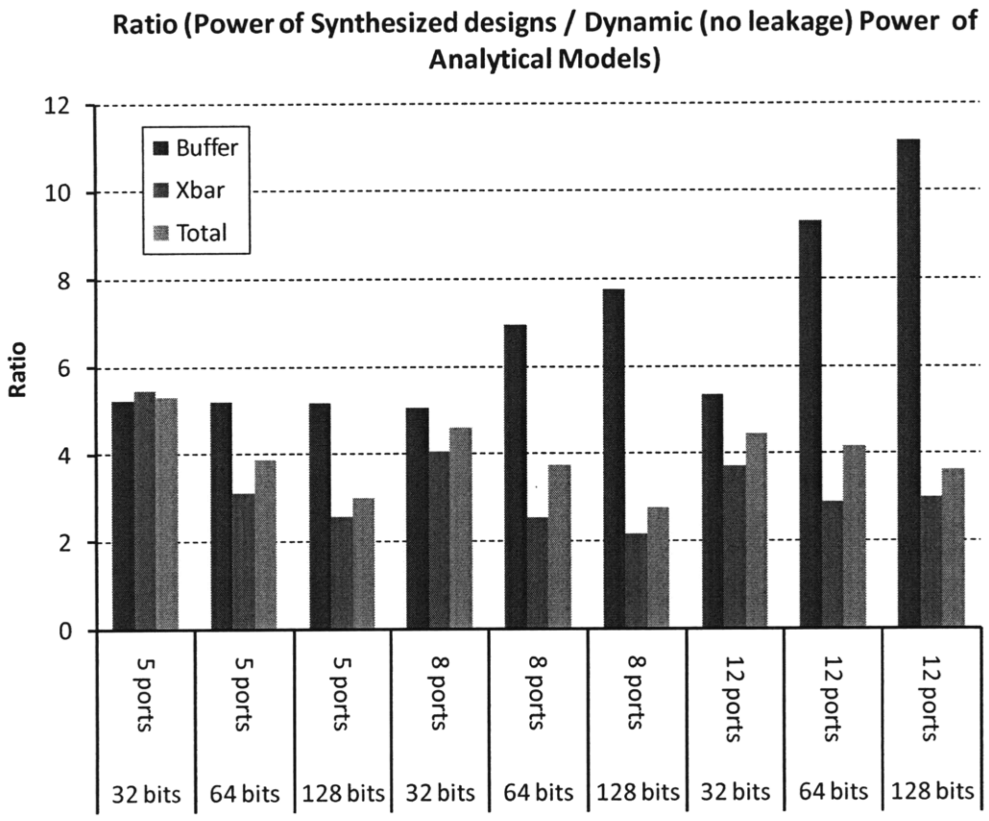

Figure 34: Ratio of router buffers and matrix crossbar power compared to Orion 1.0 analytical power estimates

5. CONCLUSION AND FUTURE WORK

Looking at the latest surge in multi-core chips, it is evident that the trend in the computer hardware industry to keep on increasing the number of cores on a chip will continue. Clearly multi-core systems are here and they are here to stay. However, shifting from single-core processors to multi-core systems presents new problems and challenges to be solved. One such problem is the design of the router microarchitecture to cope with ever increasing bandwidth requirements, while minimizing the area overhead and power consumption. This thesis presented a study of the area versus power tradeoff for routers using data obtained from designs synthesized using a commercially available IBM 90nm process technology.

The thesis showed that multiplexer routers are more area and power efficient compared to matrix routers. Matrix routers quickly exhibit a quadratic like increase in area and power as the number of ports and port width increases. The transition for multiplexer routers is not as sharp and thus they tend to be more area and power efficient. In addition, we saw that there is a real gain in area when the router is shared among 4 or more cores.

Most importantly, we show that by looking at the costs and benefits of aggregation, we see that aggregation is only useful for narrow channel routers. This is because successful aggregation is a function of the crossbar complexity versus the bit/port width. By switching from a NoC with 5-port routers to higher radix routers while keeping the network bisection bandwidth approximately constant, the savings in the Router Area Overheard per Core can be up to 63%.

For the narrow design space that we analyzed we noticed that for maximum area and power efficiency, it is much better to use a multiplexer router with 8 ports as opposed to matrix routers. However, the savings by sharing the same router among multiple cores does not continue indefinitely, since as the number of ports or port width grows, the quadratic-like increase in the crossbar size can no longer be compensated by sharing the router. So for a fixed port-width, there is always a sweet spot for the number of ports where a local minimum can be found for the Router Area Overheard per Core of the router compared to the area of the cores sharing the same router. Finally, keeping the flit size to 32-bits or 64-bits results in a larger Router Area Overheard per Core savings as opposed to using a 128-bit router. Even in situations where the core manipulates 128-bits of data, using two 64-bit routers running in parallel at the core frequency will result a larger area and power savings.

The results of this thesis will be used to calibrate our analytical Orion 1.0 tool models to match the thesis results, which should in principle be much closer to reality than results obtained through analytical models alone. In addition, the results of this research will be used to narrow down the microarchitecture of a router to be used in the Integrated Photonics Network project at the Integrated Systems Group (ISG) at the Research Laboratory for Electronics (RLE). Furthermore, we believe that the methodology presented in this thesis can be extended and used to explore the router design space for other technologies and architectures. In the near future, we would also like to compare the results of this study with the new Orion 2.0 analytical models.

6. APPENDIX A: NUMERICAL ROUTER DATA

MATRIX ROUTER

Port Width (bits)	# Ports	x (um)	y (um)	Power DNC (mW)	Power UR (mW)	Area (mm^2)	Area Fraction (%)	Pxbar (mW)	Parbiter(mW)	Pbuffer (mW)
32	5	540	540	83.076	76.0692	0.2916	4.6656	31.062	0.7152	51.2988
64	5	570	570	121.1136	110.5224	0.3249	5.1984	61.23	0.7152	59.1684
128	5	650	650	263.1828	238.8756	0.4225	6.76	187.1844	0.7152	75.2832
32	8	540	540	132.3156	126.3396	0.2916	1.1664	54.395	1.189	76.7316
64	8	660	660	246.3924	233.5752	0.4356	1.7424	122.143	1.189	123.0604
128	8	940	940	568.2672	536.7	0.8836	3.5344	390.786	1.189	176.2922
32	12	610	610	227.2464	219.0792	0.3721	0.7442	105.951	1.864	119.4314
64	12	830	830	546.8928	526.5048	0.6889	1.3778	302.753	1.864	242.2758
128	12	1320	1320	1584.54	1533.8256	1.7424	3.4848	1209.549	1.864	373.127

MUX ROUTER

Port Width (bits)	# Ports	x (um)	y (um)	Power DNC (mW)	Power UR (mW)	Area (mm^2)	Area Fraction (%)	Pxbar (mW)	Parbiter(mW)	Pbuffer (mW)
32	5	540	540	67.0764	61.7712	0.2916	4.6656	19.653	0.7152	46.7082
64	5	560	560	102.6468	94.242	0.3136	5.0176	40.981	0.7152	60.9506
128	5	610	610	186.2004	170.5152	0.3721	5.9536	106.2156	0.7152	79.2696
32	8	540	540	107.9232	103.4508	0.2916	1.1664	32.674	1.189	74.0602
64	8	610	610	203.268	194.394	0.3721	1.4884	74.927	1.189	127.152
128	8	850	850	391.0116	374.7276	0.7225	2.89	212.032	1.189	177.7906
32	12	610	610	185.5896	179.8116	0.3721	0.7442	54.816	1.864	128.9096
64	12	750	750	389.5896	376.2324	0.5625	1.125	149.508	1.864	238.2176
128	12	1130	1130	926.2188	895.3848	1.2769	2.5538	538.84	1.864	385.5148

7. APPENDIX B: 5x5 ROUTER VERILOG CODE

```
`timescale 100ps/10ps

// ***********************************************
// 5port Round Robin Arbiter

module round_robin5(clk, rst, r, g, credit);

        input clk, rst;
        input [4:0] r;
        output [4:0] g, credit;
        wire [4:0] g, next_p, credit;
        reg [4:0] p;

        assign next_p = |r ? {p[3:0],p[4]} : p ;

        always @(posedge clk)
        begin
                if(rst)
                  p <= 5'b00001;
                else
                  p <= next_p;
        end

        assign g = r & p;
        assign credit = ~g;


endmodule // End of 5port Round Robin Arbiter


// ********************************************************************************************
// 128bit 16word SRAM input buffer module (empty so that it is treated as a Hard Block by RTL
Compiler)

  module rf_2p_adv (
               QA,
               CLKA,
               CENA,
               AA,
               CLKB,
               CENB,
               AB,
               DB,
               EMAA,
               EMAB
               );


   output [127:0]          QA;
   input                   CLKA;
   input                   CENA;
   input [3:0]             AA;
   input                   CLKB;
   input                   CENB;
   input [3:0]             AB;
   input [127:0]           DB;
   input [2:0]             EMAA;
   input [2:0]             EMAB;

endmodule
```

```
// ************************************************************************
// 5x5 128-bit Multiplexer Crossbar

`define portwidth 128

module mux5x5_xbar_128b(clk, in1, in2, in3, in4, in5, sel1, sel2, sel3, sel4, sel5, out1, out2,
out3, out4, out5);

        input clk;
        input [`portwidth-1:0] in1, in2, in3, in4, in5;
        input [4:0] sel1, sel2, sel3, sel4, sel5;
        output [`portwidth-1:0] out1, out2, out3, out4, out5;

        reg [`portwidth-1:0] out1, out2, out3, out4, out5;

        always @(posedge clk)
        begin

                case(sel1)
                5'b00001: out1 <= in1;
                5'b00010: out1 <= in2;
                5'b00100: out1 <= in3;
                5'b01000: out1 <= in4;
                5'b10000: out1 <= in5;
                default:  out1 <= out1;
                endcase

                case(sel2)
                5'b00001: out2 <= in1;
                5'b00010: out2 <= in2;
                5'b00100: out2 <= in3;
                5'b01000: out2 <= in4;
                5'b10000: out2 <= in5;
                default:  out2 <= out2;
                endcase

                case(sel3)
                5'b00001: out3 <= in1;
                5'b00010: out3 <= in2;
                5'b00100: out3 <= in3;
                5'b01000: out3 <= in4;
                5'b10000: out3 <= in5;
                default:  out3 <= out3;
                endcase

                case(sel4)
                5'b00001: out4 <= in1;
                5'b00010: out4 <= in2;
                5'b00100: out4 <= in3;
                5'b01000: out4 <= in4;
                5'b10000: out4 <= in5;
                default:  out4 <= out4;
                endcase

                case(sel5)
                5'b00001: out5 <= in1;
                5'b00010: out5 <= in2;
                5'b00100: out5 <= in3;
                5'b01000: out5 <= in4;
                5'b10000: out5 <= in5;
                default:  out5 <= out5;
                endcase

        end

endmodule // End of module
// ******************************************************************************************

// Matrix Xbar 5x5 128-bit
```

```
`define portwidth 128
`define ports 5

module matrix5x5_xbar_128b(clk, in1, in2, in3, in4, in5, sel1, sel2, sel3, sel4, sel5, out1,
out2, out3, out4, out5);

        input clk;
        input [`portwidth-1:0] in1, in2, in3, in4, in5;
        input [`ports-1:0] sel1, sel2, sel3, sel4, sel5;
        output [`portwidth-1:0] out1, out2, out3, out4, out5;

        reg [`portwidth-1:0] out1, out2, out3, out4, out5;
        wire [`portwidth-1:0] outw1, outw2, outw3, outw4, outw5;

always @(posedge clk)
begin
        out1 = outw1;
        out2 = outw2;
        out3 = outw3;
        out4 = outw4;
        out5 = outw5;

end

bufif1 b1(outw1[0], in1[0], sel1[0]);
bufif1 b2(outw1[1], in1[1], sel1[0]);
bufif1 b3(outw1[2], in1[2], sel1[0]);
bufif1 b4(outw1[3], in1[3], sel1[0]);
bufif1 b5(outw1[4], in1[4], sel1[0]);
bufif1 b6(outw1[5], in1[5], sel1[0]);
bufif1 b7(outw1[6], in1[6], sel1[0]);
bufif1 b8(outw1[7], in1[7], sel1[0]);
bufif1 b9(outw1[8], in1[8], sel1[0]);
bufif1 b10(outw1[9], in1[9], sel1[0]);
bufif1 b11(outw1[10], in1[10], sel1[0]);
bufif1 b12(outw1[11], in1[11], sel1[0]);
bufif1 b13(outw1[12], in1[12], sel1[0]);
bufif1 b14(outw1[13], in1[13], sel1[0]);
bufif1 b15(outw1[14], in1[14], sel1[0]);
bufif1 b16(outw1[15], in1[15], sel1[0]);
bufif1 b17(outw1[16], in1[16], sel1[0]);
bufif1 b18(outw1[17], in1[17], sel1[0]);
bufif1 b19(outw1[18], in1[18], sel1[0]);
bufif1 b20(outw1[19], in1[19], sel1[0]);
bufif1 b21(outw1[20], in1[20], sel1[0]);

bufif1 b641(outw2[0], in1[0], sel2[0]);
bufif1 b642(outw2[1], in1[1], sel2[0]);
bufif1 b643(outw2[2], in1[2], sel2[0]);
bufif1 b644(outw2[3], in1[3], sel2[0]);
bufif1 b645(outw2[4], in1[4], sel2[0]);
bufif1 b646(outw2[5], in1[5], sel2[0]);
bufif1 b647(outw2[6], in1[6], sel2[0]);
bufif1 b648(outw2[7], in1[7], sel2[0]);
bufif1 b649(outw2[8], in1[8], sel2[0]);
bufif1 b650(outw2[9], in1[9], sel2[0]);
bufif1 b651(outw2[10], in1[10], sel2[0]);
bufif1 b652(outw2[11], in1[11], sel2[0]);
bufif1 b653(outw2[12], in1[12], sel2[0]);
bufif1 b654(outw2[13], in1[13], sel2[0]);
bufif1 b655(outw2[14], in1[14], sel2[0]);
bufif1 b656(outw2[15], in1[15], sel2[0]);
bufif1 b657(outw2[16], in1[16], sel2[0]);
bufif1 b658(outw2[17], in1[17], sel2[0]);
bufif1 b659(outw2[18], in1[18], sel2[0]);
bufif1 b660(outw2[19], in1[19], sel2[0]);
bufif1 b661(outw2[20], in1[20], sel2[0]);

bufif1 b1281(outw3[0], in1[0], sel3[0]);
bufif1 b1282(outw3[1], in1[1], sel3[0]);
```

```
bufif1 b1283(outw3[2], in1[2], sel3[0]);
bufif1 b1284(outw3[3], in1[3], sel3[0]);
bufif1 b1285(outw3[4], in1[4], sel3[0]);
bufif1 b1286(outw3[5], in1[5], sel3[0]);
bufif1 b1287(outw3[6], in1[6], sel3[0]);
bufif1 b1288(outw3[7], in1[7], sel3[0]);
bufif1 b1289(outw3[8], in1[8], sel3[0]);
bufif1 b1290(outw3[9], in1[9], sel3[0]);
bufif1 b1291(outw3[10], in1[10], sel3[0]);
bufif1 b1292(outw3[11], in1[11], sel3[0]);
bufif1 b1293(outw3[12], in1[12], sel3[0]);
bufif1 b1294(outw3[13], in1[13], sel3[0]);
bufif1 b1295(outw3[14], in1[14], sel3[0]);
bufif1 b1296(outw3[15], in1[15], sel3[0]);
bufif1 b1297(outw3[16], in1[16], sel3[0]);
bufif1 b1298(outw3[17], in1[17], sel3[0]);
bufif1 b1299(outw3[18], in1[18], sel3[0]);
bufif1 b1300(outw3[19], in1[19], sel3[0]);
bufif1 b1301(outw3[20], in1[20], sel3[0]);

bufif1 b1921(outw4[0], in1[0], sel4[0]);
bufif1 b1922(outw4[1], in1[1], sel4[0]);
bufif1 b1923(outw4[2], in1[2], sel4[0]);
bufif1 b1924(outw4[3], in1[3], sel4[0]);
bufif1 b1925(outw4[4], in1[4], sel4[0]);
bufif1 b1926(outw4[5], in1[5], sel4[0]);
bufif1 b1927(outw4[6], in1[6], sel4[0]);
bufif1 b1928(outw4[7], in1[7], sel4[0]);
bufif1 b1929(outw4[8], in1[8], sel4[0]);
bufif1 b1930(outw4[9], in1[9], sel4[0]);
bufif1 b1931(outw4[10], in1[10], sel4[0]);
bufif1 b1932(outw4[11], in1[11], sel4[0]);
bufif1 b1933(outw4[12], in1[12], sel4[0]);
bufif1 b1934(outw4[13], in1[13], sel4[0]);
bufif1 b1935(outw4[14], in1[14], sel4[0]);
bufif1 b1936(outw4[15], in1[15], sel4[0]);
bufif1 b1937(outw4[16], in1[16], sel4[0]);
bufif1 b1938(outw4[17], in1[17], sel4[0]);
bufif1 b1939(outw4[18], in1[18], sel4[0]);
bufif1 b1940(outw4[19], in1[19], sel4[0]);
bufif1 b1941(outw4[20], in1[20], sel4[0]);

bufif1 b2561(outw5[0], in1[0], sel5[0]);
bufif1 b2562(outw5[1], in1[1], sel5[0]);
bufif1 b2563(outw5[2], in1[2], sel5[0]);
bufif1 b2564(outw5[3], in1[3], sel5[0]);
bufif1 b2565(outw5[4], in1[4], sel5[0]);
bufif1 b2566(outw5[5], in1[5], sel5[0]);
bufif1 b2567(outw5[6], in1[6], sel5[0]);
bufif1 b2568(outw5[7], in1[7], sel5[0]);
bufif1 b2569(outw5[8], in1[8], sel5[0]);
bufif1 b2570(outw5[9], in1[9], sel5[0]);
bufif1 b2571(outw5[10], in1[10], sel5[0]);
bufif1 b2572(outw5[11], in1[11], sel5[0]);
bufif1 b2573(outw5[12], in1[12], sel5[0]);
bufif1 b2574(outw5[13], in1[13], sel5[0]);
bufif1 b2575(outw5[14], in1[14], sel5[0]);
bufif1 b2576(outw5[15], in1[15], sel5[0]);
bufif1 b2577(outw5[16], in1[16], sel5[0]);
bufif1 b2578(outw5[17], in1[17], sel5[0]);
bufif1 b2579(outw5[18], in1[18], sel5[0]);
bufif1 b2580(outw5[19], in1[19], sel5[0]);
bufif1 b2581(outw5[20], in1[20], sel5[0]);

endmodule

// *******************************************************************************************

// Router 5x5 128b Top Module
// ************************************************************************************************

`define ports 5
`define portwidth 128
`define addy 4
`define nwords 16

// can be substituted by mux5x5_xbar_128b.v
`include "matrix5x5_xbar_128b.v"
`include "round_robin5.v"
`include "rf_2p_adv.v"


module router5x5_128b(clk, rst, in1, in2, in3, in4, in5, out1, out2, out3, out4, out5, grant,
CENA1, CENA2, CENA3, CENA4, CENA5, rp1, rp2, rp3, rp4, rp5, CENB1, CENB2, CENB3, CENB4, CENB5,
wp1, wp2, wp3, wp4, wp5, valid1, valid2, valid3, valid4, valid5, full1, full2, full3, full4,
full5, inb1, inb2, inb3, inb4, inb5, state1, state2, state3, state4, state5, sel1, sel2, sel3,
sel4, sel5, req, credit, space1, space2, space3, space4, space5, empty1, empty2, empty3, empty4,
empty5);

// clock and reset for initialization of state registers
        input clk, rst;

// router input ports and valid data bits
        input [`portwidth-1:0] in1, in2, in3, in4, in5;
        input valid1, valid2, valid3, valid4, valid5;

// router output ports
        output [`portwidth-1:0] out1, out2, out3, out4, out5;

// STATE: size is 5bits for 5ports and 8ports. Size is 6bits for 12ports
        output [4:0] state1, state2, state3, state4, state5;

// miscellaneous signals
        input [`addy-1:0] wp1, wp2, wp3, wp4, wp5, rp1, rp2, rp3, rp4, rp5;
        output CENA1, CENA2, CENA3, CENA4, CENA5, CENB1, CENB2, CENB3, CENB4, CENB5;
        output full1, full2, full3, full4, full5;
        output [`portwidth-1:0] inb1, inb2, inb3, inb4, inb5;
        output [`ports-1:0] sel1, sel2, sel3, sel4, sel5;
        output [`ports-1:0]  req, grant, credit;
        output [`addy-1:0] space1, space2, space3, space4, space5;
        output empty1, empty2, empty3, empty4, empty5;


//  input registers to hold data read from memory
        reg [`portwidth-1:0] inb1, inb2, inb3, inb4, inb5;

        reg [4:0] state1, state2, state3, state4, state5;
         reg [`ports-1:0] allow;
        reg [`ports-1:0] sel1, sel2, sel3, sel4, sel5;
        reg [`addy-1:0] space1, space2, space3, space4, space5;
        reg             full1, full2, full3, full4, full5;
        reg    empty1, empty2, empty3, empty4, empty5;
        reg [2:0] EMAA,  EMAB;
        wire   CENA1, CENA2, CENA3, CENA4, CENA5, CENB1, CENB2, CENB3, CENB4, CENB5;
        wire [`ports-1:0] req, grant, credit;
         wire [`portwidth-1:0] out1, out2, out3, out4, out5;
         wire [`portwidth-1:0] QA1, QA2, QA3, QA4, QA5;



        always @(posedge clk)
        begin

// state*[0]= valid bit, state*[4] = tail bit detected, state*[3:1] = output address allow= state
reserve bit
// inb*[1:0] = 11 (head), 10 (body), 01 (tail)

// if rst is high, initialize the registers in the router
         if(rst)
```

```
                begin
// space is initialized to number of words - 1
                        space1 = `addy'd15;
                        space2 = `addy'd15;
                        space3 = `addy'd15;
                        space4 = `addy'd15;
                        space5 = `addy'd15;

                        full1 <= 1'b0;
                        full2 <= 1'b0;
                        full3 <= 1'b0;
                        full4 <= 1'b0;
                        full5 <= 1'b0;

                        empty1 <= 1'b1;
                        empty2 <= 1'b1;
                        empty3 <= 1'b1;
                        empty4 <= 1'b1;
                        empty5 <= 1'b1;

                        EMAA <= 1'b0;
                        EMAB <= 1'b0;

                        state1 = 5'b0;
                        state2 = 5'b0;
                        state3 = 5'b0;
                        state4 = 5'b0;
                        state5 = 5'b0;

                        allow = `ports'b0;


                end
                else
                begin


                                if (CENB1 == 0)
                                begin
                                        if (space1 == `addy'd0)
                                                full1 <= 1;
                                        else
                                                space1 = space1 - 1;
                                end
                                else
                                begin
                                        if (space1 == `addy'd0)
                                                full1 <= 1;
                                        else
                                                full1 <= 0;
                                end


                                if (CENA1 == 0)
                                begin
                                        inb1 <= QA1;
                                        if (space1 == `addy'd15)
                                                empty1 <= 1;
                                        else
                                                space1 = space1 + 1;
                                end
                                else
                                begin
                                        if (space1 == `addy'd15)
                                                empty1 <= 1;
                                        else
                                                empty1 <= 0;
                                end

//
```

```
                              if (CENB2 == 0)
                              begin
                                      if (space2 == `addy'd0)
                                              full2 <= 1;
                                      else
                                              space2 = space2 - 1;
                              end
                              else
                              begin
                                      if (space2 == `addy'd0)
                                              full2 <= 1;
                                      else
                                              full2 <= 0;
                              end


                              if (CENA2 == 0)
                              begin
                                      inb2 <= QA2;
                                      if (space2 == `addy'd15)
                                              empty2 <= 1;
                                      else
                                              space2 = space2 + 1;
                              end
                              else
                              begin
                                      if (space2 == `addy'd15)
                                              empty2 <= 1;
                                      else
                                              empty2 <= 0;
                              end
//


                              if (CENB3 == 0)
                              begin
                                      if (space3 == `addy'd0)
                                              full3 <= 1;
                                      else
                                              space3 = space3 - 1;
                              end
                              else
                              begin
                                      if (space3 == `addy'd0)
                                              full3 <= 1;
                                      else
                                              full3 <= 0;
                              end


                              if (CENA3 == 0)
                              begin
                                      inb3 <= QA3;
                                      if (space3 == `addy'd15)
                                              empty3 <= 1;
                                      else
                                              space3 = space3 + 1;
                              end
                              else
                              begin
                                      if (space3 == `addy'd15)
                                              empty3 <= 1;
                                      else
                                              empty3 <= 0;
                              end

//
```

```
                                        if (CENB4 == 0)
                                        begin
                                                if (space4 == `addy'd0)
                                                        full4 <= 1;
                                                else
                                                        space4 = space4 - 1;
                                        end
                                        else
                                        begin
                                                if (space4 == `addy'd0)
                                                        full4 <= 1;
                                                else
                                                        full4 <= 0;
                                        end


                                        if (CENA4 == 0)
                                        begin
                                                inb4 <= QA4;
                                                if (space4 == `addy'd15)
                                                        empty4 <= 1;
                                                else
                                                        space4 = space4 + 1;
                                        end
                                        else
                                        begin
                                                if (space4 == `addy'd15)
                                                        empty4 <= 1;
                                                else
                                                        empty4 <= 0;
                                        end

//

                                        if (CENB5 == 0)
                                        begin
                                                if (space5 == `addy'd0)
                                                        full5 <= 1;
                                                else
                                                        space5 = space5 - 1;
                                        end
                                        else
                                        begin
                                                if (space5 == `addy'd0)
                                                        full5 <= 1;
                                                else
                                                        full5 <= 0;
                                        end


                                        if (CENA5 == 0)
                                        begin
                                                inb5 <= QA5;
                                                if (space5 == `addy'd15)
                                                        empty5 <= 1;
                                                else
                                                        space5 = space5 + 1;
                                        end
                                        else
                                        begin
                                                if (space5 == `addy'd15)
                                                        empty5 <= 1;
                                                else
                                                        empty5 <= 0;
                                        end


// evaluating state using the header flit info
// assuming 64-cores, this CPU is number 27 near the center
// assuming x-y routing



// setting valid bit to zero if tail is detected

            if(grant[0] == 1'b1)
            begin
               if ( (state1[0] == 1'b1) & (state1[4] == 1'b1) )
                       state1[0] = 1'b0;
            end
            else

               if( (inb1[1:0] == 2'b11) & (state1[0] == 1'b0) )
               begin
                       state1[0] = 1'b1;
                       state1[4] = 1'b0;

                       case(inb1[7:2])
                       6'd0: state1[3:1] = 3'd2;
                       6'd1: state1[3:1] = 3'd2;
                       6'd2: state1[3:1] = 3'd2;
                       6'd3: state1[3:1] = 3'd1;
                       6'd4: state1[3:1] = 3'd4;
                       6'd5: state1[3:1] = 3'd4;
                       6'd6: state1[3:1] = 3'd4;
                       6'd7: state1[3:1] = 3'd4;
                       6'd8: state1[3:1] = 3'd2;
                       6'd9: state1[3:1] = 3'd2;
                       6'd10: state1[3:1] = 3'd2;
                       6'd11: state1[3:1] = 3'd1;
                       6'd12: state1[3:1] = 3'd4;
                       6'd13: state1[3:1] = 3'd4;
                       6'd14: state1[3:1] = 3'd4;
                       6'd15: state1[3:1] = 3'd4;
                       6'd16: state1[3:1] = 3'd2;
                       6'd17: state1[3:1] = 3'd2;
                       6'd18: state1[3:1] = 3'd2;
                       6'd19: state1[3:1] = 3'd1;
                       6'd20: state1[3:1] = 3'd4;
                       6'd21: state1[3:1] = 3'd4;
                       6'd22: state1[3:1] = 3'd4;
                       6'd23: state1[3:1] = 3'd4;
                       6'd24: state1[3:1] = 3'd2;
                       6'd25: state1[3:1] = 3'd2;
                       6'd26: state1[3:1] = 3'd2;
                       6'd27: state1[3:1] = 3'd5;
                       6'd28: state1[3:1] = 3'd4;
                       6'd29: state1[3:1] = 3'd4;
                       6'd30: state1[3:1] = 3'd4;
                       6'd31: state1[3:1] = 3'd4;
                       6'd32: state1[3:1] = 3'd2;
                       6'd33: state1[3:1] = 3'd2;
                       6'd34: state1[3:1] = 3'd2;
                       6'd35: state1[3:1] = 3'd3;
                       6'd36: state1[3:1] = 3'd4;
                       6'd37: state1[3:1] = 3'd4;
                       6'd38: state1[3:1] = 3'd4;
                       6'd39: state1[3:1] = 3'd4;
                       6'd40: state1[3:1] = 3'd2;
                       6'd41: state1[3:1] = 3'd2;
                       6'd42: state1[3:1] = 3'd2;
                       6'd43: state1[3:1] = 3'd3;
                       6'd44: state1[3:1] = 3'd4;
                       6'd45: state1[3:1] = 3'd4;
                       6'd46: state1[3:1] = 3'd4;
                       6'd47: state1[3:1] = 3'd4;
                       6'd48: state1[3:1] = 3'd2;
                       6'd49: state1[3:1] = 3'd2;
                       6'd50: state1[3:1] = 3'd2;
            6'd51: state1[3:1] = 3'd3;
            6'd52: state1[3:1] = 3'd4;
            6'd53: state1[3:1] = 3'd4;
            6'd54: state1[3:1] = 3'd4;
            6'd55: state1[3:1] = 3'd4;
            6'd56: state1[3:1] = 3'd2;
            6'd57: state1[3:1] = 3'd2;
            6'd58: state1[3:1] = 3'd2;
            6'd59: state1[3:1] = 3'd3;
            6'd60: state1[3:1] = 3'd4;
            6'd61: state1[3:1] = 3'd4;
            6'd62: state1[3:1] = 3'd4;
            6'd63: state1[3:1] = 3'd4;
            endcase

    end
    else
    if( (inb1[1:0] == 2'b01) & (state1[0] == 1'b1) )
            state1[4] = 1'b1;
    else
    if ( inb1[1:0] != 2'b01)
            state1[4] = 1'b0;

if(grant[1] == 1'b1)
begin
    if ( (state2[0] == 1'b1) & (state2[4] == 1'b1) )
            state2[0] = 1'b0;
end
else
    if( (inb2[1:0] == 2'b11) & (state2[0] == 1'b0) )
    begin
            state2[0] = 1'b1;
            state2[4] = 1'b0;

            case(inb2[7:2])
            6'd0: state2[3:1] = 3'd2;
            6'd1: state2[3:1] = 3'd2;
            6'd2: state2[3:1] = 3'd2;
            6'd3: state2[3:1] = 3'd1;
            6'd4: state2[3:1] = 3'd4;
            6'd5: state2[3:1] = 3'd4;
            6'd6: state2[3:1] = 3'd4;
            6'd7: state2[3:1] = 3'd4;
            6'd8: state2[3:1] = 3'd2;
            6'd9: state2[3:1] = 3'd2;
            6'd10: state2[3:1] = 3'd2;
            6'd11: state2[3:1] = 3'd1;
            6'd12: state2[3:1] = 3'd4;
            6'd13: state2[3:1] = 3'd4;
            6'd14: state2[3:1] = 3'd4;
            6'd15: state2[3:1] = 3'd4;
            6'd16: state2[3:1] = 3'd2;
            6'd17: state2[3:1] = 3'd2;
            6'd18: state2[3:1] = 3'd2;
            6'd19: state2[3:1] = 3'd1;
            6'd20: state2[3:1] = 3'd4;
            6'd21: state2[3:1] = 3'd4;
            6'd22: state2[3:1] = 3'd4;
            6'd23: state2[3:1] = 3'd4;
            6'd24: state2[3:1] = 3'd2;
            6'd25: state2[3:1] = 3'd2;
            6'd26: state2[3:1] = 3'd2;
            6'd27: state2[3:1] = 3'd5;
            6'd28: state2[3:1] = 3'd4;
            6'd29: state2[3:1] = 3'd4;
            6'd30: state2[3:1] = 3'd4;
            6'd31: state2[3:1] = 3'd4;
            6'd32: state2[3:1] = 3'd2;
            6'd33: state2[3:1] = 3'd2;
            6'd34: state2[3:1] = 3'd2;
            6'd35: state2[3:1] = 3'd3;
            6'd36: state2[3:1] = 3'd4;
            6'd37: state2[3:1] = 3'd4;
            6'd38: state2[3:1] = 3'd4;
            6'd39: state2[3:1] = 3'd4;
            6'd40: state2[3:1] = 3'd2;
            6'd41: state2[3:1] = 3'd2;
            6'd42: state2[3:1] = 3'd2;
            6'd43: state2[3:1] = 3'd3;
            6'd44: state2[3:1] = 3'd4;
            6'd45: state2[3:1] = 3'd4;
            6'd46: state2[3:1] = 3'd4;
            6'd47: state2[3:1] = 3'd4;
            6'd48: state2[3:1] = 3'd2;
            6'd49: state2[3:1] = 3'd2;
            6'd50: state2[3:1] = 3'd2;
            6'd51: state2[3:1] = 3'd3;
            6'd52: state2[3:1] = 3'd4;
            6'd53: state2[3:1] = 3'd4;
            6'd54: state2[3:1] = 3'd4;
            6'd55: state2[3:1] = 3'd4;
            6'd56: state2[3:1] = 3'd2;
            6'd57: state2[3:1] = 3'd2;
            6'd58: state2[3:1] = 3'd2;
            6'd59: state2[3:1] = 3'd3;
            6'd60: state2[3:1] = 3'd4;
            6'd61: state2[3:1] = 3'd4;
            6'd62: state2[3:1] = 3'd4;
            6'd63: state2[3:1] = 3'd4;
            endcase
    end
    else
    if( (inb2[1:0] == 2'b01) & (state2[0] == 1'b1) )
            state2[4] = 1'b1;
    else
    if ( inb2[1:0] != 2'b01)
            state2[4] = 1'b0;

if(grant[2] == 1'b1)
begin
    if ( (state3[0] == 1'b1) & (state3[4] == 1'b1) )
            state3[0] = 1'b0;
end
else
    if( (inb3[1:0] == 2'b11) & (state3[0] == 1'b0) )
    begin
            state3[0] = 1'b1;
            state3[4] = 1'b0;

            case(inb3[7:2])
            6'd0: state3[3:1] = 3'd2;
            6'd1: state3[3:1] = 3'd2;
            6'd2: state3[3:1] = 3'd2;
            6'd3: state3[3:1] = 3'd1;
            6'd4: state3[3:1] = 3'd4;
            6'd5: state3[3:1] = 3'd4;
            6'd6: state3[3:1] = 3'd4;
            6'd7: state3[3:1] = 3'd4;
            6'd8: state3[3:1] = 3'd2;
            6'd9: state3[3:1] = 3'd2;
            6'd10: state3[3:1] = 3'd2;
            6'd11: state3[3:1] = 3'd1;
            6'd12: state3[3:1] = 3'd4;
            6'd13: state3[3:1] = 3'd4;
            6'd14: state3[3:1] = 3'd4;
            6'd15: state3[3:1] = 3'd4;
            6'd16: state3[3:1] = 3'd2;
            6'd17: state3[3:1] = 3'd2;
            6'd18: state3[3:1] = 3'd2;
            6'd19: state3[3:1] = 3'd1;
            6'd20: state3[3:1] = 3'd4;
            6'd21: state3[3:1] = 3'd4;
            6'd22: state3[3:1] = 3'd4;
            6'd23: state3[3:1] = 3'd4;
            6'd24: state3[3:1] = 3'd2;
            6'd25: state3[3:1] = 3'd2;
            6'd26: state3[3:1] = 3'd2;
            6'd27: state3[3:1] = 3'd5;
            6'd28: state3[3:1] = 3'd4;
            6'd29: state3[3:1] = 3'd4;
            6'd30: state3[3:1] = 3'd4;
            6'd31: state3[3:1] = 3'd4;
            6'd32: state3[3:1] = 3'd2;
            6'd33: state3[3:1] = 3'd2;
            6'd34: state3[3:1] = 3'd2;
            6'd35: state3[3:1] = 3'd3;
            6'd36: state3[3:1] = 3'd4;
            6'd37: state3[3:1] = 3'd4;
            6'd38: state3[3:1] = 3'd4;
            6'd39: state3[3:1] = 3'd4;
            6'd40: state3[3:1] = 3'd2;
            6'd41: state3[3:1] = 3'd2;
            6'd42: state3[3:1] = 3'd2;
            6'd43: state3[3:1] = 3'd3;
            6'd44: state3[3:1] = 3'd4;
            6'd45: state3[3:1] = 3'd4;
            6'd46: state3[3:1] = 3'd4;
            6'd47: state3[3:1] = 3'd4;
            6'd48: state3[3:1] = 3'd2;
            6'd49: state3[3:1] = 3'd2;
            6'd50: state3[3:1] = 3'd2;
            6'd51: state3[3:1] = 3'd3;
            6'd52: state3[3:1] = 3'd4;
            6'd53: state3[3:1] = 3'd4;
            6'd54: state3[3:1] = 3'd4;
            6'd55: state3[3:1] = 3'd4;
            6'd56: state3[3:1] = 3'd2;
            6'd57: state3[3:1] = 3'd2;
            6'd58: state3[3:1] = 3'd2;
            6'd59: state3[3:1] = 3'd3;
            6'd60: state3[3:1] = 3'd4;
            6'd61: state3[3:1] = 3'd4;
            6'd62: state3[3:1] = 3'd4;
            6'd63: state3[3:1] = 3'd4;
            endcase
    end
    else
    if( (inb3[1:0] == 2'b01) & (state3[0] == 1'b1) )
            state3[4] = 1'b1;
    else
    if ( inb3[1:0] != 2'b01)
            state3[4] = 1'b0;

if(grant[3] == 1'b1)
begin
    if ( (state4[0] == 1'b1) & (state4[4] == 1'b1) )
            state4[0] = 1'b0;
end
else
    if( (inb4[1:0] == 2'b11) & (state4[0] == 1'b0) )
    begin
            state4[0] = 1'b1;
            state4[4] = 1'b0;

            case(inb4[7:2])
            6'd0: state4[3:1] = 3'd2;
            6'd1: state4[3:1] = 3'd2;
            6'd2: state4[3:1] = 3'd2;
            6'd3: state4[3:1] = 3'd1;
            6'd4: state4[3:1] = 3'd4;
```

```
                6'd5: state4[3:1] = 3'd4;
                6'd6: state4[3:1] = 3'd4;
                6'd7: state4[3:1] = 3'd4;
                6'd8: state4[3:1] = 3'd2;
                6'd9: state4[3:1] = 3'd2;
                6'd10: state4[3:1] = 3'd2;
                6'd11: state4[3:1] = 3'd1;
                6'd12: state4[3:1] = 3'd4;
                6'd13: state4[3:1] = 3'd4;
                6'd14: state4[3:1] = 3'd4;
                6'd15: state4[3:1] = 3'd4;
                6'd16: state4[3:1] = 3'd2;
                6'd17: state4[3:1] = 3'd2;
                6'd18: state4[3:1] = 3'd2;
                6'd19: state4[3:1] = 3'd1;
                6'd20: state4[3:1] = 3'd4;
                6'd21: state4[3:1] = 3'd4;
                6'd22: state4[3:1] = 3'd4;
                6'd23: state4[3:1] = 3'd4;
                6'd24: state4[3:1] = 3'd2;
                6'd25: state4[3:1] = 3'd2;
                6'd26: state4[3:1] = 3'd2;
                6'd27: state4[3:1] = 3'd5;
                6'd28: state4[3:1] = 3'd4;
                6'd29: state4[3:1] = 3'd4;
                6'd30: state4[3:1] = 3'd4;
                6'd31: state4[3:1] = 3'd4;
                6'd32: state4[3:1] = 3'd2;
                6'd33: state4[3:1] = 3'd2;
                6'd34: state4[3:1] = 3'd2;
                6'd35: state4[3:1] = 3'd3;
                6'd36: state4[3:1] = 3'd4;
                6'd37: state4[3:1] = 3'd4;
                6'd38: state4[3:1] = 3'd4;
                6'd39: state4[3:1] = 3'd4;
                6'd40: state4[3:1] = 3'd2;
                6'd41: state4[3:1] = 3'd2;
                6'd42: state4[3:1] = 3'd2;
                6'd43: state4[3:1] = 3'd3;
                6'd44: state4[3:1] = 3'd4;
                6'd45: state4[3:1] = 3'd4;
                6'd46: state4[3:1] = 3'd4;
                6'd47: state4[3:1] = 3'd4;
                6'd48: state4[3:1] = 3'd2;
                6'd49: state4[3:1] = 3'd2;
                6'd50: state4[3:1] = 3'd2;
                6'd51: state4[3:1] = 3'd3;
                6'd52: state4[3:1] = 3'd4;
                6'd53: state4[3:1] = 3'd4;
                6'd54: state4[3:1] = 3'd4;
                6'd55: state4[3:1] = 3'd4;
                6'd56: state4[3:1] = 3'd2;
                6'd57: state4[3:1] = 3'd2;
                6'd58: state4[3:1] = 3'd2;
                6'd59: state4[3:1] = 3'd3;
                6'd60: state4[3:1] = 3'd4;
                6'd61: state4[3:1] = 3'd4;
                6'd62: state4[3:1] = 3'd4;
                6'd63: state4[3:1] = 3'd4;
                endcase
        end
        else
        if( (inb4[1:0] == 2'b01) & (state4[0] == 1'b1) )
                state4[4] = 1'b1;
        else
        if ( inb4[1:0] != 2'b01)
                state4[4] = 1'b0;

if(grant[4] == 1'b1)
begin
```

```
        if ( (state5[0] == 1'b1) & (state5[4] == 1'b1) )
                state5[0] = 1'b0;
end
else
        if( (inb5[1:0] == 2'b11) & (state5[0] == 1'b0) )
        begin
                state5[0] = 1'b1;
                state5[4] = 1'b0;

                case(inb5[7:2])
                6'd0: state5[3:1] = 3'd2;
                6'd1: state5[3:1] = 3'd2;
                6'd2: state5[3:1] = 3'd2;
                6'd3: state5[3:1] = 3'd1;
                6'd4: state5[3:1] = 3'd4;
                6'd5: state5[3:1] = 3'd4;
                6'd6: state5[3:1] = 3'd4;
                6'd7: state5[3:1] = 3'd4;
                6'd8: state5[3:1] = 3'd2;
                6'd9: state5[3:1] = 3'd2;
                6'd10: state5[3:1] = 3'd2;
                6'd11: state5[3:1] = 3'd1;
                6'd12: state5[3:1] = 3'd4;
                6'd13: state5[3:1] = 3'd4;
                6'd14: state5[3:1] = 3'd4;
                6'd15: state5[3:1] = 3'd4;
                6'd16: state5[3:1] = 3'd2;
                6'd17: state5[3:1] = 3'd2;
                6'd18: state5[3:1] = 3'd2;
                6'd19: state5[3:1] = 3'd1;
                6'd20: state5[3:1] = 3'd4;
                6'd21: state5[3:1] = 3'd4;
                6'd22: state5[3:1] = 3'd4;
                6'd23: state5[3:1] = 3'd4;
                6'd24: state5[3:1] = 3'd2;
                6'd25: state5[3:1] = 3'd2;
                6'd26: state5[3:1] = 3'd2;
                6'd27: state5[3:1] = 3'd5;
                6'd28: state5[3:1] = 3'd4;
                6'd29: state5[3:1] = 3'd4;
                6'd30: state5[3:1] = 3'd4;
                6'd31: state5[3:1] = 3'd4;
                6'd32: state5[3:1] = 3'd2;
                6'd33: state5[3:1] = 3'd2;
                6'd34: state5[3:1] = 3'd2;
                6'd35: state5[3:1] = 3'd3;
                6'd36: state5[3:1] = 3'd4;
                6'd37: state5[3:1] = 3'd4;
                6'd38: state5[3:1] = 3'd4;
                6'd39: state5[3:1] = 3'd4;
                6'd40: state5[3:1] = 3'd2;
                6'd41: state5[3:1] = 3'd2;
                6'd42: state5[3:1] = 3'd2;
                6'd43: state5[3:1] = 3'd3;
                6'd44: state5[3:1] = 3'd4;
                6'd45: state5[3:1] = 3'd4;
                6'd46: state5[3:1] = 3'd4;
                6'd47: state5[3:1] = 3'd4;
                6'd48: state5[3:1] = 3'd2;
                6'd49: state5[3:1] = 3'd2;
                6'd50: state5[3:1] = 3'd2;
                6'd51: state5[3:1] = 3'd3;
                6'd52: state5[3:1] = 3'd4;
                6'd53: state5[3:1] = 3'd4;
                6'd54: state5[3:1] = 3'd4;
                6'd55: state5[3:1] = 3'd4;
                6'd56: state5[3:1] = 3'd2;
                6'd57: state5[3:1] = 3'd2;
                6'd58: state5[3:1] = 3'd2;
                6'd59: state5[3:1] = 3'd3;
                6'd60: state5[3:1] = 3'd4;
                6'd61: state5[3:1] = 3'd4;
                6'd62: state5[3:1] = 3'd4;
                6'd63: state5[3:1] = 3'd4;
                endcase
        end
        else
        if( (inb5[1:0] == 2'b01) & (state5[0] == 1'b1) )
                state5[4] = 1'b1;
        else
        if ( inb5[1:0] != 2'b01)
                state5[4] = 1'b0;



        if ( (state1[0]==1) )
                allow[0] = 1;
        else
                allow[0] = 0;

        if ( (state2[0]==1) & (state2[3:1] != state1[3:1]) )
                allow[1] = 1;
        else
                allow[1] = 0;

        if ( (state3[0]==1) & (state3[3:1] != state2[3:1]) & (state3[3:1] != state1[3:1])
)
                allow[2] = 1;
        else
                allow[2] = 0;

        if ( (state4[0]==1) & (state4[3:1] != state3[3:1]) & (state4[3:1] != state2[3:1])
& (state4[3:1] != state1[3:1]) )
                allow[3] = 1;
        else
                allow[3] = 0;

        if ( (state5[0]==1) & (state5[3:1] != state4[3:1]) & (state5[3:1] != state3[3:1])
& (state5[3:1] != state2[3:1]) & (state5[3:1] != state1[3:1]) )
                allow[4] = 1;
        else
                allow[4] = 0;

// generate xbar select line encoder logic using grant signal

        if (grant[0])
                case(state1[3:1])
                        3'd1: sel1 = `ports'b00001;
                        3'd2: sel1 = `ports'b00010;
                        3'd3: sel1 = `ports'b00100;
                        3'd4: sel1 = `ports'b01000;
                        3'd5: sel1 = `ports'b10000;
                        default: sel1 = sel1;
                endcase

        if (grant[1])
                case(state2[3:1])
                        3'd1: sel2 = `ports'b00001;
                        3'd2: sel2 = `ports'b00010;
                        3'd3: sel2 = `ports'b00100;
                        3'd4: sel2 = `ports'b01000;
                        3'd5: sel2 = `ports'b10000;
                        default: sel2 = sel2;
                endcase

        if (grant[2])
                case(state3[3:1])
                        3'd1: sel3 = `ports'b00001;
                        3'd2: sel3 = `ports'b00010;
                        3'd3: sel3 = `ports'b00100;
                        3'd4: sel3 = `ports'b01000;
                        3'd5: sel3 = `ports'b10000;
                        default: sel3 = sel3;
                endcase

        if (grant[3])
                case(state4[3:1])
                        3'd1: sel4 = `ports'b00001;
                        3'd2: sel4 = `ports'b00010;
                        3'd3: sel4 = `ports'b00100;
                        3'd4: sel4 = `ports'b01000;
                        3'd5: sel4 = `ports'b10000;
                        default: sel4 = sel4;
                endcase

        if (grant[4])
                case(state5[3:1])
                        3'd1: sel5 = `ports'b00001;
                        3'd2: sel5 = `ports'b00010;
                        3'd3: sel5 = `ports'b00100;
                        3'd4: sel5 = `ports'b01000;
                        3'd5: sel5 = `ports'b10000;
                        default: sel5 = sel5;
                endcase

        end // if not reset
    end // end always


// instance crossbar (in this case its a matrix crossbar)

    matrix5x5_xbar_128b xbar1(clk, inb1, inb2, inb3, inb4, inb5, sel1, sel2, sel3, sel4,
sel5, out1, out2, out3, out4, out5);

// instance arbiter

    round_robin5 arbiter(clk, rst, req, grant, credit);


// instance 5 input port buffers

    rf_2p_adv port1(
            QA1,
            clk,
            CENA1,
            rp1,
            clk,
            CENB1,
            wp1,
            in1,
            EMAA,
            EMAB
            );

    rf_2p_adv port2(
            QA2,
            clk,
            CENA2,
            rp2,
            clk,
            CENB2,
            wp2,
            in2,
            EMAA,
            EMAB
            );

    rf_2p_adv port3(
            QA3,
            clk,
                CENA3,
                rp3,
                clk,
                CENB3,
                wp3,
                in3,
                EMAA,
                EMAB
                );

        rf_2p_adv port4(
                QA4,
                clk,
                CENA4,
                rp4,
                clk,
                CENB4,
                wp4,
                in4,
                EMAA,
                EMAB
                );

        rf_2p_adv port5(
                QA5,
                clk,
                CENA5,
                rp5,
                clk,
                CENB5,
                wp5,
                in5,
                EMAA,
                EMAB
                );

// generate request signal, which is the logical OR of the Valid Bit(state*[0])
// and the Tail Detection Bit(state*[4]) AND with allow

        assign req = {((state5[0] | state5[4]) & allow[4]), ((state4[0] | state4[4]) & allow[3]),
((state3[0] | state3[4]) & allow[2]), ((state2[0] | state2[4]) & allow[1]), ((state1[0] |
state1[4]) & allow[0])};

        assign CENB1 = ~( (valid1) & (~full1));
        assign CENA1 = ~( (valid1 | full1) & (~empty1));

        assign CENB2 = ~( (valid2) & (~full2));
        assign CENA2 = ~( (valid2 | full2) & (~empty2));

        assign CENB3 = ~( (valid3) & (~full3));
        assign CENA3 = ~( (valid3 | full3) & (~empty3));

        assign CENB4 = ~( (valid4) & (~full4));
        assign CENA4 = ~( (valid4 | full4) & (~empty4));

        assign CENB5 = ~( (valid5) & (~full5));
        assign CENA5 = ~( (valid5 | full5) & (~empty5));


endmodule

// end of router module
```

8. APPENDIX C: 5x5 ROUTER TEST BENCH CODE

```
`timescale 100ps/10ps

// Router 5x5 128b TESTBENCH Matrix
// ********************************

`define portwidth 128
`define router_number 27
`define ports 5

module router5x5_128b_tb;

        reg clk, rst;
        reg [`portwidth-1:0] in1, in2, in3, in4, in5;
        reg valid1, valid2, valid3, valid4, valid5;
        reg [3:0] wp1, wp2, wp3, wp4, wp5, rp1, rp2, rp3, rp4, rp5;

        wire [`portwidth-1:0] out1, out2, out3, out4, out5;

        wire CENA1, CENA2, CENA3, CENA4, CENA5, CENB1, CENB2, CENB3, CENB4, CENB5;
        wire full1, full2, full3, full4, full5;
        reg [6:0] cc1, cc2, cc3, cc4, cc5;

        wire [`portwidth-1:0] inb1, inb2, inb3, inb4, inb5;
        wire [4:0] state1, state2, state3, state4, state5;
        wire [4:0] sel1, sel2, sel3, sel4, sel5;
        wire [4:0]  req, grant, credit;
        wire [3:0] space1, space2, space3, space4, space5;
        wire empty1, empty2, empty3, empty4, empty5;

router5x5_128b I1(clk, rst, in1, in2, in3, in4, in5, out1, out2, out3, out4, out5, grant, CENA1,
CENA2, CENA3, CENA4, CENA5, rp1, rp2, rp3, rp4, rp5, CENB1, CENB2, CENB3, CENB4, CENB5, wp1, wp2,
wp3, wp4, wp5, valid1, valid2, valid3, valid4, valid5, full1, full2, full3, full4, full5, inb1,
inb2, inb3, inb4, inb5, state1, state2, state3, state4, state5, sel1, sel2, sel3, sel4, sel5,
req, credit, space1, space2, space3, space4, space5, empty1, empty2, empty3, empty4, empty5);


        always #50
         begin
                clk = ~clk;
        end

        initial begin
                $stop;

                in1 = `portwidth'b0;
                in2 = `portwidth'b0;
                in3 = `portwidth'b0;
                in4 = `portwidth'b0;
                in5 = `portwidth'b0;

                clk = 1'b1;
                rst = 1'b1;

                valid1 = 0;
                valid2 = 0;
                valid3 = 0;
                valid4 = 0;
                valid5 = 0;

                rp1 = 4'd15;
                wp1 = 4'd0;

                rp2 = 4'd15;
                wp2 = 4'd0;

                rp3 = 4'd15;
                wp3 = 4'd0;

                rp4 = 4'd15;
                wp4 = 4'd0;

                rp5 = 4'd15;
                wp5 = 4'd0;

                        cc1 = 7'b00;
                        cc2 = 7'b00;
                        cc3 = 7'b00;
                        cc4 = 7'b00;
                        cc5 = 7'b00;

#150
                rst = 1'b0;



#100





                if(CENB1 == 0) wp1 = wp1 + 1;
                if(CENA1 == 0)  rp1 = rp1 + 1;

                if(CENB2 == 0) wp2 = wp2 + 1;
                if(CENA2 == 0)  rp2 = rp2 + 1;

                if(CENB3 == 0) wp3 = wp3 + 1;
                if(CENA3 == 0)  rp3 = rp3 + 1;

                if(CENB4 == 0) wp4 = wp4 + 1;
                if(CENA4 == 0)  rp4 = rp4 + 1;

                if(CENB5 == 0) wp5 = wp5 + 1;
                if(CENA5 == 0)  rp5 = rp5 + 1;


// start here

#100

                if(grant[0] == 1'b1)
                        cc1 = cc1 + 7'd1;
                if(grant[1] == 1'b1)
                        cc2 = cc2 + 7'd1;
                if(grant[2] == 1'b1)
                        cc3 = cc3 + 7'd1;
                if(grant[3] == 1'b1)
                        cc4 = cc4 + 7'd1;
                if(grant[4] == 1'b1)
                        cc5 = cc5 + 7'd1;


                if(CENB1 == 0) wp1 = wp1 + 1;
                if(CENA1 == 0)  rp1 = rp1 + 1;

                if(CENB2 == 0) wp2 = wp2 + 1;
                if(CENA2 == 0)  rp2 = rp2 + 1;

                if(CENB3 == 0) wp3 = wp3 + 1;
                if(CENA3 == 0)  rp3 = rp3 + 1;

                if(CENB4 == 0) wp4 = wp4 + 1;
                if(CENA4 == 0)  rp4 = rp4 + 1;

                if(CENB5 == 0) wp5 = wp5 + 1;
                if(CENA5 == 0)  rp5 = rp5 + 1;

                if(full1 != 1'b1)
                valid1 = 1;
                else
                valid1 = 0;

                if(full2 != 1'b1)
                valid2 = 1;
                else
                valid2 = 0;

                if(full3 != 1'b1)
                valid3 = 1;
                else
                valid3 = 0;

                if(full4 != 1'b1)
                valid4 = 1;
                else
                valid4 = 0;

                if(full5 != 1'b1)
                valid5 = 1;
                else
                valid5 = 0;

//new port start
if (cc1 == 7'd0)
     in1 = 128'h6414251c7106255459441368740517 0f;
else if (cc1 == 7'd1)
     in1 = 128'h710050707134286625 0d582e1466660e;
else if (cc1 == 7'd2)
     in1 = 128'h2e5b4f5612216c47614f2f1d1413640e;
else if (cc1 == 7'd3)
     in1 = 128'h72697748770537231 32b46012e143b0d;
else if (cc1 == 7'd4)
     in1 = 128'h5a31112e1622041a6f622c4677382f0f;
else if (cc1 == 7'd5)
     in1 = 128'h0f4210164545640f4e4d07301320750e;
else if (cc1 == 7'd6)
     in1 = 128'h7520304e6e776e447169493b243c5a0e;
else if (cc1 == 7'd7)
     in1 = 128'h11526053603425482c1c375b0c67350d;
else if (cc1 == 7'd8)
     in1 = 128'h14272113502856340c662c440a2f640f;
else if (cc1 == 7'd9)
     in1 = 128'h6247426a643a351c24081d601a1a4d0e;
else if (cc1 == 7'd10)
     in1 = 128'h2f4008121702693a2e70421c6c14180e;
else if (cc1 == 7'd11)
     in1 = 128'h4b4d0b56400f744a5703474511211f0d;
else if (cc1 == 7'd12)
     in1 = 128'h194e6f2a2b16120c5353277135454a0f;
else if (cc1 == 7'd13)
     in1 = 128'h2a516a76513858440452335744 3a310e;
else if (cc1 == 7'd14)
     in1 = 128'h640216070c6b7145236b65665f1d190e;
else if (cc1 == 7'd15)
     in1 = 128'h7158576817004e1021065e6a7418640d;
else if (cc1 == 7'd16)
     in1 = 128'h6e2a541a306e76057401760e2d4b100f;
else if (cc1 == 7'd17)
     in1 = 128'h57286d0d184f3c455218380e1832010e;
else if (cc1 == 7'd18)
     in1 = 128'h6d0c664b31485951593f1213082f620e;
else if (cc1 == 7'd19)
```

```
         in1 = 128'h1d3c3a2814040d68632a142e03094a0d;
    else if (cc1 == 7'd20)
         in1 = 128'h0944293a522455482c1a45301c5c1c0f;
    else if (cc1 == 7'd21)
         in1 = 128'h086311473b42105c545d1e231d10210e;
    else if (cc1 == 7'd22)
         in1 = 128'h2d2d42235063680237515777 4d57100e;
    else if (cc1 == 7'd23)
         in1 = 128'h0c0e535c6d664d176138261c3038130d;
    else if (cc1 == 7'd24)
         in1 = 128'h066b742370180d3974600e443075190f;
    else if (cc1 == 7'd25)
         in1 = 128'h57170c3f5361034e31114c687265690e;
    else if (cc1 == 7'd26)
         in1 = 128'h293948470c437643480267741f171b0e;
    else if (cc1 == 7'd27)
         in1 = 128'h4e59160c5d2e4d6c71015d0a4c1d3e0d;
    else if (cc1 == 7'd28)
         in1 = 128'h560a1d4c6432754a3b220d3c2e524c0f;
    else if (cc1 == 7'd29)
         in1 = 128'h3a276830226f2f63571e761e6e243a0e;
    else if (cc1 == 7'd30)
         in1 = 128'h17342a082f28356a71263736224d750e;
    else if (cc1 == 7'd31)
         in1 = 128'h52276234552052676874 6b5b4110000d;
    else if (cc1 == 7'd32)
         in1 = 128'h68725a020155102753603710 31275e0f;
    else if (cc1 == 7'd33)
         in1 = 128'h3b70412d370318080d0076670e680a0e;
    else if (cc1 == 7'd34)
         in1 = 128'h08663a1a1f0b226e517462714e4f380e;
    else if (cc1 == 7'd35)
         in1 = 128'h3c6807460f4140640e53046d5d0f2d0d;
    else if (cc1 == 7'd36)
         in1 = 128'h354152535441433813220217 1d29520f;
    else if (cc1 == 7'd37)
         in1 = 128'h0d454350340552460a1440090516590e;
    else if (cc1 == 7'd38)
         in1 = 128'h5a5407313d585f6c744f184633261c0e;
    else if (cc1 == 7'd39)
         in1 = 128'h3c45153d646468760642454 33f74180d;
    else if (cc1 == 7'd40)
         in1 = 128'h5f2245761b1e066d0267462c2566100f;
    else if (cc1 == 7'd41)
         in1 = 128'h123414766077595929 6a35513f20460e;
    else if (cc1 == 7'd42)
         in1 = 128'h07083665726d0f74702c44362c24430e;
    else if (cc1 == 7'd43)
         in1 = 128'h203b015f082b2f48110a0a293628470d;
    else if (cc1 == 7'd44)
         in1 = 128'h071d725e5333053c2233361261313f0f;
    else if (cc1 == 7'd45)
         in1 = 128'h584a44310532160 96f652510446c490e;
    else if (cc1 == 7'd46)
         in1 = 128'h395c6d29405f0f582e6d5167144f240e;
    else if (cc1 == 7'd47)
         in1 = 128'h165731202e1f335a007033736e1c3c0d;
    else if (cc1 == 7'd48)
         in1 = 128'h680f15047140124c1c50025f6049760f;
    else if (cc1 == 7'd49)
         in1 = 128'h200a3b440f3976746b57224d356a520e;
    else if (cc1 == 7'd50)
         in1 = 128'h082050195511213 61a26333a6a6a1e0e;
    else if (cc1 == 7'd51)
         in1 = 128'h53195458496a5842035a0c206656650d;
    else if (cc1 == 7'd52)
         in1 = 128'h017110291d445463110c4d644b23630f;
    else if (cc1 == 7'd53)
         in1 = 128'h1d174e07504f0c28637150692f51630e;
    else if (cc1 == 7'd54)
```

```
     in1 = 128'h1c43046d00196e3b5e106d694a1c0f0e;
else if (cc1 == 7'd55)
     in1 = 128'h1d6348261f1e0e1c395e5700500b250d;
else if (cc1 == 7'd56)
     in1 = 128'h37720f5d4c755615280b281f2767240f;
else if (cc1 == 7'd57)
     in1 = 128'h6373046936583d73610e0c042537740e;
else if (cc1 == 7'd58)
     in1 = 128'h1648002824721c3d095c67135c30570e;
else if (cc1 == 7'd59)
     in1 = 128'h5865704008061a2c356b47310e23090d;
else if (cc1 == 7'd60)
     in1 = 128'h6a0b0b6f511f202648600202091468 0f;
else if (cc1 == 7'd61)
     in1 = 128'h163970206563011c142021586707500e;
else if (cc1 == 7'd62)
     in1 = 128'h75005f1017005f3848214040611a570e;
else if (cc1 == 7'd63)
     in1 = 128'h69510722395716586f052d2e2c4f5d0d;
else if (cc1 == 7'd64)
     in1 = 128'h0177244f5c46144201346f122a2c390f;
else if (cc1 == 7'd65)
     in1 = 128'h58550a302b64724c4711642e3925390e;
else if (cc1 == 7'd66)
     in1 = 128'h23514c7318171d4b774e48185a703c0e;
else if (cc1 == 7'd67)
     in1 = 128'h77751749007119211e0e68027123380d;
else if (cc1 == 7'd68)
     in1 = 128'h3a6654217735773e232f3d660b135e0f;
else if (cc1 == 7'd69)
     in1 = 128'h2f722a3f55175c3113003931181f080e;
else if (cc1 == 7'd70)
     in1 = 128'h221d274a61001d083426094f1656690e;
else if (cc1 == 7'd71)
     in1 = 128'h2a0e095b2e63403c451956243f3b0c0d;
else if (cc1 == 7'd72)
     in1 = 128'h4a75031d670265097264372622343 20f;
else if (cc1 == 7'd73)
     in1 = 128'h160e5d6b5c276229566972143928670e;
else if (cc1 == 7'd74)
     in1 = 128'h582722604a5f2071216c4414262c1d0e;
else if (cc1 == 7'd75)
     in1 = 128'h08731c4c1b69626b473b1b4748211f0d;
else if (cc1 == 7'd76)
     in1 = 128'h6d005d384e2b06476e205904575e390f;
else if (cc1 == 7'd77)
     in1 = 128'h64071a00380e09326f591d6f05363c0e;
else if (cc1 == 7'd78)
     in1 = 128'h2c644915284b51492c662b3f333d600e;
else if (cc1 == 7'd79)
     in1 = 128'h756150272827022 44a2f5d427551560d;
else if (cc1 == 7'd80)
     in1 = 128'h693a34486055314b2e70734a5830360f;
else if (cc1 == 7'd81)
     in1 = 128'h5d1d5c486a7704081f3a2f430f41550e;
else if (cc1 == 7'd82)
     in1 = 128'h316e0f077567683f1c340b4f4758150e;
else if (cc1 == 7'd83)
     in1 = 128'h3e5f721234695d57704d72710061600d;
else if (cc1 == 7'd84)
     in1 = 128'h5d5843336c616b0d2f3458074e26400f;
else if (cc1 == 7'd85)
     in1 = 128'h7551060e57506d382d2f341c125c730e;
else if (cc1 == 7'd86)
     in1 = 128'h660911314510096d3168463f554e380e;
else if (cc1 == 7'd87)
     in1 = 128'h1335616a5311414647385d0f6e323a0d;
else if (cc1 == 7'd88)
     in1 = 128'h460269774b38003a6926464344263d0f;
else if (cc1 == 7'd89)
```

```
     in1 = 128'h1046352d2e2d2a77385c022e503b6e0e;
else if (cc1 == 7'd90)
     in1 = 128'h2c29513c43091a29556f0e086866050e;
else if (cc1 == 7'd91)
     in1 = 128'h0739712f733a0f66200221636752540d;
else if (cc1 == 7'd92)
     in1 = 128'h16551f2673130971060a735d7215140f;
else if (cc1 == 7'd93)
     in1 = 128'h1351595f314526693b67654d24406e0e;
else if (cc1 == 7'd94)
     in1 = 128'h0c11676617413002372d1d2949114c0e;
else if (cc1 == 7'd95)
     in1 = 128'h540144742516144c3916322e6c4d3a0d;
else if (cc1 == 7'd96)
     in1 = 128'h482d1c213c141d322659731c7269110f;
else if (cc1 == 7'd97)
     in1 = 128'h195f351d64603c00183f06246c754d0e;
else if (cc1 == 7'd98)
     in1 = 128'h380c0800710c6e52180709725e63130e;
else if (cc1 == 7'd99)
     in1 = 128'h1c6a4a5b545e7433000f020665591e0d;
else if (cc1 == 7'd100)
     in1 = 128'h5f02036211550b675e73281e4e0f5b0f;
else if (cc1 == 7'd101)
     in1 = 128'h15476875182e606c5a20111f281f100e;
else if (cc1 == 7'd102)
     in1 = 128'h203d0a66342b4f02075f5e3261114b0e;
else if (cc1 == 7'd103)
     in1 = 128'h353757322f43616645476a756d10740d;
else if (cc1 == 7'd104)
     in1 = 128'h4b220d3b532727386f633d5428081b0f;
else if (cc1 == 7'd105)
     in1 = 128'h230614196712325a3b0b48370c34480e;
else if (cc1 == 7'd106)
     in1 = 128'h741a20742f06711c07421b1306612c0e;
else if (cc1 == 7'd107)
     in1 = 128'h1d594b59764375163061560f41506a0d;
else if (cc1 == 7'd108)
     in1 = 128'h014c693567602072414a3a4452190f0f;
else if (cc1 == 7'd109)
     in1 = 128'h2e584f2853307050284b0b070c75070e;
else if (cc1 == 7'd110)
     in1 = 128'h3711500e4b12514a146f6b0f4e50670e;
else if (cc1 == 7'd111)
     in1 = 128'h2758504e42515d171b0834600706770d;
else if (cc1 == 7'd112)
     in1 = 128'h4970523a34451c1e24065d2d3361050f;
else if (cc1 == 7'd113)
     in1 = 128'h724b2737425e100a291b5036156e590e;
else if (cc1 == 7'd114)
     in1 = 128'h394f295b70394b2b72673668121f230e;
else if (cc1 == 7'd115)
     in1 = 128'h475d0b5d561f4b250b112f29045f420d;
else if (cc1 == 7'd116)
     in1 = 128'h41491f2a295f4f3c401d091b363c420f;
else if (cc1 == 7'd117)
     in1 = 128'h490e1a240873234969563 62f076d760e;
else if (cc1 == 7'd118)
     in1 = 128'h36161468396943326d445d5d1721690e;
else if (cc1 == 7'd119)
     in1 = 128'h372e19116707502c4a2f623d334f140d;
else if (cc1 == 7'd120)
     in1 = 128'h6e6d4f1f471b37165d59085a0f56150f;
else if (cc1 == 7'd121)
     in1 = 128'h1760636d012b10501a4c6512234e540e;
else if (cc1 == 7'd122)
     in1 = 128'h3a09062c7533321c4b270c6414430f0e;
else if (cc1 == 7'd123)
     in1 = 128'h20534c1e4e034108720b004a370a480d;
else if (cc1 == 7'd124)
```

```
     in1 = 128'h764b1228073a0f072355473c3d0b1c0f;
else if (cc1 == 7'd125)
     in1 = 128'h717233320a1a40651632452174 27340e;
else if (cc1 == 7'd126)
     in1 = 128'h2d34216e69601065623730537 6386b0e;
else if (cc1 == 7'd127)
     in1 = 128'h171b1f20065a566a752333334c5a080d;

//new port start
if (cc2 == 7'd0)
     in2 = 128'h5511250a2c1b443845327034463c3e0b;
else if (cc2 == 7'd1)
     in2 = 128'h04141d7263211222333a70446c68290a;
else if (cc2 == 7'd2)
     in2 = 128'h636b525b54306415535d34735065550a;
else if (cc2 == 7'd3)
     in2 = 128'h26606c643c4a384e605533211e483e09;
else if (cc2 == 7'd4)
     in2 = 128'h5a454c026d410e085e2f604d3a360a0b;
else if (cc2 == 7'd5)
     in2 = 128'h385b692b4b4a1f2c4a6325417416190a;
else if (cc2 == 7'd6)
     in2 = 128'h25214c245a3f662e2f521f6a0f71130a;
else if (cc2 == 7'd7)
     in2 = 128'h004c6a0b55356b492a2f3906041e4809;
else if (cc2 == 7'd8)
     in2 = 128'h486b741e355960234e2d72074c33610b;
else if (cc2 == 7'd9)
     in2 = 128'h5c5f5d06533636036c315d0d4d69340a;
else if (cc2 == 7'd10)
     in2 = 128'h194a5f66251a23713d3f174f3830540a;
else if (cc2 == 7'd11)
     in2 = 128'h2c1e0b6835144f0e5143105f71676709;
else if (cc2 == 7'd12)
     in2 = 128'h2c590203363056354f0b5f6f751a1b0b;
else if (cc2 == 7'd13)
     in2 = 128'h50562f041940461e21676b0110340b0a;
else if (cc2 == 7'd14)
     in2 = 128'h511d546f594b7325150f4b142115220a;
else if (cc2 == 7'd15)
     in2 = 128'h746f5f1453044a0e500e1c00003f5b09;
else if (cc2 == 7'd16)
     in2 = 128'h0c2248404b1e335a581129053a36680b;
else if (cc2 == 7'd17)
     in2 = 128'h0c09630f143c0d054d00715f261c6c0a;
else if (cc2 == 7'd18)
     in2 = 128'h382f6a430672120b4077176c285f2e0a;
else if (cc2 == 7'd19)
     in2 = 128'h225a081609136c3d56483c2b6f071509;
else if (cc2 == 7'd20)
     in2 = 128'h53480b4b755458771b305a4c722f510b;
else if (cc2 == 7'd21)
     in2 = 128'h1e2a5635061c51532c5a28301429650a;
else if (cc2 == 7'd22)
     in2 = 128'h314048363c7032134868452f470f480a;
else if (cc2 == 7'd23)
     in2 = 128'h6e6c4e2b2d22056c69524248375c4409;
else if (cc2 == 7'd24)
     in2 = 128'h0d437326216828613146 2f683c6c370b;
else if (cc2 == 7'd25)
     in2 = 128'h124e492962591f6c066b5f0f5127190a;
else if (cc2 == 7'd26)
     in2 = 128'h65512243581058 1b55674935152a710a;
else if (cc2 == 7'd27)
     in2 = 128'h713b62090b4d11515b1f331623054709;
else if (cc2 == 7'd28)
     in2 = 128'h614a5e6e26337359421d642f752c3c0b;
else if (cc2 == 7'd29)
     in2 = 128'h55317461340c6e2b4705454163085e0a;
else if (cc2 == 7'd30)
```

```
     in2 = 128'h2a3a20714f34171e566b313277754a0a;
else if (cc2 == 7'd31)
     in2 = 128'h43406c1c22492c421526392565523709;
else if (cc2 == 7'd32)
     in2 = 128'h3735002950302f55094f3f066a0d0a0b;
else if (cc2 == 7'd33)
     in2 = 128'h3714625f014a194e4128700b295e1a0a;
else if (cc2 == 7'd34)
     in2 = 128'h6007721469570c602a77285b2f466a0a;
else if (cc2 == 7'd35)
     in2 = 128'h4a497312566f270b2c43014e6a163e09;
else if (cc2 == 7'd36)
     in2 = 128'h6e0f0b01110d0526664e071037495a0b;
else if (cc2 == 7'd37)
     in2 = 128'h642868162c6f35023b56594109215c0a;
else if (cc2 == 7'd38)
     in2 = 128'h2f64007360225c6c5e59351c3c1d300a;
else if (cc2 == 7'd39)
     in2 = 128'h566347470e385a4856771353666f4a09;
else if (cc2 == 7'd40)
     in2 = 128'h3d056000663b493f092f75192c0a3f0b;
else if (cc2 == 7'd41)
     in2 = 128'h4e59047538772459223a283c4f5b4c0a;
else if (cc2 == 7'd42)
     in2 = 128'h086d2124030f7117046e7119102d600a;
else if (cc2 == 7'd43)
     in2 = 128'h307061266d2c5a162b5514406a6f2209;
else if (cc2 == 7'd44)
     in2 = 128'h6c4415486774067O395d281c264f710b;
else if (cc2 == 7'd45)
     in2 = 128'h0a0070443658484f0848160b712e080a;
else if (cc2 == 7'd46)
     in2 = 128'h683e773531205f54260f724b2c4b310a;
else if (cc2 == 7'd47)
     in2 = 128'h4a3623505f0f4e1c600a3f544c356609;
else if (cc2 == 7'd48)
     in2 = 128'h28125e694b266e092e0e332a513d750b;
else if (cc2 == 7'd49)
     in2 = 128'h490f5348690155011a3c45345b2e0c0a;
else if (cc2 == 7'd50)
     in2 = 128'h514c650205546d1818032d180735530a;
else if (cc2 == 7'd51)
     in2 = 128'h1225672424484a621b636f63333e2b09;
else if (cc2 == 7'd52)
     in2 = 128'h0d0f6828123415423b6828252303450b;
else if (cc2 == 7'd53)
     in2 = 128'h17560254081812400b0d28000d195d0a;
else if (cc2 == 7'd54)
     in2 = 128'h3b1819485c3d1f62404e3c38471e670a;
else if (cc2 == 7'd55)
     in2 = 128'h1256303d0737565d4564596f04750c09;
else if (cc2 == 7'd56)
     in2 = 128'h61384e217507423b0b200e5b56176b0b;
else if (cc2 == 7'd57)
     in2 = 128'h2b142f0b3d194e1b6e171d765f4b300a;
else if (cc2 == 7'd58)
     in2 = 128'h417726601224683c2909060c6c1c580a;
else if (cc2 == 7'd59)
     in2 = 128'h5b4977661c5b300e331f04115c0c3309;
else if (cc2 == 7'd60)
     in2 = 128'h2a3b622357430e633b1f71154e4d150b;
else if (cc2 == 7'd61)
     in2 = 128'h371d0023050742406d362c1b4332500a;
else if (cc2 == 7'd62)
     in2 = 128'h5a5e755f1e6d753b571c5a3d2575530a;
else if (cc2 == 7'd63)
     in2 = 128'h3d41392b15542934063134405d005009;
else if (cc2 == 7'd64)
     in2 = 128'h1e1c4b151c40286f776d5210555f1e0b;
else if (cc2 == 7'd65)
```

```
     in2 = 128'h072f1263010e3b55086405571e743c0a;
else if (cc2 == 7'd66)
     in2 = 128'h6f756d5b35102b3a2d654f4d4c52510a;
else if (cc2 == 7'd67)
     in2 = 128'h0c6d76086e524a6d22485a1167397209;
else if (cc2 == 7'd68)
     in2 = 128'h1d02060c0d4f575d635268400f0f760b;
else if (cc2 == 7'd69)
     in2 = 128'h257504336569270d28132d19375a480a;
else if (cc2 == 7'd70)
     in2 = 128'h23442462702d4a726566554808510a0a;
else if (cc2 == 7'd71)
     in2 = 128'h70552b74442e2476704c2c711a303d09;
else if (cc2 == 7'd72)
     in2 = 128'h144946684847282259 4c6818061e030b;
else if (cc2 == 7'd73)
     in2 = 128'h6d28253a0c54633a4867253928620b0a;
else if (cc2 == 7'd74)
     in2 = 128'h3921231a321f065d3418703653696b0a;
else if (cc2 == 7'd75)
     in2 = 128'h5b6e362c18645868337005005023 3a09;
else if (cc2 == 7'd76)
     in2 = 128'h460e694c62193e6e3c690a552739000b;
else if (cc2 == 7'd77)
     in2 = 128'h50266261371c2b52562a740a35574e0a;
else if (cc2 == 7'd78)
     in2 = 128'h7340262c5d3a2247703f120e3824320a;
else if (cc2 == 7'd79)
     in2 = 128'h144023124d546431665e070c76660409;
else if (cc2 == 7'd80)
     in2 = 128'h3b526d3b20570966200871192 96b620b;
else if (cc2 == 7'd81)
     in2 = 128'h54441b0d764c68055f714a4c774c770a;
else if (cc2 == 7'd82)
     in2 = 128'h242f143c3f00077014101b5e430a360a;
else if (cc2 == 7'd83)
     in2 = 128'h474732606a775f636134674644244609;
else if (cc2 == 7'd84)
     in2 = 128'h5b17005f7120095d5b23136c3a66220b;
else if (cc2 == 7'd85)
     in2 = 128'h030e144906250305156d5d744a234d0a;
else if (cc2 == 7'd86)
     in2 = 128'h4c58030a50420e6c553b1e466f62680a;
else if (cc2 == 7'd87)
     in2 = 128'h0a38073b0758641f5514312159476d09;
else if (cc2 == 7'd88)
     in2 = 128'h4828380b2c5d36643f6d776333533a0b;
else if (cc2 == 7'd89)
     in2 = 128'h0f383e6b6d6d703f632f4844203e010a;
else if (cc2 == 7'd90)
     in2 = 128'h17623129182e39263f376f393c695c0a;
else if (cc2 == 7'd91)
     in2 = 128'h2d24250815041b4159272f6757263a09;
else if (cc2 == 7'd92)
     in2 = 128'h0d1c52755145244e244431126d07740b;
else if (cc2 == 7'd93)
     in2 = 128'h5e56606e1c544537723c42323163110a;
else if (cc2 == 7'd94)
     in2 = 128'h54381140616237232469 2f153e5c640a;
else if (cc2 == 7'd95)
     in2 = 128'h1d385b2a16343d68406f2f6c5d4a4e09;
else if (cc2 == 7'd96)
     in2 = 128'h2b736769496b770b704e4844270c330b;
else if (cc2 == 7'd97)
     in2 = 128'h01230d151a7414332a694d502d29100a;
else if (cc2 == 7'd98)
     in2 = 128'h686d0c2320316545601b552764576c0a;
else if (cc2 == 7'd99)
     in2 = 128'h1a330d475e113370334847650b291109;
else if (cc2 == 7'd100)
```

```
     in2 = 128'h3477292b143f3b343577460e0a45040b;
else if (cc2 == 7'd101)
     in2 = 128'h543d3a5e72594c4f142f67702567160a;
else if (cc2 == 7'd102)
     in2 = 128'h1028513a394e663d57652f3a4f505a0a;
else if (cc2 == 7'd103)
     in2 = 128'h2c283a200f1c1d38424d76250c4b3709;
else if (cc2 == 7'd104)
     in2 = 128'h550e45572d754314480a52594b424b0b;
else if (cc2 == 7'd105)
     in2 = 128'h5622733c6f223a1a075a6b4632284a0a;
else if (cc2 == 7'd106)
     in2 = 128'h2d1c6c413e777232209564410b31630a;
else if (cc2 == 7'd107)
     in2 = 128'h0026680c3e732453662d362d41435d09;
else if (cc2 == 7'd108)
     in2 = 128'h02044b2f102b5e222f2a5d432e3d290b;
else if (cc2 == 7'd109)
     in2 = 128'h04524b6644251c576a666b5d3802450a;
else if (cc2 == 7'd110)
     in2 = 128'h5b01611905240d1d7663503c475e360a;
else if (cc2 == 7'd111)
     in2 = 128'h326f5c2451220a6e3257621f132a1609;
else if (cc2 == 7'd112)
     in2 = 128'h27100b02382e747564380420622320b;
else if (cc2 == 7'd113)
     in2 = 128'h0932615e3a116e2d67740f0d00560f0a;
else if (cc2 == 7'd114)
     in2 = 128'h0a13425f2a081c0f1460133963146e0a;
else if (cc2 == 7'd115)
     in2 = 128'h75696725174470104101 4d075f712509;
else if (cc2 == 7'd116)
     in2 = 128'h13283971101a30572b303721704d460b;
else if (cc2 == 7'd117)
     in2 = 128'h101e2a645b57286b481b44510559280a;
else if (cc2 == 7'd118)
     in2 = 128'h63584d09537270256f60392e5933200a;
else if (cc2 == 7'd119)
     in2 = 128'h6c300a6f36083a51422569140f466709;
else if (cc2 == 7'd120)
     in2 = 128'h2a2475021711753327432069213461 0b;
else if (cc2 == 7'd121)
     in2 = 128'h1f403d34212a3a3268581746025c1d0a;
else if (cc2 == 7'd122)
     in2 = 128'h6c61533d14085017605c6f04565f080a;
else if (cc2 == 7'd123)
     in2 = 128'h594b66304e232b245106566d19592c09;
else if (cc2 == 7'd124)
     in2 = 128'h2442470c104f18663c0a1e6e2632740b;
else if (cc2 == 7'd125)
     in2 = 128'h5e48192a3a46210b1a642937605f6e0a;
else if (cc2 == 7'd126)
     in2 = 128'h393136342c254a3b131d3d6a12131e0a;
else if (cc2 == 7'd127)
     in2 = 128'h5f083f5e01663a574d536756652c4b09;

//new port start
if (cc3 == 7'd0)
     in3 = 128'h3c5c20072a0a201402701c164066358f;
else if (cc3 == 7'd1)
     in3 = 128'h706b3230614f2574776f14490760608e;
else if (cc3 == 7'd2)
     in3 = 128'h112a6733695a1d08274d35582d62138e;
else if (cc3 == 7'd3)
     in3 = 128'h081f094b221e1628581b412d050b0b8d;
else if (cc3 == 7'd4)
     in3 = 128'h66191b44305a4a640e5b6b6d6d10318f;
else if (cc3 == 7'd5)
     in3 = 128'h0221081f0d6350351909722a3d52038e;
else if (cc3 == 7'd6)
```

```
          in3 = 128'h6a13245c010e43146f590b22014b6b8e;
     else if (cc3 == 7'd7)
          in3 = 128'h02096657085b6a122c354f3306142a8d;
     else if (cc3 == 7'd8)
          in3 = 128'h290b157741095961536d5533542a368f;
     else if (cc3 == 7'd9)
          in3 = 128'h0a26093a2d5b1f77296a0d4a2836008e;
     else if (cc3 == 7'd10)
          in3 = 128'h3164611d18070917681a2e4706466c8e;
     else if (cc3 == 7'd11)
          in3 = 128'h4068572f363e364650154a5e3c75668d;
     else if (cc3 == 7'd12)
          in3 = 128'h02675d6a5b0e0101196d3327345e508f;
     else if (cc3 == 7'd13)
          in3 = 128'h1b09433d3768354c695a0a621771158e;
     else if (cc3 == 7'd14)
          in3 = 128'h73021a170042746876544c3268201a8e;
     else if (cc3 == 7'd15)
          in3 = 128'h60526d526d0e2a174d473c4f3e5a3c8d;
     else if (cc3 == 7'd16)
          in3 = 128'h183f671861111343295d31333f5b6d8f;
     else if (cc3 == 7'd17)
          in3 = 128'h6c5c3e192f703a2d14070f4f505e618e;
     else if (cc3 == 7'd18)
          in3 = 128'h2e3c3b49111d5f546c71480a6764428e;
     else if (cc3 == 7'd19)
          in3 = 128'h4c69410a39177607503d2161134e508d;
     else if (cc3 == 7'd20)
          in3 = 128'h0f535657644d3e36035105230a4d4b8f;
     else if (cc3 == 7'd21)
          in3 = 128'h6d29727534276c101a57233a2826658e;
     else if (cc3 == 7'd22)
          in3 = 128'h393d2a0546241d4c66067465760f1e8e;
     else if (cc3 == 7'd23)
          in3 = 128'h5d7157765b0e380f0b051f6f0d26328d;
     else if (cc3 == 7'd24)
          in3 = 128'h5024464f15444d3718506156336444 8f;
     else if (cc3 == 7'd25)
          in3 = 128'h601f6e2024667458426352185205138e;
     else if (cc3 == 7'd26)
          in3 = 128'h046f226d63152f54487411213250408e;
     else if (cc3 == 7'd27)
          in3 = 128'h0e53411a392c0c3d2f0025672520758d;
     else if (cc3 == 7'd28)
          in3 = 128'h334a4f6475741b56156f140d353b218f;
     else if (cc3 == 7'd29)
          in3 = 128'h57455e310e483a3558774b1330740e8e;
     else if (cc3 == 7'd30)
          in3 = 128'h13280e3e6d68252f1e07041f1e463a8e;
     else if (cc3 == 7'd31)
          in3 = 128'h00567315365b43253b08250d161d108d;
     else if (cc3 == 7'd32)
          in3 = 128'h5152161e0e76746e6f716c2d01216c8f;
     else if (cc3 == 7'd33)
          in3 = 128'h67116b4c2d26670b366017156d113d8e;
     else if (cc3 == 7'd34)
          in3 = 128'h625a405e0b2c5732622f1e322708458e;
     else if (cc3 == 7'd35)
          in3 = 128'h173947304820084c5611340028635e8d;
     else if (cc3 == 7'd36)
          in3 = 128'h3f251d764845221739005f6b0729248f;
     else if (cc3 == 7'd37)
          in3 = 128'h34543d472f2f34453d47134314192b8e;
     else if (cc3 == 7'd38)
          in3 = 128'h074f024a67255b4c737417244e513c8e;
     else if (cc3 == 7'd39)
          in3 = 128'h591059555819533e204d23033a1d3b8d;
     else if (cc3 == 7'd40)
          in3 = 128'h5c2740585b23542e1d2b3636421b628f;
     else if (cc3 == 7'd41)
     in3 = 128'h447125431f1c5346740960547232398e;
else if (cc3 == 7'd42)
     in3 = 128'h05106042185e68201b257423356d5d8e;
else if (cc3 == 7'd43)
     in3 = 128'h6e164e204c744b00652c776b0109238d;
else if (cc3 == 7'd44)
     in3 = 128'h45032c3f4c405d056b0838000d61328f;
else if (cc3 == 7'd45)
     in3 = 128'h6c6e0f0e2607744c1c541a3849391a8e;
else if (cc3 == 7'd46)
     in3 = 128'h635f122e0c1d2d4b270b6621182b368e;
else if (cc3 == 7'd47)
     in3 = 128'h4b762f173d71150d6d276c4d32162d8d;
else if (cc3 == 7'd48)
     in3 = 128'h191d29383347343a076e423e0d184f8f;
else if (cc3 == 7'd49)
     in3 = 128'h4a056234486d2a5324735168434a118e;
else if (cc3 == 7'd50)
     in3 = 128'h0256051e3b6b0150001a07751d555b8e;
else if (cc3 == 7'd51)
     in3 = 128'h300c3744410f123b21292952250e1f8d;
else if (cc3 == 7'd52)
     in3 = 128'h30615d11184f6c4a606d6f2c01542e8f;
else if (cc3 == 7'd53)
     in3 = 128'h22003d03270225190405271c7451358e;
else if (cc3 == 7'd54)
     in3 = 128'h6918445c2d5b365a525763711e4f778e;
else if (cc3 == 7'd55)
     in3 = 128'h093a526b701f320f04076009070b108d;
else if (cc3 == 7'd56)
     in3 = 128'h60061503693e6d70282a216730700e8f;
else if (cc3 == 7'd57)
     in3 = 128'h37230540390e5e46770a360f4756068e;
else if (cc3 == 7'd58)
     in3 = 128'h105c2e514915043f3525711266280a8e;
else if (cc3 == 7'd59)
     in3 = 128'h3d382a261d18591c532628013c38318d;
else if (cc3 == 7'd60)
     in3 = 128'h0924651c54556f41751308655a15518f;
else if (cc3 == 7'd61)
     in3 = 128'h5a05673a52147047185b6a357351028e;
else if (cc3 == 7'd62)
     in3 = 128'h511d14175e15200f56073e084a27698e;
else if (cc3 == 7'd63)
     in3 = 128'h6b1550123f36602b5329316b104f3e8d;
else if (cc3 == 7'd64)
     in3 = 128'h0525422c6b1c0b16451f305d4a595b8f;
else if (cc3 == 7'd65)
     in3 = 128'h63382e695c402f3826132b1c5500768e;
else if (cc3 == 7'd66)
     in3 = 128'h737606110f42513006695509155 46a8e;
else if (cc3 == 7'd67)
     in3 = 128'h474f4f3d1c070d612c3e030e5b0a208d;
else if (cc3 == 7'd68)
     in3 = 128'h5a60153d10453137072524562a34348f;
else if (cc3 == 7'd69)
     in3 = 128'h1a5d0b07255c392a6a611e316b164f8e;
else if (cc3 == 7'd70)
     in3 = 128'h20630111073a09183d6901690c23528e;
else if (cc3 == 7'd71)
     in3 = 128'h67071f773c6d620b186b1a00220a4b8d;
else if (cc3 == 7'd72)
     in3 = 128'h013835432f731f33471d3f1b1e6f268f;
else if (cc3 == 7'd73)
     in3 = 128'h2c552714014d4b32116e264c3e296e8e;
else if (cc3 == 7'd74)
     in3 = 128'h5a2f671a6e4f46096b7118045b65108e;
else if (cc3 == 7'd75)
     in3 = 128'h182e24323f58343e185477126606628d;
else if (cc3 == 7'd76)
```

```
          in3 = 128'h691e39085a221b7720400241632e508f;
      else if (cc3 == 7'd77)
          in3 = 128'h44512b01292d165923122157500e3e8e;
      else if (cc3 == 7'd78)
          in3 = 128'h01545a4f3d4f0d2a2a3f6f19504a588e;
      else if (cc3 == 7'd79)
          in3 = 128'h261808102f6c1b1e621175091b02258d;
      else if (cc3 == 7'd80)
          in3 = 128'h075b111b0010081c5011145f0e21608f;
      else if (cc3 == 7'd81)
          in3 = 128'h0f13576b424c73134d4d412466555d8e;
      else if (cc3 == 7'd82)
          in3 = 128'h44154c062535545d720677343c11568e;
      else if (cc3 == 7'd83)
          in3 = 128'h735f4224251b0f361c5908682e31338d;
      else if (cc3 == 7'd84)
          in3 = 128'h464a372b685a40646f0541190d592e8f;
      else if (cc3 == 7'd85)
          in3 = 128'h044b126d3d743e6b422647492f11408e;
      else if (cc3 == 7'd86)
          in3 = 128'h1777110b633b210d0c34070e4930628e;
      else if (cc3 == 7'd87)
          in3 = 128'h29040830221a534a591c59266669758d;
      else if (cc3 == 7'd88)
          in3 = 128'h4b1b551068373071276d0a666e00428f;
      else if (cc3 == 7'd89)
          in3 = 128'h29130373142c054531073e44436c2a8e;
      else if (cc3 == 7'd90)
          in3 = 128'h51376a3a35517515386f081b056f1e8e;
      else if (cc3 == 7'd91)
          in3 = 128'h6334143808572d553505333c7760288d;
      else if (cc3 == 7'd92)
          in3 = 128'h35400c021d362c3c3b6031101863138f;
      else if (cc3 == 7'd93)
          in3 = 128'h2b766039434b007169734c203f232a8e;
      else if (cc3 == 7'd94)
          in3 = 128'h625f6a385b293f583a676970462a178e;
      else if (cc3 == 7'd95)
          in3 = 128'h503f6c6d2e2267754a6d4b3a0a321b8d;
      else if (cc3 == 7'd96)
          in3 = 128'h0742164d765e3110541616396f576e8f;
      else if (cc3 == 7'd97)
          in3 = 128'h3b69086b450d1243682058341037468e;
      else if (cc3 == 7'd98)
          in3 = 128'h092a35044d414c44211e6d5a11142f8e;
      else if (cc3 == 7'd99)
          in3 = 128'h36544d110041365d644954301726008d;
      else if (cc3 == 7'd100)
          in3 = 128'h543d494c1f52705c5952322e3060078f;
      else if (cc3 == 7'd101)
          in3 = 128'h24615315480b72513161061b19184e8e;
      else if (cc3 == 7'd102)
          in3 = 128'h053a0a73421f2760321f1c085c38658e;
      else if (cc3 == 7'd103)
          in3 = 128'h376757733a27251902643b622135458d;
      else if (cc3 == 7'd104)
          in3 = 128'h6b440c625d1733364a2a6f2b4932198f;
      else if (cc3 == 7'd105)
          in3 = 128'h36423262414e143166093d6f1c69348e;
      else if (cc3 == 7'd106)
          in3 = 128'h4e355302404d0a684b6715123d6e438e;
      else if (cc3 == 7'd107)
          in3 = 128'h6928740c610e290d145b2c0e601d498d;
      else if (cc3 == 7'd108)
          in3 = 128'h4d54451506560d100c495a171e30378f;
      else if (cc3 == 7'd109)
          in3 = 128'h0b1d0d2427634026254b361f3461588e;
      else if (cc3 == 7'd110)
          in3 = 128'h0a6d2b054a55075d5820702f6c3f208e;
      else if (cc3 == 7'd111)
```

```
     in3 = 128'h110844452b1f24594b1d607000103b8d;
else if (cc3 == 7'd112)
     in3 = 128'h76122c1e0c6c0537395c375a641e3b8f;
else if (cc3 == 7'd113)
     in3 = 128'h6206015c2f0a6d722a2309615b55628e;
else if (cc3 == 7'd114)
     in3 = 128'h334b460d68076c44446f6f3d560a5a8e;
else if (cc3 == 7'd115)
     in3 = 128'h5d0b1d5d73081c71406b4d495625458d;
else if (cc3 == 7'd116)
     in3 = 128'h690a58333b12340f19606021520d1e8f;
else if (cc3 == 7'd117)
     in3 = 128'h2c72714d4c1e26542d3a2650323f428e;
else if (cc3 == 7'd118)
     in3 = 128'h3f6802604e2165235f4e1405764a528e;
else if (cc3 == 7'd119)
     in3 = 128'h5b1511735e5f576e575274653c641a8d;
else if (cc3 == 7'd120)
     in3 = 128'h0f13314e4c453f472a0632691d70558f;
else if (cc3 == 7'd121)
     in3 = 128'h260d60530343284d4312354b0f17558e;
else if (cc3 == 7'd122)
     in3 = 128'h7438053f4032336a65481a136c554f8e;
else if (cc3 == 7'd123)
     in3 = 128'h366d72153f0c30173139522542621b8d;
else if (cc3 == 7'd124)
     in3 = 128'h0411226923710c4f373774027230358f;
else if (cc3 == 7'd125)
     in3 = 128'h70161b58253e1c26632e58033e0a438e;
else if (cc3 == 7'd126)
     in3 = 128'h500d071a311a6f2a601d321f3f1a648e;
else if (cc3 == 7'd127)
     in3 = 128'h035f30574b6305570555433d5f54378d;

//new port start
if (cc4 == 7'd0)
     in4 = 128'h47636e34140c5873035b102049380a13;
else if (cc4 == 7'd1)
     in4 = 128'h3324302d5b2573312e420041614c1012;
else if (cc4 == 7'd2)
     in4 = 128'h21185632116e745b69145b0c4a591712;
else if (cc4 == 7'd3)
     in4 = 128'h033f286b126622613059392063470911;
else if (cc4 == 7'd4)
     in4 = 128'h5a2b4b6123610864326d61304a103613;
else if (cc4 == 7'd5)
     in4 = 128'h5e082856531b415b140b0a3e5f480912;
else if (cc4 == 7'd6)
     in4 = 128'h3917773e6d753a75191e052e05605b12;
else if (cc4 == 7'd7)
     in4 = 128'h610d4b0335682c36365d6872713b1211;
else if (cc4 == 7'd8)
     in4 = 128'h6c6e0e3f201e5902752f214a1c3d3713;
else if (cc4 == 7'd9)
     in4 = 128'h0f43552b09412410335f366c032b5212;
else if (cc4 == 7'd10)
     in4 = 128'h0b63681a6e616e29690d3029690d0d12;
else if (cc4 == 7'd11)
     in4 = 128'h641b0c186d136d684b754b3e40262c11;
else if (cc4 == 7'd12)
     in4 = 128'h4737282240320a125f76671847160713;
else if (cc4 == 7'd13)
     in4 = 128'h1b396c2a4a426b074b55605d6e020012;
else if (cc4 == 7'd14)
     in4 = 128'h1e154e5c4873584750591c31141a0212;
else if (cc4 == 7'd15)
     in4 = 128'h7210202c2b2920047337141a290e2e11;
else if (cc4 == 7'd16)
     in4 = 128'h286e265c410160576c22524f4c2c2a13;
else if (cc4 == 7'd17)
```

```
     in4 = 128'h6d0c622174530d3a5d5e4c27764e3612;
else if (cc4 == 7'd18)
     in4 = 128'h2d1a5a570742681e2e262750062a6612;
else if (cc4 == 7'd19)
     in4 = 128'h664f77517735032e6019556247670711;
else if (cc4 == 7'd20)
     in4 = 128'h190d030c6e3d6c092051593d59274613;
else if (cc4 == 7'd21)
     in4 = 128'h250d514c712a391167582b0d09394312;
else if (cc4 == 7'd22)
     in4 = 128'h3e284c2116466e2d1d4a141d31232d12;
else if (cc4 == 7'd23)
     in4 = 128'h31572c69321d0d010f31111e6c052111;
else if (cc4 == 7'd24)
     in4 = 128'h685a0104744769757131056 72f745413;
else if (cc4 == 7'd25)
     in4 = 128'h50233a321714076f3317584b54454412;
else if (cc4 == 7'd26)
     in4 = 128'h68694b235b30071a450d6e663c010b12;
else if (cc4 == 7'd27)
     in4 = 128'h3e1640234d523066071738522b540811;
else if (cc4 == 7'd28)
     in4 = 128'h066a1e5b4d10142c1d15355355632213;
else if (cc4 == 7'd29)
     in4 = 128'h063a2f3a1b772c2d662a171811694812;
else if (cc4 == 7'd30)
     in4 = 128'h7037297515080b023645576163452512;
else if (cc4 == 7'd31)
     in4 = 128'h1e5835732700073f5345482a525c4d11;
else if (cc4 == 7'd32)
     in4 = 128'h311f223246203647346a24301b603e13;
else if (cc4 == 7'd33)
     in4 = 128'h373f2b5a7103307442340315027219 12;
else if (cc4 == 7'd34)
     in4 = 128'h380f4a622e6454116d6d602a77212012;
else if (cc4 == 7'd35)
     in4 = 128'h18164e5b264a6a3d2a2d551227184611;
else if (cc4 == 7'd36)
     in4 = 128'h58170b46152e66531f74556b1b493613;
else if (cc4 == 7'd37)
     in4 = 128'h05540c4a1c1b3e323e553b1b6b4b7312;
else if (cc4 == 7'd38)
     in4 = 128'h165b30691d384232350e30111d102912;
else if (cc4 == 7'd39)
     in4 = 128'h3356466f29121d1c753f705d76586211;
else if (cc4 == 7'd40)
     in4 = 128'h5166294e195e1f4159713a1b2b420e13;
else if (cc4 == 7'd41)
     in4 = 128'h4a6a41283e2a5a6822772f476f734812;
else if (cc4 == 7'd42)
     in4 = 128'h62122e306557007321315c3f081f0b12;
else if (cc4 == 7'd43)
     in4 = 128'h7464285316130f3d3c031331204e3d11;
else if (cc4 == 7'd44)
     in4 = 128'h1a67351117445d1a0e3d5126315a4c13;
else if (cc4 == 7'd45)
     in4 = 128'h36356a3d113e01132c32041a70325812;
else if (cc4 == 7'd46)
     in4 = 128'h54264d5730720f6627585e7443505b12;
else if (cc4 == 7'd47)
     in4 = 128'h351f1f32221b4422053d154b765a7611;
else if (cc4 == 7'd48)
     in4 = 128'h2b342e5813456d01502d43393a025613;
else if (cc4 == 7'd49)
     in4 = 128'h5477302627076b3e6d17155c563c5a12;
else if (cc4 == 7'd50)
     in4 = 128'h6444620e1d65116343725e6536495612;
else if (cc4 == 7'd51)
     in4 = 128'h764401746d695729263517204d280711;
else if (cc4 == 7'd52)
```

```
          in4 = 128'h150a6f1d3f226f0c2959125a43514313;
     else if (cc4 == 7'd53)
          in4 = 128'h2b624e461407523c07575062395d7412;
     else if (cc4 == 7'd54)
          in4 = 128'h7259561f636f5d3b0b7046726a2a0412;
     else if (cc4 == 7'd55)
          in4 = 128'h7366170c6c570d59525558560c720011;
     else if (cc4 == 7'd56)
          in4 = 128'h0477435b07270c5c6763186b6b163313;
     else if (cc4 == 7'd57)
          in4 = 128'h314d441f570f522b613a23680d095f12;
     else if (cc4 == 7'd58)
          in4 = 128'h4f550f260937046a0e48756d77682c12;
     else if (cc4 == 7'd59)
          in4 = 128'h4b24565e0f2c571e5d13452f2c606d11;
     else if (cc4 == 7'd60)
          in4 = 128'h0a6a5a440061516218752d291f107213;
     else if (cc4 == 7'd61)
          in4 = 128'h61333e19490d155f611f6b384a0e5612;
     else if (cc4 == 7'd62)
          in4 = 128'h562d2b6850243a476c53673a4c621a12;
     else if (cc4 == 7'd63)
          in4 = 128'h3a243d6b374e04660e58283142542711;
     else if (cc4 == 7'd64)
          in4 = 128'h4531775e60675f6a4d061e3259531313;
     else if (cc4 == 7'd65)
          in4 = 128'h4b2237106c316634596c2d1968302912;
     else if (cc4 == 7'd66)
          in4 = 128'h1863663812510c2e603a0e27532a7012;
     else if (cc4 == 7'd67)
          in4 = 128'h255e335f52772d4c0841777229376d11;
     else if (cc4 == 7'd68)
          in4 = 128'h6d6f0c1272375d3027094d155c634813;
     else if (cc4 == 7'd69)
          in4 = 128'h613f481b1d0948445466636c33274a12;
     else if (cc4 == 7'd70)
          in4 = 128'h03156c324e56603843380b5e50031e12;
     else if (cc4 == 7'd71)
          in4 = 128'h18466f310e071877616f6b2918323f11;
     else if (cc4 == 7'd72)
          in4 = 128'h6f6800104a40413f016c281062512713;
     else if (cc4 == 7'd73)
          in4 = 128'h060332616c6534041c403f5e51126912;
     else if (cc4 == 7'd74)
          in4 = 128'h011067100013240e5c3d695b1e634212;
     else if (cc4 == 7'd75)
          in4 = 128'h5a043816191418193d01276f55297311;
     else if (cc4 == 7'd76)
          in4 = 128'h480a6a4c15531862442f2b561b506813;
     else if (cc4 == 7'd77)
          in4 = 128'h5f6f244165574e6f113207204c5b1d12;
     else if (cc4 == 7'd78)
          in4 = 128'h14651e565911202617694a495b483712;
     else if (cc4 == 7'd79)
          in4 = 128'h33374c48614746561225284d41311611;
     else if (cc4 == 7'd80)
          in4 = 128'h48614b51440e2631301d6c336f3c0413;
     else if (cc4 == 7'd81)
          in4 = 128'h0e322315766d61001c643b364f4b1712;
     else if (cc4 == 7'd82)
          in4 = 128'h4d33566b2950020e61546e2c1b526f12;
     else if (cc4 == 7'd83)
          in4 = 128'h775477365c3a0c73117714612a260311;
     else if (cc4 == 7'd84)
          in4 = 128'h2147512c1929223675544f2b421a6a13;
     else if (cc4 == 7'd85)
          in4 = 128'h700d61355c2a1734516f1a0b0b206512;
     else if (cc4 == 7'd86)
          in4 = 128'h064b742d734a1b135f216c6d14343112;
     else if (cc4 == 7'd87)
     in4 = 128'h1765160d292e013a591434694f0c7211;
else if (cc4 == 7'd88)
     in4 = 128'h38461a070034437054641d1934304013;
else if (cc4 == 7'd89)
     in4 = 128'h6e0940046a1c2c2820502124074e6e12;
else if (cc4 == 7'd90)
     in4 = 128'h5d4932523f6a62192e55096659625d12;
else if (cc4 == 7'd91)
     in4 = 128'h1d53062e4450317005371f344b436411;
else if (cc4 == 7'd92)
     in4 = 128'h2d253e6d6c283108324706724565271З;
else if (cc4 == 7'd93)
     in4 = 128'h756d284421274f594e5d59095e715412;
else if (cc4 == 7'd94)
     in4 = 128'h1b4e0508594b6d2a08653262051e3c12;
else if (cc4 == 7'd95)
     in4 = 128'h5f230f100b131c4a3c1f1f261e5c6311;
else if (cc4 == 7'd96)
     in4 = 128'h17776b564e512b526f1a383c34764613;
else if (cc4 == 7'd97)
     in4 = 128'h197776014c3a5d7531290b43093b5112;
else if (cc4 == 7'd98)
     in4 = 128'h4d320a73014b372a5f23501452430712;
else if (cc4 == 7'd99)
     in4 = 128'h475d616871614d330332642c722f1811;
else if (cc4 == 7'd100)
     in4 = 128'h6c5b745649615813605c1f2d4c562113;
else if (cc4 == 7'd101)
     in4 = 128'h6a0f56352577103474310a2f2f5c1512;
else if (cc4 == 7'd102)
     in4 = 128'h0f1c4877675e74124931766a3a6a1312;
else if (cc4 == 7'd103)
     in4 = 128'h1f75144a131b0847563f4259186c2111;
else if (cc4 == 7'd104)
     in4 = 128'h764e4916630032485e74394f5d0d5113;
else if (cc4 == 7'd105)
     in4 = 128'h38210f2d29651c346912724f6f265d12;
else if (cc4 == 7'd106)
     in4 = 128'h72700b0a243e155f696222461e417612;
else if (cc4 == 7'd107)
     in4 = 128'h35036f744f05361f480d1b2653361b11;
else if (cc4 == 7'd108)
     in4 = 128'h4e4b231d562375675c271d56426b7513;
else if (cc4 == 7'd109)
     in4 = 128'h0645082d0e315f3e26434d650a264812;
else if (cc4 == 7'd110)
     in4 = 128'h14175c7648150c124b6e64666e695212;
else if (cc4 == 7'd111)
     in4 = 128'h415159706735443c4b5d5c3272614b11;
else if (cc4 == 7'd112)
     in4 = 128'h2e2e39632e52394961700f751f465d13;
else if (cc4 == 7'd113)
     in4 = 128'h0d1839405e512f1a48284046493a2712;
else if (cc4 == 7'd114)
     in4 = 128'h4535575b4a6f77045a594726704c4d12;
else if (cc4 == 7'd115)
     in4 = 128'h1f2206102a5f203e5e71140a1c241211;
else if (cc4 == 7'd116)
     in4 = 128'h056d0e533a355c2117695e46694d4613;
else if (cc4 == 7'd117)
     in4 = 128'h0f1a6a0a083f260a295c0a160b562f12;
else if (cc4 == 7'd118)
     in4 = 128'h4b446e60495f1f6765626f2903072d12;
else if (cc4 == 7'd119)
     in4 = 128'h752269252f435c425533416d43451511;
else if (cc4 == 7'd120)
     in4 = 128'h191b535208632b034771302b00623613;
else if (cc4 == 7'd121)
     in4 = 128'h14352e1d4c564a6c65040a4b01533212;
else if (cc4 == 7'd122)
```

```
     in4 = 128'h4b483e451b502c604e4804440e2a3612;
else if (cc4 == 7'd123)
     in4 = 128'h43306534730f6a440311454e17316f11;
else if (cc4 == 7'd124)
     in4 = 128'h303a111c08087310116e4b60150a1d13;
else if (cc4 == 7'd125)
     in4 = 128'h1321401f5f6314185c0a615546361c12;
else if (cc4 == 7'd126)
     in4 = 128'h6d5e4b25611b0c500f17694e0a341b12;
else if (cc4 == 7'd127)
     in4 = 128'h42502d6f713d7633635424695d702211;

//new port start
if (cc5 == 7'd0)
     in5 = 128'h0c0c6d6b151b253f282721644b5b2d6f;
else if (cc5 == 7'd1)
     in5 = 128'h33490005551871471f1a395c1411466e;
else if (cc5 == 7'd2)
     in5 = 128'h0b44386e305041664f4f520b0622506e;
else if (cc5 == 7'd3)
     in5 = 128'h66115b0a3430666f021d3d2f2d54556d;
else if (cc5 == 7'd4)
     in5 = 128'h285a2e1d693d7424343d0533161c336f;
else if (cc5 == 7'd5)
     in5 = 128'h5d60011935686a696d01494470776a6e;
else if (cc5 == 7'd6)
     in5 = 128'h5a425e02314b591936086f10636b756e;
else if (cc5 == 7'd7)
     in5 = 128'h65601473732472220c7169213045716d;
else if (cc5 == 7'd8)
     in5 = 128'h57351834623b517169442d055108186f;
else if (cc5 == 7'd9)
     in5 = 128'h24453d44042d2c6f4b372f574e525e6e;
else if (cc5 == 7'd10)
     in5 = 128'h3e0f2b74024c5b11446811375b41546e;
else if (cc5 == 7'd11)
     in5 = 128'h2a6b1a0931494d0e4e0243380d67026d;
else if (cc5 == 7'd12)
     in5 = 128'h26024c622c6a5f4a360b206562044f6f;
else if (cc5 == 7'd13)
     in5 = 128'h6823693c505d3a162a17656532481b6e;
else if (cc5 == 7'd14)
     in5 = 128'h2c205e4a565f10381e25541d1a491e6e;
else if (cc5 == 7'd15)
     in5 = 128'h060561350a72394c045b6a740c48626d;
else if (cc5 == 7'd16)
     in5 = 128'h053e155d2f30083b266d01411a50456f;
else if (cc5 == 7'd17)
     in5 = 128'h35260f373a30404742143023061a166e;
else if (cc5 == 7'd18)
     in5 = 128'h2542500040144d5526356b4f0010366e;
else if (cc5 == 7'd19)
     in5 = 128'h2f0a4b0c23006c616d293c1c0e2f2d6d;
else if (cc5 == 7'd20)
     in5 = 128'h34453131216212344c2e46162440006f;
else if (cc5 == 7'd21)
     in5 = 128'h167074034b27400a696d023d071a046e;
else if (cc5 == 7'd22)
     in5 = 128'h720f5f4d436e69301e58546642291f6e;
else if (cc5 == 7'd23)
     in5 = 128'h0474744521464114473c512315526b6d;
else if (cc5 == 7'd24)
     in5 = 128'h09004f1a2e25223e42171e057474636f;
else if (cc5 == 7'd25)
     in5 = 128'h23593431032f271c1a535a267263476e;
else if (cc5 == 7'd26)
     in5 = 128'h2020623c744c0a6f575d193c3a3c156e;
else if (cc5 == 7'd27)
     in5 = 128'h031b6b4114406e17164f494e6e013d6d;
else if (cc5 == 7'd28)
```

```
        in5 = 128'h0b0e0f016766014856431b047602336f;
    else if (cc5 == 7'd29)
        in5 = 128'h552d36326c036c420d5b644165064c6e;
    else if (cc5 == 7'd30)
        in5 = 128'h39276a02612a5d2a672765361008476e;
    else if (cc5 == 7'd31)
        in5 = 128'h32044371532e29406b4642742774456d;
    else if (cc5 == 7'd32)
        in5 = 128'h6f3623252e116f11727164415c77246f;
    else if (cc5 == 7'd33)
        in5 = 128'h10194a48123161274b6b11662e24556e;
    else if (cc5 == 7'd34)
        in5 = 128'h567473155c3725010d6a4a6e0c3e296e;
    else if (cc5 == 7'd35)
        in5 = 128'h5f6b0c2c134e17501f25504b5538276d;
    else if (cc5 == 7'd36)
        in5 = 128'h352b1374263f2b535832100b5d08246f;
    else if (cc5 == 7'd37)
        in5 = 128'h7067226776311b34387328324c1b656e;
    else if (cc5 == 7'd38)
        in5 = 128'h361b1b01761f214861006f6d473f446e;
    else if (cc5 == 7'd39)
        in5 = 128'h6c441a523452366f5b31621a6851606d;
    else if (cc5 == 7'd40)
        in5 = 128'h142d2b1f4619433b0b624f366155546f;
    else if (cc5 == 7'd41)
        in5 = 128'h613822365a16756b63521d28141d3f6e;
    else if (cc5 == 7'd42)
        in5 = 128'h287374671423703d1914584c3a2b3a6e;
    else if (cc5 == 7'd43)
        in5 = 128'h4d5b626c17243c19595c0851492a0c6d;
    else if (cc5 == 7'd44)
        in5 = 128'h1d300800776b5a6b4207765e565b3e6f;
    else if (cc5 == 7'd45)
        in5 = 128'h27283350375e6642436e425f1643596e;
    else if (cc5 == 7'd46)
        in5 = 128'h53360d193d0228264b412d6d3656696e;
    else if (cc5 == 7'd47)
        in5 = 128'h0b76421a700d263174360e1e142c516d;
    else if (cc5 == 7'd48)
        in5 = 128'h1b112b3110584e42294520375c546d6f;
    else if (cc5 == 7'd49)
        in5 = 128'h24251c626d1f705c481f35522345626e;
    else if (cc5 == 7'd50)
        in5 = 128'h1b30621932583c70463528665348596e;
    else if (cc5 == 7'd51)
        in5 = 128'h164f4b10500f2651561f7519053b326d;
    else if (cc5 == 7'd52)
        in5 = 128'h0f5d1136604c22330e1e11546c3c516f;
    else if (cc5 == 7'd53)
        in5 = 128'h2815015a46373d746d1e150d56494d6e;
    else if (cc5 == 7'd54)
        in5 = 128'h491b463c2634433c4e3820486c0d326e;
    else if (cc5 == 7'd55)
        in5 = 128'h110e213027514a1a76723763672b576d;
    else if (cc5 == 7'd56)
        in5 = 128'h5c3358332d6c0567732c280c4c45006f;
    else if (cc5 == 7'd57)
        in5 = 128'h5e11101557645e4553132f0a492f346e;
    else if (cc5 == 7'd58)
        in5 = 128'h226c2b49651b18090a633d10610f106e;
    else if (cc5 == 7'd59)
        in5 = 128'h2831363a51362f031a3801576c3b316d;
    else if (cc5 == 7'd60)
        in5 = 128'h260b57291415011c1a1828643972536f;
    else if (cc5 == 7'd61)
        in5 = 128'h5b044d6f54701d7249505c6b6c0b546e;
    else if (cc5 == 7'd62)
        in5 = 128'h31460a34446c5e0a0522416e0f77446e;
    else if (cc5 == 7'd63)
```

```
        in5 = 128'h5c643d004e55760b5d1868065c4b3d6d;
    else if (cc5 == 7'd64)
        in5 = 128'h060b6c0e101b59444e3c504e24646c6f;
    else if (cc5 == 7'd65)
        in5 = 128'h15470131432e693e422e05746c3f716e;
    else if (cc5 == 7'd66)
        in5 = 128'h614d0d6d044e245039220542421a026e;
    else if (cc5 == 7'd67)
        in5 = 128'h463d5a4e6e4e203360310f6c6a1f776d;
    else if (cc5 == 7'd68)
        in5 = 128'h28310d2b3463540d294401372e075f6f;
    else if (cc5 == 7'd69)
        in5 = 128'h034f7161122e753e0d525b20456d476e;
    else if (cc5 == 7'd70)
        in5 = 128'h726e3a062b5b7509651000724077446e;
    else if (cc5 == 7'd71)
        in5 = 128'h5b25576a3c77065377005431335 21c6d;
    else if (cc5 == 7'd72)
        in5 = 128'h340b42040650194632023265 1b4e556f;
    else if (cc5 == 7'd73)
        in5 = 128'h013d45513577320e323b21342e16456e;
    else if (cc5 == 7'd74)
        in5 = 128'h5a25473358082f33146c2d095223686e;
    else if (cc5 == 7'd75)
        in5 = 128'h1f22045a2d6b470e593406013c2d5d6d;
    else if (cc5 == 7'd76)
        in5 = 128'h2f360f6b776a72026b14614c3044276f;
    else if (cc5 == 7'd77)
        in5 = 128'h26461125752d5241040a616b13064c6e;
    else if (cc5 == 7'd78)
        in5 = 128'h436d09300a065d3b5d5b4b77201f156e;
    else if (cc5 == 7'd79)
        in5 = 128'h736108195d433963766d1a3e1436266d;
    else if (cc5 == 7'd80)
        in5 = 128'h664326304015613c6b17325f3955696f;
    else if (cc5 == 7'd81)
        in5 = 128'h18446a476c240b310710743e4e484a6e;
    else if (cc5 == 7'd82)
        in5 = 128'h4239675a16102b2f4e386b226e4c186e;
    else if (cc5 == 7'd83)
        in5 = 128'h441b3e6a524803181d01530323452b6d;
    else if (cc5 == 7'd84)
        in5 = 128'h0b5e2d2e134e6c7618574e01734e386f;
    else if (cc5 == 7'd85)
        in5 = 128'h2e005c17207254413318266c15372b6e;
    else if (cc5 == 7'd86)
        in5 = 128'h6754025a2c02610f490b74210444416e;
    else if (cc5 == 7'd87)
        in5 = 128'h19397300313f0732043a48253c69656d;
    else if (cc5 == 7'd88)
        in5 = 128'h7647420f4826323b56453c165b4a746f;
    else if (cc5 == 7'd89)
        in5 = 128'h0316115b3b170b293667101e420a126e;
    else if (cc5 == 7'd90)
        in5 = 128'h2c1248533325681c296c47164029176e;
    else if (cc5 == 7'd91)
        in5 = 128'h0b1e696a25382a2b0014320e5e0e446d;
    else if (cc5 == 7'd92)
        in5 = 128'h022977482f36081d390a0b4f2c5a5d6f;
    else if (cc5 == 7'd93)
        in5 = 128'h744253270859410 82c354f724b36766e;
    else if (cc5 == 7'd94)
        in5 = 128'h27270a2a563e4f3e31511c1715624c6e;
    else if (cc5 == 7'd95)
        in5 = 128'h52500b201e176213091653650d41726d;
    else if (cc5 == 7'd96)
        in5 = 128'h721367585a440a126b5520416a0c696f;
    else if (cc5 == 7'd97)
        in5 = 128'h674c4d726a191b4a12175e31641d636e;
    else if (cc5 == 7'd98)
```

```
     in5 = 128'h16504f2504091a280e4f0f1f22543b6e;
else if (cc5 == 7'd99)
     in5 = 128'h0d4b351d214948265926334750742 86d;
else if (cc5 == 7'd100)
     in5 = 128'h24033d68073a34700a2220311d4f5e6f;
else if (cc5 == 7'd101)
     in5 = 128'h560f31174b3b4426470d252f426f0b6e;
else if (cc5 == 7'd102)
     in5 = 128'h374a6757394a1f1643351c4408124c6e;
else if (cc5 == 7'd103)
     in5 = 128'h37086a1c20050a0b3001683551501d6d;
else if (cc5 == 7'd104)
     in5 = 128'h3321203e1013632a390e6b1d183f616f;
else if (cc5 == 7'd105)
     in5 = 128'h3f58404b2717573d5c394d734e153b6e;
else if (cc5 == 7'd106)
     in5 = 128'h34696c3f1a6d1e763f7221634a5f516e;
else if (cc5 == 7'd107)
     in5 = 128'h28683002355c2f453a124369346e3e6d;
else if (cc5 == 7'd108)
     in5 = 128'h1004003c275c680a6a6154080d525f6f;
else if (cc5 == 7'd109)
     in5 = 128'h354505154961191b044f1c652e6f076e;
else if (cc5 == 7'd110)
     in5 = 128'h60416a524a2b495a70161b444016466e;
else if (cc5 == 7'd111)
     in5 = 128'h631b5c184a1c3843100d3e450d11286d;
else if (cc5 == 7'd112)
     in5 = 128'h1e7734746d4e24446e4a3b307615536f;
else if (cc5 == 7'd113)
     in5 = 128'h6704325058732c25567511660f5c586e;
else if (cc5 == 7'd114)
     in5 = 128'h2c15484063723e1e732f200a1d1e666e;
else if (cc5 == 7'd115)
     in5 = 128'h015355615d630f15281619420c3e1b6d;
else if (cc5 == 7'd116)
     in5 = 128'h021c707101484262173e38464e03426f;
else if (cc5 == 7'd117)
     in5 = 128'h436555614f716e50583f110b09764c6e;
else if (cc5 == 7'd118)
     in5 = 128'h101f6175306e370d482751327729286e;
else if (cc5 == 7'd119)
     in5 = 128'h564645145f6e6633004169395c174a6d;
else if (cc5 == 7'd120)
     in5 = 128'h0473280c29155b484141293f3721226f;
else if (cc5 == 7'd121)
     in5 = 128'h2e6929165d0948695d6f0543502d576e;
else if (cc5 == 7'd122)
     in5 = 128'h1749191c7634561f3f1a570c034a066e;
else if (cc5 == 7'd123)
     in5 = 128'h6d224613236863080267181051604 66d;
else if (cc5 == 7'd124)
     in5 = 128'h461f4735696b21316d15710c3c77246f;
else if (cc5 == 7'd125)
     in5 = 128'h264a71495619534112104414 1e370d6e;
else if (cc5 == 7'd126)
     in5 = 128'h686518710349243b4e236c250847196e;
else if (cc5 == 7'd127)
     in5 = 128'h2b282833622a62014f2e5837240d1a6d;


        #200 $stop;
        end


        initial begin
                $dumpfile("router5x5_128b_10000ps.vcd");
                $dumpvars;
        end
endmodule
```

9. REFERENCES

[1] Kim, J.; Dally, W.J.; Towles, B.; Gupta, A.K., "Microarchitecture of a high radix router," *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, pp. 420-431, 4-8 June 2005.

[2] A. Joshi, C. Batten, Y. Kwon, S. Beamer, I. Shamim, K. Asanovic, and V. Stojanovic. Silicon-Photonic Clos Networks for Global On-Chip Communication. *Submitted to the 3rd International Symposium on Networks-on-Chip (NOCS-3), May 2009.*

[3] James Balfour and William J. Dally. Design tradeoffs for tiled cmp on-chip networks. *International Conference on Supercomputing*, pages 187–198, 2006.

[4] C. Batten, A. Joshi, J. Orcutt, A. Khilo, B. Moss, C. Holzwarth, M. Popovic, Hanqing Li, H. Smith, J. Hoyt, F. Kartner, R. Ram, V. Stojanovic, and K. Asanovic. Building manycore processor-to-dram networks with monolithic silicon photonics. *High Performance Interconnects, 2008. HOTI '08. 16th IEEE Symposium on*, pages 21-30, Aug. 2008.

[5] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann, 2004.

[6] H. Wang, L. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. *Micro-36*, pages 105–116, 2003.

[7] H. Wang, L. Peh, and S. Malik. A Power Model for Routers: Modeling Alpha 21364 and Infiniband Routers. *IEEE Micro*, pages 27-35, 2003.

[8] Andrew Kahng, Bin Li, Li-Shiuan Peh and Kambiz Samadi. ORION 1.0 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In Proceedings of *Design Automation and Test in Europe (DATE)*, Nice, France, April 2009.

[9] Amit Kumar, Partha Kundu, Arvind Singh, Li-Shiuan Peh and Niraj K. Jha. A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS. In *25th International Conference on Computer Design*, October 2007.