DESIGN AND INTEGRATION OF HIGH SPEED RELATIVE TIMED NETWORK-ON-CHIP ROUTERS

by

Dheeraj Singh Takur

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

The University of Utah

August 2016

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Dheeraj Singh Takur**

has been approved by the following supervisory committee members:

Kenneth S. Stevens	, Chair	**04/28/2016** Date Approved
Erik Brunvand	, Member	**04/28/2016** Date Approved
Priyank Kalla	, Member	**04/28/2016** Date Approved

and by **Gianluca Lazzi** , Chair/Dean of

the Department/College/School of **Electrical and Computer Engineering**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

Integrated circuits often consist of multiple processing elements that are regularly tiled across the two-dimensional surface of a die. This work presents the design and integration of high speed relative timed routers for asynchronous network-on-chip. It researches NoC's efficiency through simplicity by directly translating simple T-router, source-routing, single-flit packet to higher radix routers.

This work is intended to study performance and power trade-offs adding higher radix routers, 3D topologies, Virtual Channels, Accurate NoC modeling, and Transmission line communication links. Routers with and without virtual channels are designed and integrated to arrayed communication networks. Furthermore, the work investigates 3D networks with diffusive RC wires and transmission lines on long wrap interconnects.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor Dr. Kenneth Stevens for his continuous support, motivation, and comments throughout my MS study. His guidance helped me throughout my research. He is one of the best mentors of all time.

Secondly, I would like to thank my committee members Dr. Priyank Kalla and Dr. Erik Brunvand for their valuable suggestions and continuous encouragement.

I would also like to thank all the members of CMOS VLSI research group (Tannu, Jotham, Dipanjan, William Lee, and Shomit) especially Mac Wibbels for their friendship, encouragement, and collaboration. I would like to express my sincere gratitude to my friends Divya, Baji Babu, Dileep, Rama Mohan, Nikhilesh, Phani, Charan, and Praveen for bearing with me and helping me in these two years.

Last but not the least, I thank my parents and my brothers for their love, support, and belief in me throughout my MS study here at the University of Utah.

CHAPTER 1

INTRODUCTION

Today's complex System on Chip (SoC) designs have been using buses for communication between IP cores. Various buses have been built either for a specific set of applications or for general purpose processor designs. Single chip complexity is on the rise as we transition from the multicore to the many-core era, as shown in Figure 1.1. Innovations in system architecture design has been driving the evolution in bus architectures. Bus architectures cannot closely align to process evolution, nor system architecture evolution. This section throws some light on the advancements in SoC design and reasons to change the way designers think about on-chip communication.

1.1 Advancements in digital system design

Moore's law has been the driving force for the newer SoC designs to integrate various heterogeneous IP blocks such as processor, memory, FFT, etc., onto a single chip. Processing more data, which was once considered a performance bottleneck, has now been achieved due to new approaches such as parallel processing on chip using multiple processors.

Buses have played a major role as data communication channels on a digital system. They have been serving as the backbone of an SoC, sharing resources among various IP blocks. Newer bus architectures have provided the designers with flexibility to customize an SoC by simply plug and use IP-based system, for intended performance requirements [1].

However, history shows that there have been conflicting trade-offs in terms of IP block reuse strategies and evolution in bus architectures.

1.2 Limitations of bus-based approaches

Buses support a shared resource type of communication. They work on two key properties - serialize and broadcast the messages. They work in terms of sets of lines where they have one common line which breaks down into multiple lines with each line going into an

IP block. The common line serves as the backbone of a bus broadcasting a message to all its components.

Moore's Law has enabled integration of many IP blocks onto a single chip. However, limited fanout and other electrical constraints on the buses have been the major hurdle for the shared type of communication they support, among many IP blocks. Serializing the messages on buses also has an impact on the performance of an SoC which has many IP blocks communicating with one another. Also, global wire delays do not scale well with technology scaling, which has been the major problem in design of an SoC [2, 3].

1.2.1 System architecture

Integration of newer components into a system has always come with changes to bus interfaces. There have been cases requiring upgrades to bus architectures such as AMBA ASB to AHB2.0, then AMBA AHB-Lite, then to AMBA AXI. Such changes in bus architectures have a huge impact in IP re-usability or in designing a new IP block. Changes to bus physical implementation can also have serious effects on the implementations of higher-level bus behaviors [4].

General bus activities such as transaction, transport and physical layer behaviors cannot be separated. For example, bus protocols such as OCP2.0 and AMBA AXI socket standards have been developed to support transaction types which have exclusive accesses [4].

Every new system architecture comes with a requirement of a new transaction type or characteristics. This hinders the adaptation of buses to changes in the system architecture, or take advantage of the rapid advances in silicon process technology. For example an out-of-order response type system architecture has a complicated data transaction type [4]. This change has huge impact on bus architectures, which can handle transactions only by addition of wires.

1.3 Motivation

Process scaling supports the implementation of many processing cores on a single planar die. Distributed memory multiprocessors are an important class of such multiprocessor designs [5, 6, 7, 8]. These systems consist of arrays of a local microprocessor and associated memory, which communicate via a mesh-based Networks-on-Chip (NoC) communication fabric. NoC have been studied in great detail in the past and there is adequate understanding about NoC implementation choices such as topology, routing techniques, switching methods, flow control, etc. They have been researched as a solution for efficient on-chip communication as they closely follow both process and system architecture evolution. They also support systematic integration of heterogeneous IP blocks to enable use of the SoC design methodology. The network fabric supports point-to-point communication when compared to a bus's serial communication [9]. They can effectively handle multiple processing elements (PEs) communicating among each other with high fidelity while consuming little energy and offering low latency [4].

The size of the network array is defined by n, the number of processors on each edge of the array. The planar nature of the die maps well to regularly spaced two-dimensional processing and NoC arrays such as a square 8×8 array containing 64 processors and memory banks. Each node will communicate with its four adjacent nodes in the array. The hop count is the number of routers a message must traverse when communicating between source and destination nodes. The worst case hop count for a square planar mesh unconnected at the periphery is *2(n-1)*, illustrated in Figure 1.2(a).

The edges of an array can be connected by adding wrap wires, as shown in Figure 1.2(b). This creates a three-dimensional interconnect where the wrap wires create a toroidal NoC structure. Adding wrap lines doubles the network bisection but also can reduce the worst case hop count between any two nodes on a mesh by a factor of two to *n-1* when the wrap lines are twisted, as shown in Figure 1.2(d).

While 3D torus routers reduce the worst case hop count, they introduce several complications. First, the wrap lines invalidate the deadlock free property of common mesh-based X-Y routing algorithms [10, 11, 9, 12]. Cycles are formed in the X and Y planes that do not exist in a mesh, and these cycles produce deadlock scenarios. Second, these structures are difficult to map onto a two-dimensional plane. Simply connecting the edges of a planar array of nodes as shown in Figure 1.2(b) results in short local interconnect between nodes on the plane, but long wrap wires that are *n-1* times longer than the in-plane connections. This results in two very different delay values depending on which type of interconnect is part of the path, be it local or wrap lines [13]. One solution is to create a Folded Torus as in Figure 1.2(c). This creates communication links that are all similar in length, delay, and energy by penalizing the wire delay of the short links.

This work directly addresses the design and performance issues that arise in toroidal interconnection networks. It addresses the challenges of the toroidal deadlock scenarios by introducing virtual channels which break the X and Y cycles of the mesh [14]. A handshake-based virtual channel design approach is introduced which allows efficient sharing of the interconnect between virtual channels. A new approach to designing torus routers is presented which can create performance and energy efficiencies. The mesh-based routing cores with short interconnects between each core are implemented with traditional diffusive wires. The long wrap lines are implemented with transmission lines, which gain their power and performance advantage only after approximately two mm of wire length [13].

A flexible simulation model has been designed that allows us to obtain results with very high accuracy for performance and power. The model is based on performing Verilog simulations of the actual hardware under evaluation. This design used a cell-based design methodology for the routers. The test bench reported in this work runs simulated traffic on an 8×8 tiled system consisting of 64 router nodes.

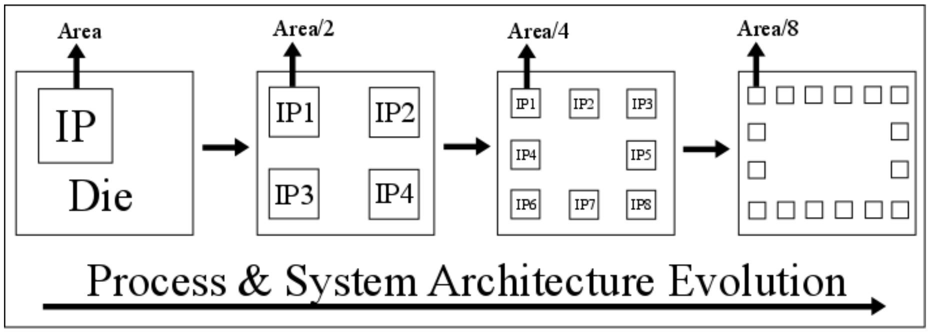
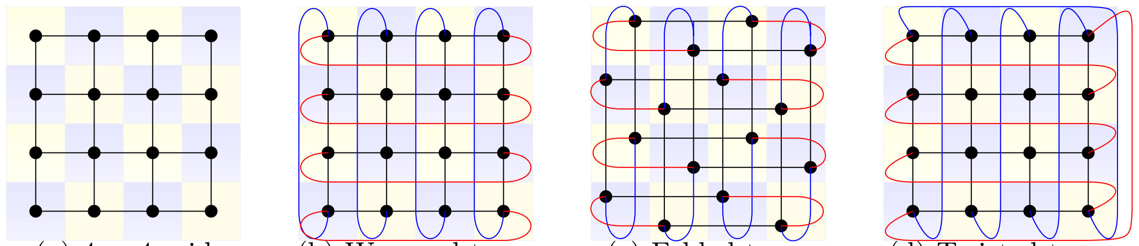


Figure 1.1. Single core to many core transition on a die



(a) 4 × 4 grid (b) Wrapped torus (c) Folded torus (d) Twisted torus

Figure 1.2. Network topologies

CHAPTER 2

BACKGROUND

Contemporary NoC designs prefer mesh topologies due to their simplicity, ease of physical mapping, and use of short wires [15]. These topologies have a strong correlation between logical minimality and physical proximity, making them perfectly suitable for applications with mostly local traffic. However, they suffer from having a larger hop count, latency, and lower path diversity. Designers have tried to solve this problem by using high radix routers to build complicated topologies such as flattened butterfly, folded clos, and dragonfly [9]. While these hierarchical topologies work very well as the underlying networks in high-performance supercomputers, they are expensive to implement in a power constrained on-chip system due to the complexity associated with the routers.

The torus offers advantages from both domains. It maintains the design and routing simplicity of meshes, while providing connectivity between nodes at the extremities. Compared to the mesh, tori offer twice the channel bisection, half the channel load for uniform as well as worst case traffic, and 24% lower hop count [9]. The concept of replacing long diffusive wires with alternative equalized interconnects on low diameter networks has been studied [16]. In contrast, this work utilizes the regularity of a torus using 5-port routers, but replaces the global channels with low latency transmission lines [13].

2.1 Asynchronous circuit basics

Asynchronous communication between two entities is generally indicated by transitions on a request and an acknowledgement signal [17]. These transitions dictate start and end of a valid token (data) through proper ordering. Asynchronous protocols normally fall into two categories: Quasi Delay-Insensitive (QDI) and Bundled-Data (BD). Generally, QDI is more robust to variations while BD allows simpler circuits. BD has a lower wire count compared to QDI's common encodings (e.g. 1-of-4 and dual-rail). This is potentially more energy-efficient due to reduced wire repeater leakage, especially with wide links [18].

2.2 Asynchronous design methodology

Bundled-Data asynchronous approach is preferred, with simple network components and switching/routing algorithms. This work uses relative timing (RT) methodology as the sequencing technique for the design of all network components [17, 19, 20]. The RT-based design flow allows the designer to specify, characterize, and validate clockless elements using synchronous CAD tools. Instances of these characterized blocks can then be inserted in the regular VLSI CAD flow with some additional timing constraints.

2.3 Asynchronous NoC design

There are a number of examples of clockless interconnection network designs [21, 22, 23, 24, 25, 26, 27, 28, 29]. This work develops a NoC architecture that provides connection oriented guaranteed service and connection-less best effort routing [23, 30]. Router design for multiple service levels with equal priority virtual channels within each level are demonstrated [31, 32]. Three port lightweight routers for binary fan-in, binary fan-out Mesh-of-Trees (MoT) networks are presented in [33]. Another approach adds limited dynamic reconfiguration capabilities to the network by identifying critical routes and bypassing arbitration on them [34].

An early arbitration resolution using a lightweight monitoring system for MoT networks is proposed in [35]. The 2-phase Bundled-Data design for a 5-port router suitable for mesh networks have been implemented [36]. Leakage reduction techniques are applied to this router design in [37]. The lightweight monitoring idea for early arbitration and channel allocation is applied to the 5-port router design in [38]. Asynchronous channels have been combined with synchronous routers to enable multihop channel reservation and router bypassing [39]. A hybrid design of 4-phase Bundled-Data routers with 2-phase channels is offered in [40]. Recently, Intel demonstrated a fabricated hybrid circuit/packet switched NoC design that employs handshake-based communication [41]. A time division multiplexed (TDM) asynchronous NoC architecture is presented that allows message passing among processor nodes in a multiprocessor environment [42]. However, none of these designs present a method of non-blocking arbitration that enables deadlock-free routing in tori configurations.

An RT constraint consists of a common timing reference called point-of-divergence *(pod)*, and a pair of events that are ordered in time for correct circuit operation called the point-of-convergence *(poc)*. A constraint is represented as $\text{pod} \mapsto \text{poc}_0 + \text{m} \prec \text{poc}_1$, where poc_0 must occur in time before poc_1 with margin m. Hence the maximum path delay from *pod* to poc_0 must be less than the minimum path delay from *pod* to poc_1. This is easily represented by two related design constraint equations *set_max_delay* and *set_min_delay*, which perform timing driven synthesis that enforce the constraints on the logic paths. The complete RT design flow is shown in Figure 2.1.

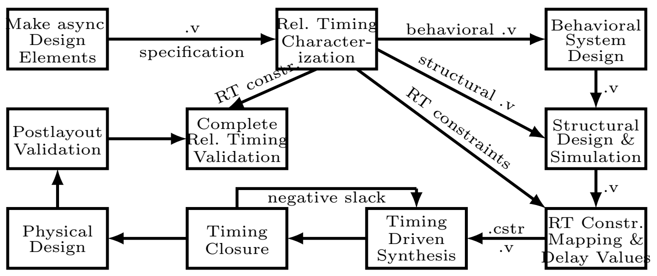


Figure 2.1. Simplified relative timing multisynchronous design flow

CHAPTER 3

ASYNCHRONOUS NOC ARCHITECTURE

Our NoC design is intended for efficiency through simplicity. Network functions performed by simple circuits result in power savings. Keeping this in mind, the network is designed to be minimal in structure as well as function. The network uses basic dimension ordered source routing, single flit packets, and simple high throughput and low latency router design. Instead of large crossbars that are energy and area inefficient, a simple demultiplexor-based direction decoding technique is employed. This work does not include advanced NoC features such as adaptive routing, router bypassing, and quality-of-service guarantees.

3.1 Early work

Early work on the router design was done by Junbok You [43]. His work focused on developing T-Routers (3 port routers) where each port constitutes a Switch and Merge module. Each Switch and Merge Port is connected with two other ports. His work developed 2-phase protocols on the node links and 4-phase for the internal logic of the router. T-routers natively implement Tree network topologies, but can be configured for larger radix networks. His work employed source routing for single flit packet routing and examined wire delays using spice simulations and efficiency of the router using different asynchronous communication protocols (4-phase and 2-phase).

This work studies the performance and power trade-offs by adding high radix routers, 3D topologies, virtual channels, accurate NoC modeling, and transmission line communications links.

3.2 Asynchronous router design

The router shown in Figure 3.1 forms the core component of the NoC and is composed of five ports. Each port comprises of a switch and a merge module. Each switch and merge module provides a single flit buffer on each input and output port with a flit size latch. Note that there is no difference between a flit and a packet and they are used interchangeably in this thesis. The switch directs a flit to one of four merge input ports and one of the four merge modules receives the flit and directs it to the output port.

The circuits employed by the routers are purely asynchronous and are built using the RT methodology shown in Figure 2.1. Design choice is in using a 2-phase protocol versus a 4-phase protocol. The mutual exclusion elements used for arbitration require 4-phase protocols. Thus we operate 4-phase internally rather than just translate back and forth at the routers. Also the throughput across long links is limited by wire delay. The 4-phase protocols present twice the link delay as compared to 2-phase protocols and almost half the bandwidth. In contrast, level sensitive 4-phase circuits are smaller, less complex, and more energy efficient as compared to edge triggered 2-phase logic circuits [44, 45].

To get the best of both protocols, routers are designed to operate using a 4-phase BD protocol internally, using 2-phase BD protocol on links for communication with other nodes in the network. Circuits used for the translation between the protocols (4-phase to 2-phase converter) are included as part of the router design [43].

3.2.1 Switch module

Switch module, as shown in Figure 3.2, is an address decoding circuit, which decodes the direction of the incoming flit to the router, based on the address bits in the flit. The circuit consists of a 2- to 4-phase converter, a Linear Controller (LC) to latch the data and forward the request, and a 1 to 4 demultiplexor built using 3-input NOR gates [46, 47, 48].

The 2- to 4-phase converter takes in the request signals coming from an adjacent network node (router) over the links (wires) and converts into a 4-phase signals *(lr_m, la_m)* which directs the request to a linear 4-phase controller (LC). The linear 4-phase controller is a burst mode controller with data valid on the rising edge of the left request *(lr_m)*. The right request *(rr_m_)* from the controller gets forwarded to the demultiplexor (all the 3-input NOR gates).

The incoming flit or packet on the data path is divided into two main parts − network address and payload. The demultiplexor is a very simple address decoding circuit based on incoming packet. It uses the first two most significant bits (MSBs) from the packet which are part of the network address to determine the direction of the packet. The *rr_m_* signal is the input to a 1 to 4 demultiplexor and the first two MSBs from the packet serve as select lines. The latch used in the design is of the size of the incoming packet where the first two MSBs of the latch serve as a transparent latch making the direction selection bits arrive to demultiplexor before *rr_m_* arrives, speeding up the circuit functionality. The remaining part of the latch is normally closed, and is controlled by a clock from LC.

The switch module employs a simple and efficient bit rotation or bit swizzling logic, where the data bits used in determining the direction are rotated back to the very end of the network address in the packet and the next set of routing bits are pushed forward to the first two MSB positions. The packet gets further swizzled in its next hop node (next router's switch module) which it goes through on its way to its destination node. This rotation of bits also directly encodes the return address. The packet doesn't require any additional bits for the PE connected at the destination node to send a packet to the source node. It uses the same network address at the destination after the final swizzling operation, by reversing and inverting them. This operation makes the network free of any additional routing logic from destination to source, making it a very light weight design.

3.2.2 Merge module

The merge module, shown in Figure 3.3, is an arbitration circuit which arbitrates and selects among the packets coming from four other switch ports within a router, shown in Figure 3.1. The arbitrarily selected packet is sent to the output port of the router which goes onto the next node in the network through the links.

The merge module consists of a 4-input mutually exclusive element (MUTEX), 4-input tri-state multiplexer (MUX), and a 4-phase to 2-phase converter [43]. The MUTEX element works as an arbiter serializing the requests coming into the merge module to a shared output channel. It also serves as a select line to the 4-input MUX. The MUTEX element, which arbitrates between input channels, is a level sensitive 4-phase circuit [44, 45]. Thus, the internal working logic of these asynchronous NoC routers are designed using a 4-phase protocol.

The MUX, shown in Figure 3.3, has data channels coming from four other switch modules in the router. The data line is channeled to the output of the multiplexer based on the request signal passing through the MUTEX to the select line of the MUX. This select line also dictates which input data line to be stored on the latch, present at the output of the merge module.

Every request on the input of the MUTEX presents a new flit on the data channel, to be stored and forwarded to the next node through the latch and 4- to 2-phase converter. Each transaction on the rising edge of input *(lr0, lr1, lr2, lr3)* through the MUTEX circuit forwards a request to transfer data through the 4-phase handshake signal *lr*. The 4-phase handshake signals *lr* and *la* gets translated to a 2-phase network link handshake on signals *rr* and *ra* through a 4- to 2-phase converter circuit, as shown in Figure 3.3. It also stores the data line selected through the MUX onto the output latch.

Every rising edge of the inputs *(lr0, lr1, lr2, lr3)* at the MUTEX goes through C-Elements to the phase converter. An acknowledgement *(la)* is sent back to the corresponding switch module, whose request was processed through the MUTEX, through one of the four output signals *(la0, la1, la2, la3)*. On the falling edge of the request at the input, the circuit acknowledges through the outputs of the MUTEX *(rr0, rr1, rr2, rr3)* decreasing the delay in the 4-phase protocol considerably. The C-Elements are used to maintain the state of the current request until the acknowledgement is seen by the rest of the components on the rising edge of request. On the falling edge of the request, the C-Element prevents any new request coming to the converter, until the internal logic is reset correctly (controlled by *la*). This circuit results in high performance due to different rising and falling acknowledgement paths.

3.3 Network topologies

The primary goal of this work is to build a 64 node NoC arranged in an 8×8 array. There are various topologies of NoC that can be built using 8×8 array. The scope of this thesis is limited to building mesh, torus, and folded torus topologies, as shown in Figure 1.2.

3.3.1 Packet routing

Every packet works at flit granularity and contains routing bits (network address bits), and payload in parallel as a single flit. The router used in building these topologies has five ports, thus enabling a k-ary n-dimension cube type topology. Source routing is employed for its routing mechanism because of its inherent advantages of speed, simplicity, and scalability [9]. The number of network address bits depends on the network diameter. They are determined by the maximum hop count of a network generated for a specific topology. The hop count is the number of routers a message must traverse when communicating from source to destination nodes. There are formulae to determine this value for regular topologies like mesh and tori [9]. The payload width of the link depends on the required throughput, power, and area budgets.

Source routing requires all routing decisions to be made a priori at the source. Lookup tables are used to store a set of precomputed routes for each sender-destination pair. A single route is calculated for each combination. The routing bits decide the direction of steering the packet at the switch module; no other computations are performed at the router. At each hop, the two most significant address bits are rotated, and the next two routing bits appear at the most significant positions. No return address is required; it is directly derived from the source route.

The NoC does not cater to any particular transaction layer protocol, such as OCP. Every packet requires routing bits, thus contributing a significant overhead for large dimension networks. This means that a network that employs fewer hops such as a torus immediately has an advantage over a regular mesh.

The router has five ports and four destinations for each inbound packet. The three routing channels are fixed, but the router knows that if the address is for the hardwired port, it sends the packet to the local node. This is done by having the last two bits of the address as an inverse of the final two route bits. The packet knows when it has arrived at its destination if it is directed to go back to the same direction from which it arrived.

The XY source routing is used in the network for routing the packets [9]. Source routing avails the use of a cost function to determine the best possible route for a message. This value can simply be a hop count, or a more complex metric involving wire and router delays. Energy expense can also be a factor in the cost function. The NoC uses packet hop count as the cost function metric. For mesh networks, XY source routing provides one single route, whereas for toroidal networks, there can be more than one route possible. The shorter hop count route is chosen for toroidal networks among multiple routes to route packets.

3.3.2 Mesh topology

A mesh is a 2-dimensional network topology shown in Figure 1.2 (a). An 8×8 array-based 64 node mesh network is built. Each node constitutes a 5-port router from Figure 3.1. The maximum diameter or the worst case hop count for a square planar mesh is determined by the formula *2(n-1)* where *n* determines the number of nodes in each edge of the network. 8×8 Mesh network with $n=8$ makes the maximum diameter or hop count to be 14. Since source routing is used, each flit has 30 network address bits, with two address bits required for each hop and the last two bits for exciting the network into an PE at the destination. A fixed size of 32 data bits and 10 address bits is used as the payload. A total flit size 72 bits for each packet (address + payload) is used.

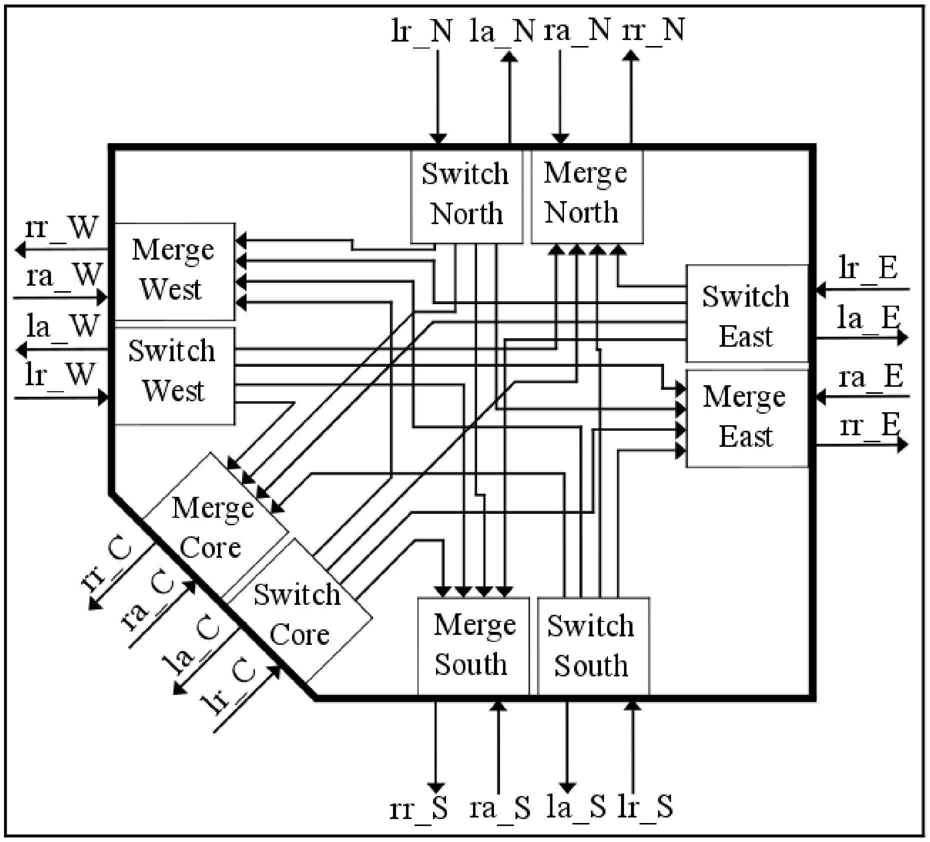


Figure 3.1. Five port router

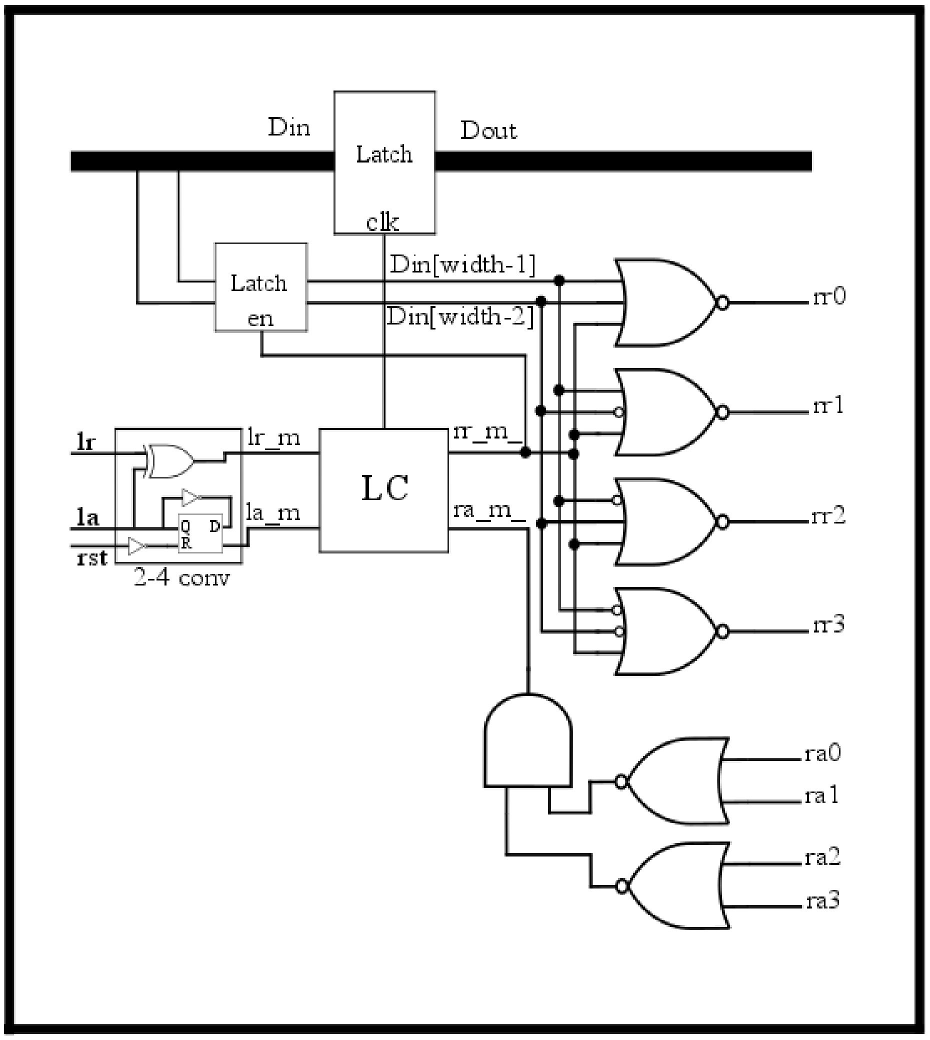


Figure 3.2. Switch module

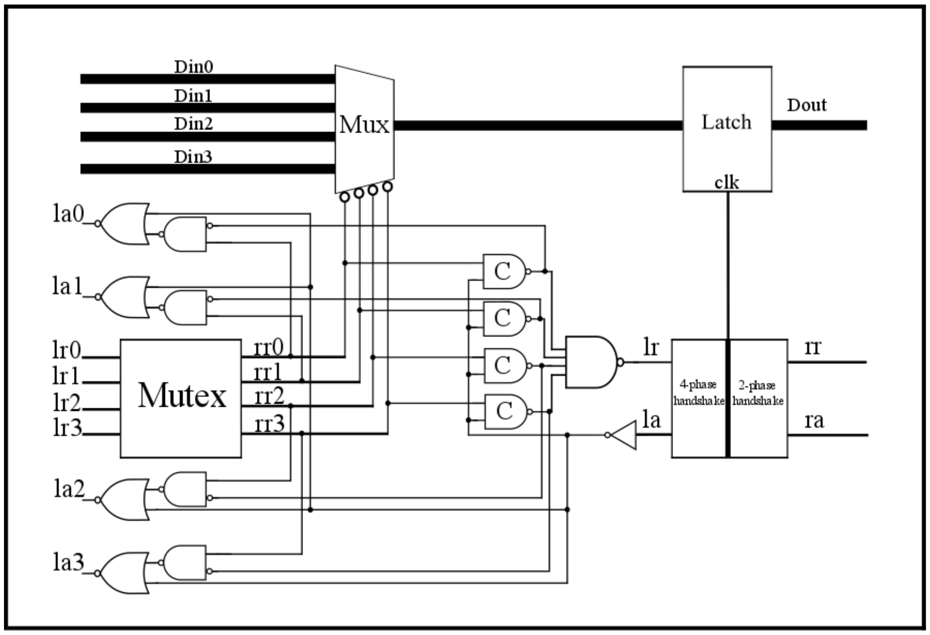


Figure 3.3. Merge module

CHAPTER 4

ASYNCHRONOUS NOC WITH VIRTUAL CHANNELS

Our 2-dimensional network topologies use an XY routing algorithm as the fundamental routing algorithm to route packets. If applied to a network with higher dimensions, the resulting network cannot be guaranteed to be free of deadlocks [9]. The torus topology, for instance, shown in Figure 1.2(b,c,d), has interconnects over the edges (third dimension). These wrap lines will result in a possible deadlock state, if implemented using the existing router shown in Figure 3.1.

4.1 Deadlocks

With XY routing, it is possible for a set of packets to have a cyclic dependency of resources in 3-dimensional networks. Each of the packets waits on a port held by another and none of them can advance. Such a scenario is called a routing level deadlock [9]. Deadlock can also described, when there is a cycle of resource dependencies where a process holds on to a resource (A) and attempts to acquire another resource (B) but A is not relinquished until B is acquired. A deadlock scenario in toroidal networks can occur over the long wrap lines using the XY routing algorithm, shown in Figure 4.1.

4.2 Deadlock avoidance with virtual channels

Deadlocks can be avoided by eliminating or breaking the cycle of dependencies in the network. Deadlock avoidance techniques mainly depend on the way the resource allocation is done in a network [9]. Deadlock free routing can be achieved by proper ordering of the resources. One of the ways resource ordering can be done is by restricting the physical routes for a packet in a network using specific routing techniques. A few very good examples of such routing techniques include *dimension ordered routing* and *turn model* [9]. This work controls resource ordering by multiplexing one external channel to n internal channels within a router, as show in Figure 4.2. Each of these multiplexed channels is called *Virtual Channels*

[49]. Each port of the 5-port router is integrated with two virtual channels, VC0 and VC1. Channel multiplexing within the router is done at the flit granularity.

4.3 Asynchronous router with virtual channels

A router with virtual channels is a possible solution to avoid deadlocks in tori-based network topologies. A 5-port router with virtual channel support, shown in Figure 4.3, is developed as a part of this work. The circuits employed by the router are completely asynchronous. Every port of the router, other than the port to the core, has a negative acknowledgement (or a nacking) signal and integrated virtual channels. The words negative acknowledgement and nacking have been used interchangeably in this thesis. Since the router supports virtual channels, every router port, except the core port, used in forming the network is built as shown in Figure 4.4.

4.3.1 Switch module

The switch module, as shown in Figure 4.5, is an address and channel decoding circuit. The circuit decodes both the direction of the incoming flit to the router and the channel which the packet has to traverse. The circuit is comprised of four major blocks - (1) A 2- to 4-phase converter with nack (ln), (2) A nacking arbiter (NARB), (3) A channel selection latch, and (4) A 1 to 4 demultiplexor with 3-input NOR gates. The incoming flit or packet on the data path is divided into three parts: channel bit, network address, and payload.

The 2- to 4-phase converter takes in the request signals coming from the longer links (wires) and converts them into a 4-phase signals. The 4-phase request signal *lr_m* is directed to the 4-phase NARB through the 2-input MUTEX. The MUTEX creates mutual exclusion between left request *(lr_m)* and right acknowledgement *(ra)* signals. This mutual exclusion in turn helps the NARB in deciding to generate an left acknowledgment *(la_m)* or a negative acknowledgment *(ln_m)*.

The channel bit is the MSB bit of the flit, which goes through an always open latch. The latch's clock is controlled by all the left positive and negative acknowledgment signals going through an 4-input NOR gate. This clock doesn't allow a new request to propagate through any channel until a new request is received on the input port and there are no existing left acknowledgment/nacking signals. The channel bit together with *lr_m* decide to which channel NARB, the request has to be forwarded.

The 4-phase NARB is a burst mode controller with data valid on the rising edge of the left request *(lr_m)*. The right request $\overline{rr\_m}$ from the NARB gets forwarded to the demultiplexor (all the 3-input NOR gates). The $\overline{rr\_m}$ signal also clocks an always open flit latch. The demultiplexor used here is the same as the one from Figure 3.2.

Latches in the switch are controlled in the always open mode. Data directly passes through them to the output. This data waits for the appropriate request signal to get forwarded. The channel bit is placed in the MSB position of the packet and is present at the channel NAND gates. The input request signal *lr* goes through a 2- to 4-phase converter to generate *lr_m*. *lr_m* then goes through a pair of 2-input MUTEXs. The MUTEX forwards the signal to the NAND gates based on the state of corresponding *rac0* or *rac1* signal. The request then goes through the corresponding NARB based on the channel selection bit ready at the other input of NAND gate. The NARB processes the request and generates a right request signal $\overline{rr\_m}$ to the 1 to 4 demultiplexor, which also latches the data. The request and data are then forwarded to corresponding merge modules shown in Figure 4.4, based on the channel which the data got processed on, through the demultiplexing logic.

The left acknowledgment *(la_m)* or a negative acknowledgment *(ln_m)* are generated by the NARB based on the availability of the channel. These 4-phase signals pass through a 2- to 4-phase converter and get converted into *la* and *ln* 2-phase signals, respectively.

4.3.1.1 2- to 4-phase converter

The 2- to 4-phase converter shown in Figure 4.5 is equipped with a nacking signal *(ln)* which toggles when the block receives a new request on *lr* input before the previous request was processed. This signal is generated by NARB. The signal transition order for the converter is shown in Figure 4.6. Here *la_m* triggers *la* and *ln_m* triggers *ln*.

4.3.1.2 Non-blocking (nacking) arbiter

The non-blocking arbiter (NARB), shown in Figure 4.7, is a 4-phase controller with a nacking logic. This circuit works on the principle of a 4-phase Bundled-Data protocol. The circuit takes in a new request on *lr* input and forwards the request to the next stage through $\overline{rr}$. This circuit generates a *ln* signal only when the there is a new request on *lr* before my right acknowledgment signal *ra* for the previous data arrives. All the other times, it works the same way a LC works. The signal transition order for NARB is shown in Figure 4.10. The NARB behavioral specification written in CCS is shown in Figure 4.8. The relative timing constraints (RTC) that enable hazard free operation and better timing optimization for asynchronous circuits are shown in Figure 4.9.

4.3.2 Merge module

The merge module, shown in Figure 4.4, is an arbitration circuit, arbitrating requests for both channels and packets. The merge port in router Figure 4.3 is composed of two blocks, the merge virtual channel and the merge pipeline.

4.3.2.1 Merge virtual

The merge module receives request and data input from the switch module. The merge virtual module, shown in Figure 4.11, forms one of the virtual channels to the merge port of the router, shown in Figure 4.3 and Figure 4.4. The working principle of the merge virtual module is the same as that of the merge module described in Section 3.2.2. The only difference lies in control logic and data latching mechanism. The merge virtual control logic *(LC)* uses a normal 4-phase handshake controller described in Section 3.2.1, to both send a request forward through *rr* to the merge pipeline and also latches the data onto the output.

4.3.2.2 Merge pipeline

The merge pipeline, shown in Figure 4.12, is a channel arbitration circuit. This module receives requests from two virtual channels, shown in Figure 4.11, and forwards one of the request using a 2-input MUTEX to the output port. It consists of a 2-input MUTEX, 2-input MUX, and a 4- to 2-phase converter.

The tri-state multiplexer, as shown in Figure 4.12, has data channels coming into it from the output of two virtual merge modules in the router. The data line is channeled to the output of the multiplexer based on the request channel passing through the MUTEX to the select line of the multiplexer. This select line also dictates which input data line to be stored on the latch in merge pipeline module.

Every transaction through the MUTEX circuit forwards a requests to transfer data through the 4-phase handshake signal *lr*. The 4-phase handshake signals *lr*, *la*, and *lna* get translated to a 2-phase network link handshake on signals *rr*, *ra*, and *rn*, through a 4- to 2-phase converter shown in Figure 4.12. It also stores the data line selected through the tri-state multiplexer onto the pipeline latch. The handshake signals *rr*, *ra*, and *rn* along with data are then forwarded to the next node in the network.

In an asynchronous design, a request waits for an acknowledgment forever until it receives it, to proceed to a new request. Even with the virtual channels, there is a possibility of a deadlock scenario when virtual channel 1 waits on virtual channel 0 to propagate which in turn waits for virtual channel 1, forming a cyclic dependency on long wrap lines. A negative acknowledgement signal from NARB is used in lowering the request to one of the channels which got processed first in the 2-input MUTEX, thus allowing the other channel to flow through and breaking the cycle.

4.3.2.3 4- to 2-phase converter

4- to 2-phase converter, shown in Figure 4.13, is a pipeline controller which converts a 4-phase signal to a network link 2-phase signal. The 4- to 2-phase converter takes in the the request signals coming from the virtual merge channel and converts into a 2-phase signals. The signal transition order of the converter is shown in Figure 4.14 and RTC in Figure 4.16.

The converter gets a new request signal on lr input and forwards the request to $\overline{lra}$ or $\overline{lrn}$ based on la or ln signal. The request goes through the 4- to 2-phase state machine. The state machine converts the 4-phase signal to 2-phase right request signal (rra or rrn). The NAND gates on the input of the converter make sure either rra or rrn are triggered based on input request lr and acknowledgments signals la or ln. The right request signals rra and rrn go through an XOR gate to generate rr, which is forwarded onto the network link.

Based on ra and rn toggling for the corresponding rr signal, the 4- to 2-phase state machine triggers either la and ln signal. Data are latched only when la is generated for a corresponding lr. The clock to latch is controlled by three signals: $\overline{lrn}$, $\overline{ck0}$, and $\overline{ck1}$. The latch is opened when $\overline{ck0}$ or $\overline{ck1}$ triggered, and closed when $\overline{lrn}$ or when a ra arrives. $\overline{ck0}$ triggers when the state of ra signal is *logic* 0 and $\overline{ck1}$ for *logic* 1.

The 4- to 2-phase converter circuit is shown in Figure 4.15. The MEAT specification is shown in Figure 4.19. The CCS specification and RTC for the circuit are shown in Figure 4.17 and Figure 4.18, respectively.

4.4 Packet routing

The packet routing technique is the same as described in Section 3.3.1. The only difference lies in packet size and classification. Each packet works at flit granularity and contains channel bit, routing bits (network address bits), and payload in parallel as a single flit. The channel bit is used to select one of the virtual channels in the switch module. Every packet is injected with channel bit 0 in its MSB position placing the packet on virtual channel 0. The only place where the channel changes or shifts to virtual channel 1 is over wrap lines where it forms a cycle in the network. Cycles create deadlocks in toroidal topologies, described in detail in Section 4.5.

4.5 Torus network topologies

A torus is a 3-dimensional mesh network topology, as shown in Figure 1.2, which has wrap lines over the edges of the network. An 8×8 array based 64 node torus network is built, where each node constitutes a 5-port router from Figure 4.3. Adding wrap lines doubles the network bisection but also can reduce the worst case hop count between any two nodes on a mesh by a factor of two. The maximum diameter or worst case hop count is determined by the formula *(n-1)* when n is odd and n itself when even. The number of nodes in each edge is identified by n. For an 8×8 torus network, $n=8$ which makes the maximum diameter or hop count to be 8.

Source routing requires all routing decisions to be made a priori at the source. Lookup tables are used to store a set of precomputed routes for each sender-destination pair. A single route is calculated for each combination. The routing bits decide the direction of steering the packet at the switch module and channel bit for sending the packet to the appropriate channel. No other computations are performed at the router. At each hop, the two most significant address bits next to channel bit are rotated, and the next two routing bits appear at the *(MSB-1)* and *(MSB-2)* positions. No return address is required; it is directly derived from the source route. Source routing avails the use of a cost function to determine the best possible route for a message. This value can simply be a hop count, or a more complex metric involving wire and router delays. Energy expense can also be a factor in the cost function. For the sake of simplicity, NoC uses packet hop count as the cost function metric.

Since source routing is used, each flit has 18 network address bits, with 2 address bits required for each hop and the last 2 bits for exiting the network onto a PE at the destination. Fixed payload size of 32 data bits plus 8 address bits are used as the payload every packet. A total flit size of 59 bits used for each packet *(Channel+address+payload)*.

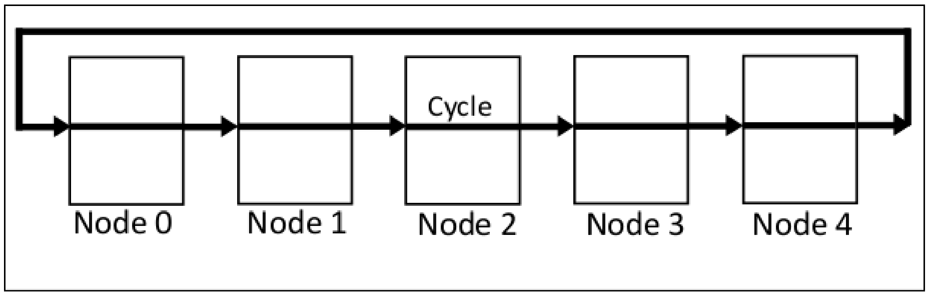


Figure 4.1. Cycle in toroidal network over the long wrap lines

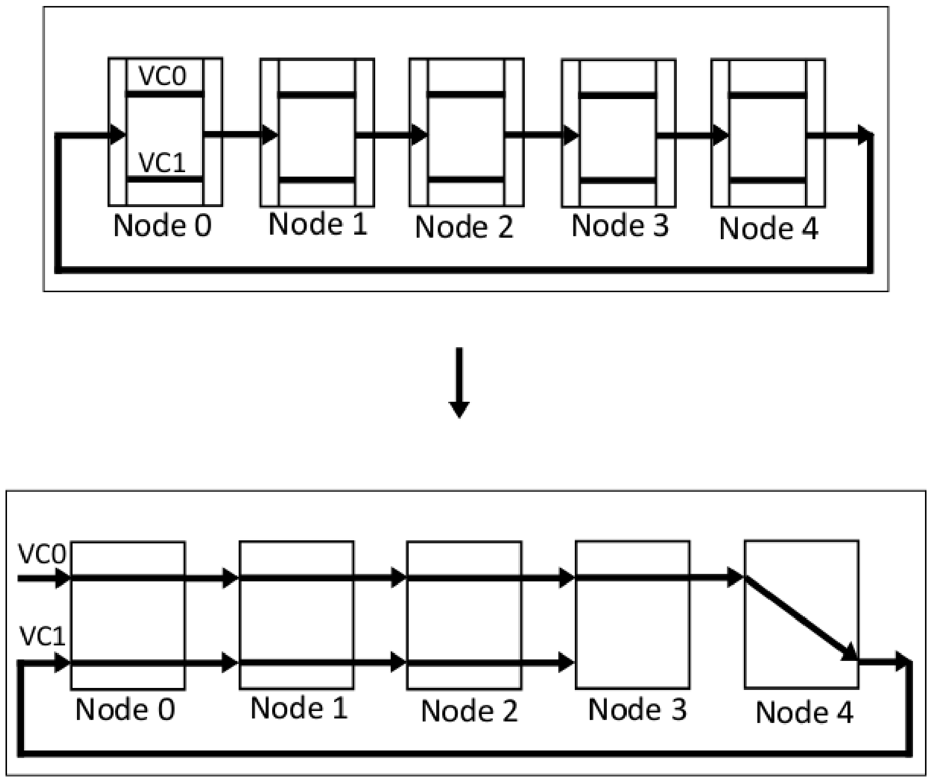


Figure 4.2. Breaking cycles using virtual channels

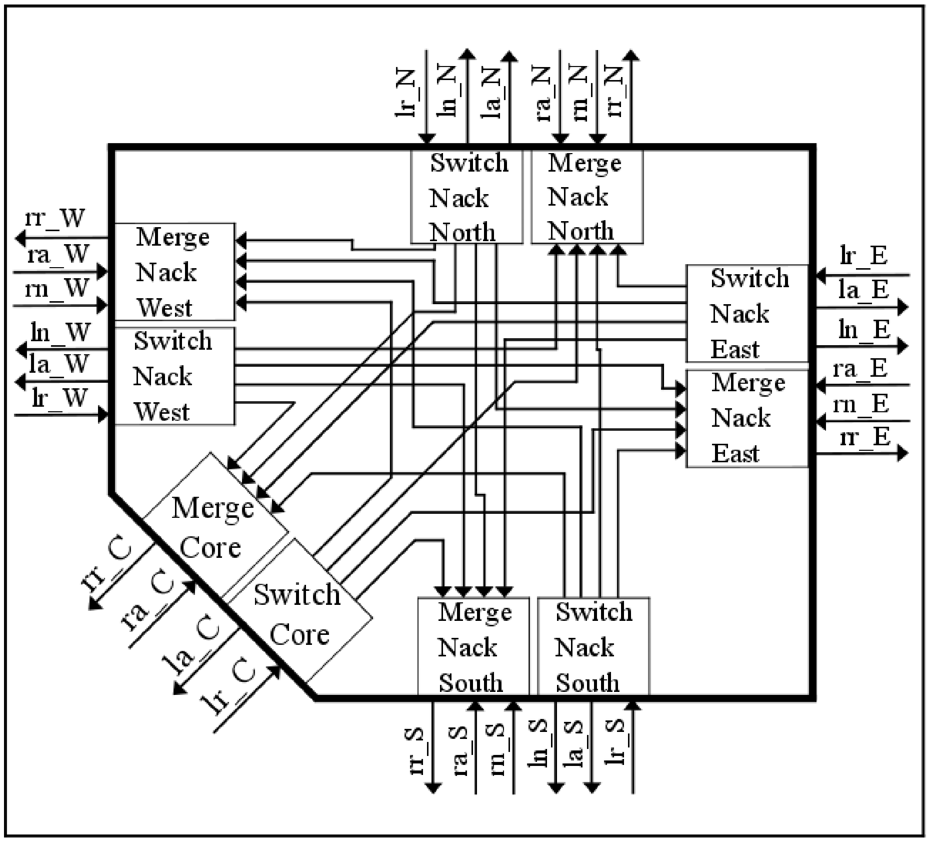


Figure 4.3. Router with virtual channels

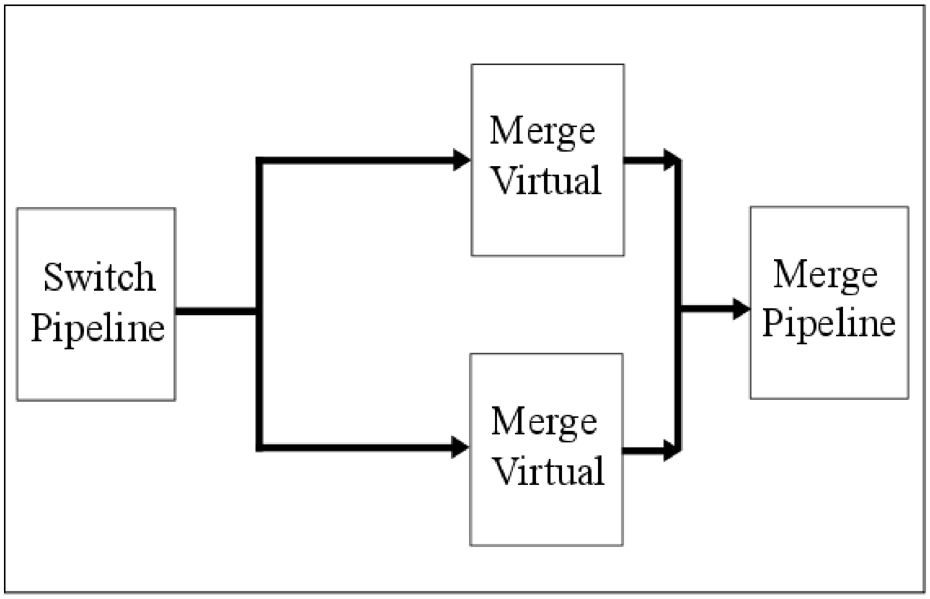


Figure 4.4. Block design of the router port

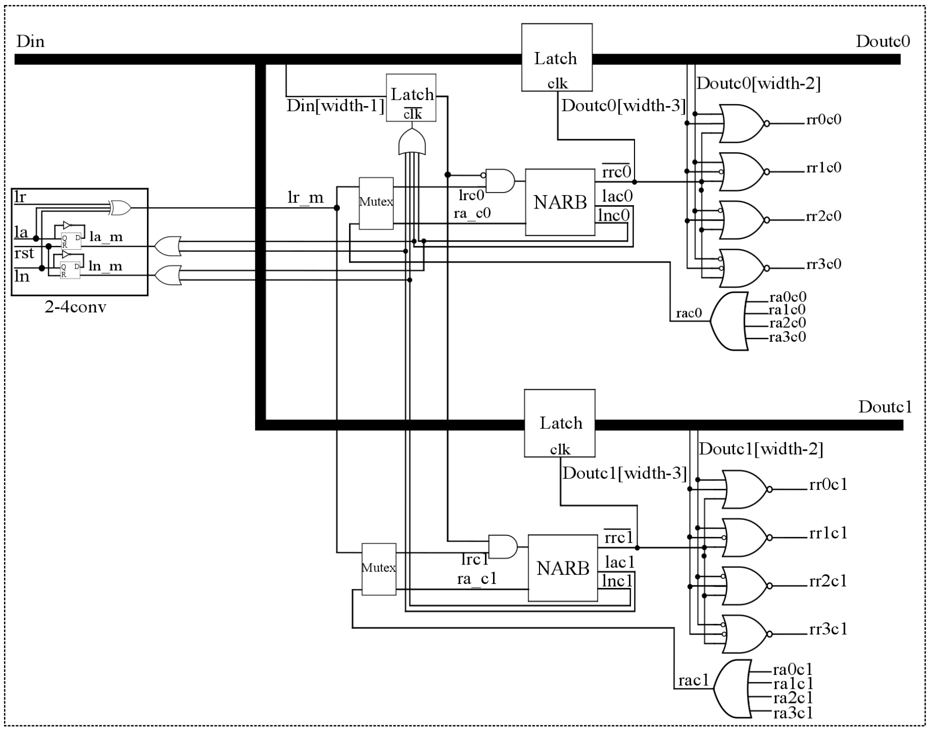


Figure 4.5. Switch module with virtual channels

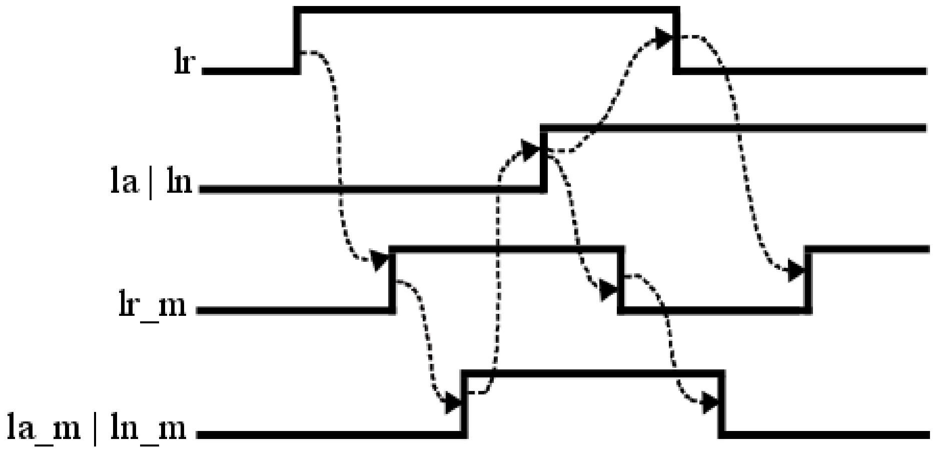


Figure 4.6. 2- to 4-phase converter signal transition order

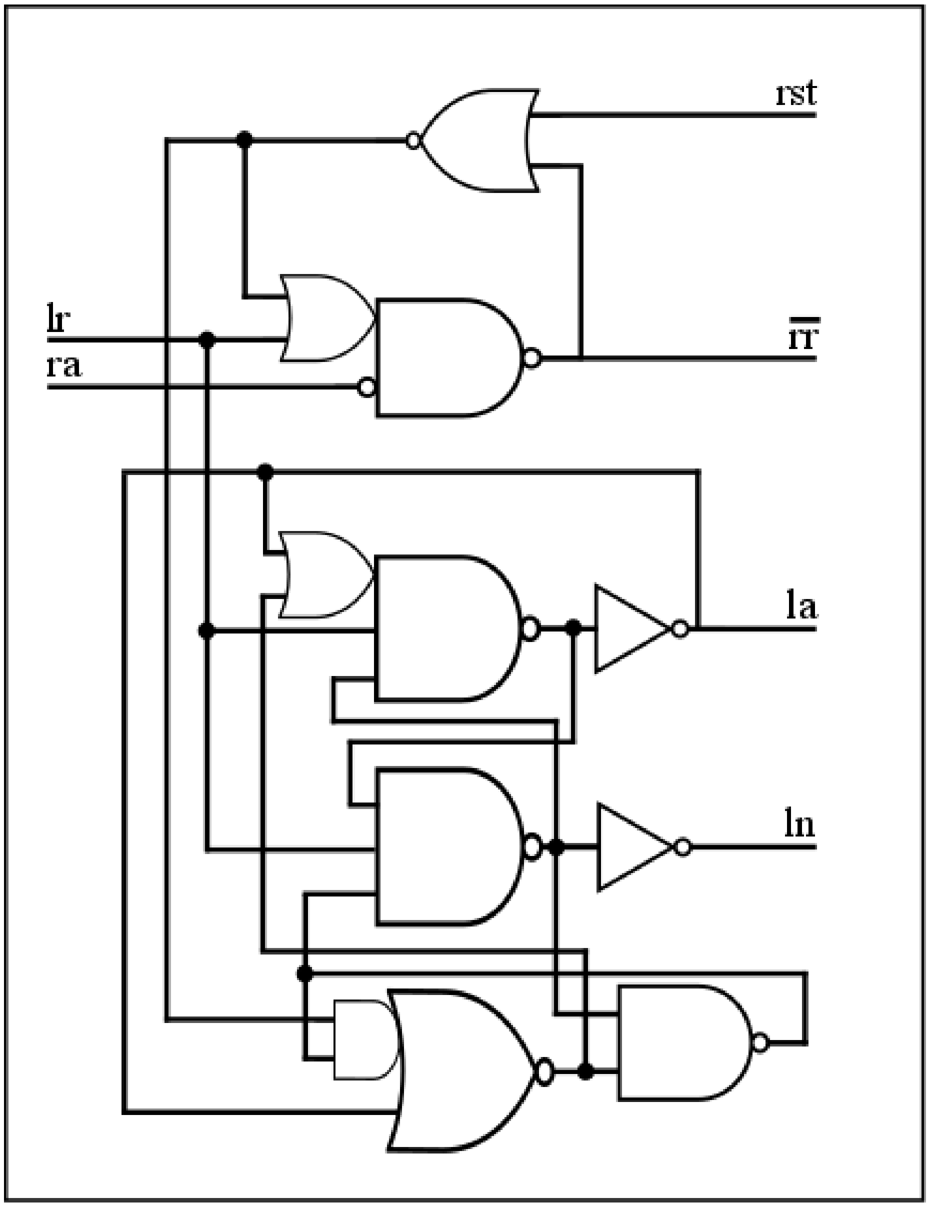


Figure 4.7. Non-blocking arbiter

```
agent SPEC =   rst.S0;
agent S0   =   lr.S1;
agent S1   =                         'la.S2   +  'rr.S3;
agent S2   =                                     'rr.S4;
agent S3   =                         'la.S4;
agent S4   =   lr.S5   +   ra.S6;
agent S5   =               ra.S7  +  'la.S10;
agent S6   =   lr.S7   +                         'rr.S8;
agent S7   =                         'la.S11  +  'rr.S9;
agent S8   =   lr.S9;
agent S9   =                         'la.S12;
agent S10  =   lr.S15  +   ra.S11;
agent S11  =                                     'rr.S12;
agent S12  =   lr.S13  +   ra.S0;
agent S13  =               ra.S1  +  'la.S14;
agent S14  =               ra.S2;
agent S15  =                         'ln.S16;
agent S16  =   lr.S17  +   ra.S18;
agent S17  =               ra.S19 +  'ln.S10;
agent S18  =   lr.S19  +                         'rr.S20;
agent S19  =                         'ln.S11  +  'rr.S21 ;
agent S20  =   lr.S21;
agent S21  =                         'ln.S12;
```

Figure 4.8. Non-blocking arbiter ccs specification

la+ $\mapsto$ y+ $\prec$ la-;
rr+ $\mapsto$ y- $\prec$ lr 2;

Figure 4.9. Relative timing constraints for non-blocking arbiter

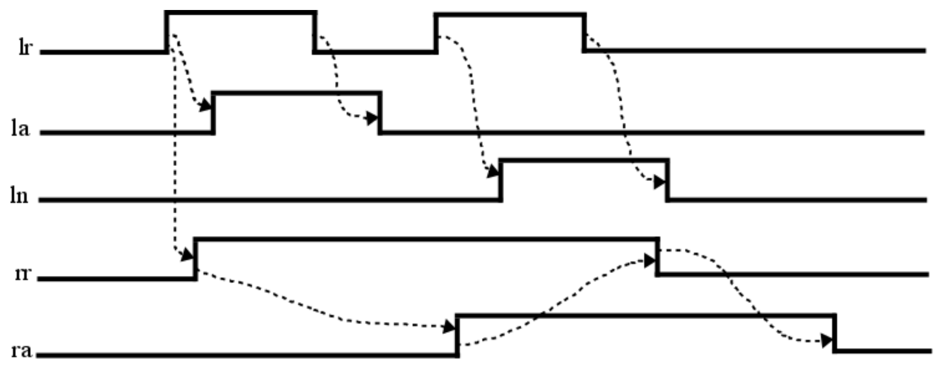


Figure 4.10. Non-blocking arbiter signal transition order

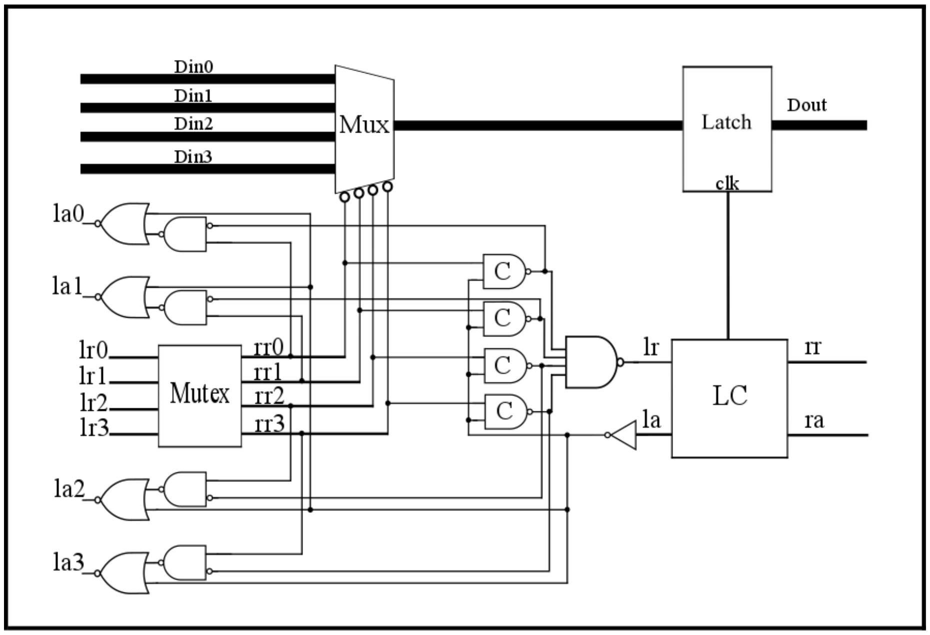


Figure 4.11. Merge virtual module

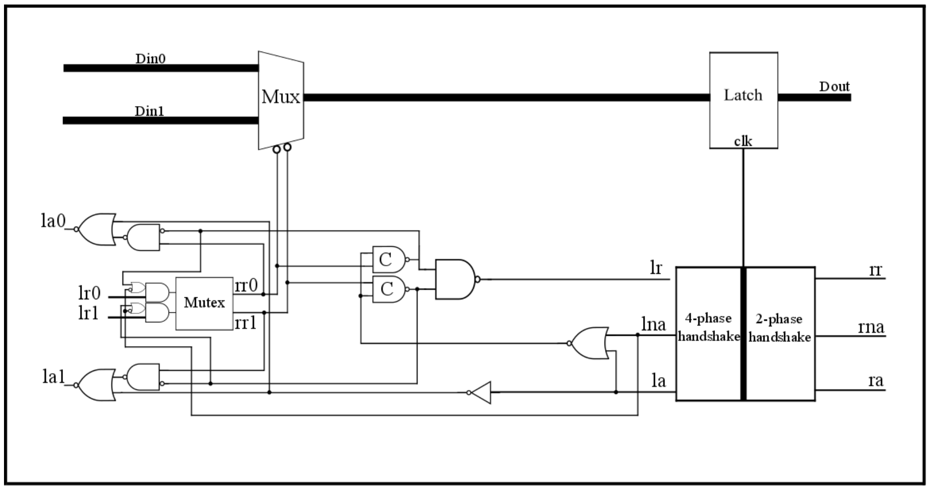


Figure 4.12. Merge pipeline module

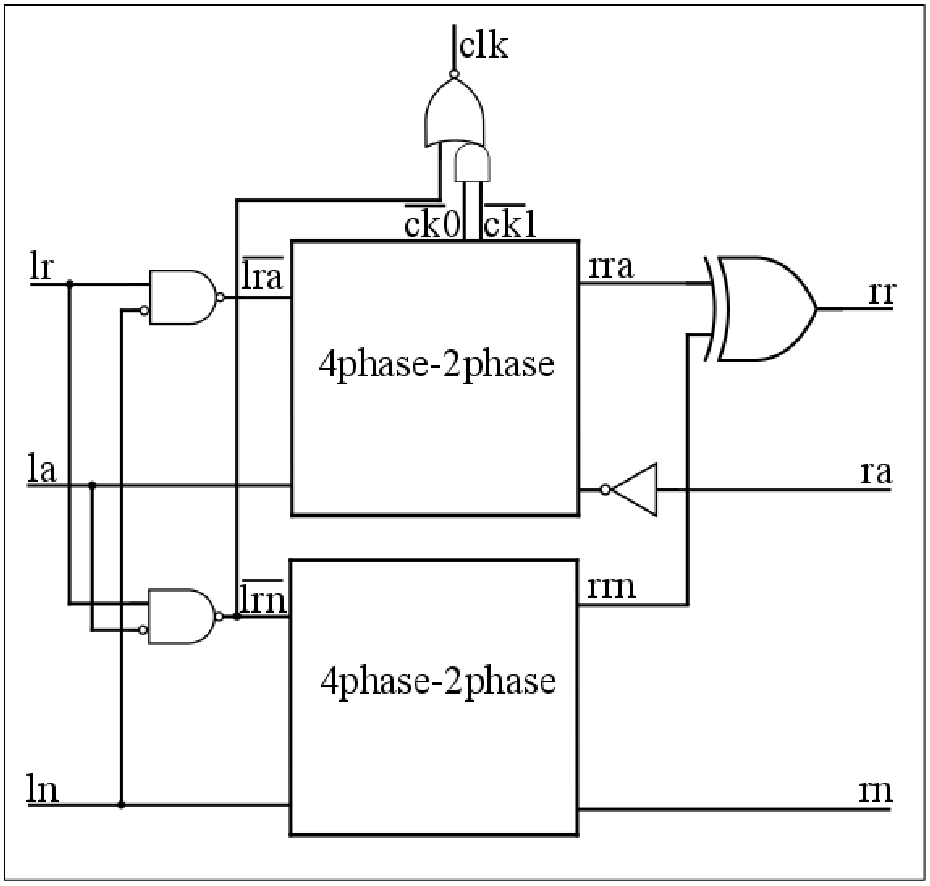


Figure 4.13. 4- to 2-phase converter

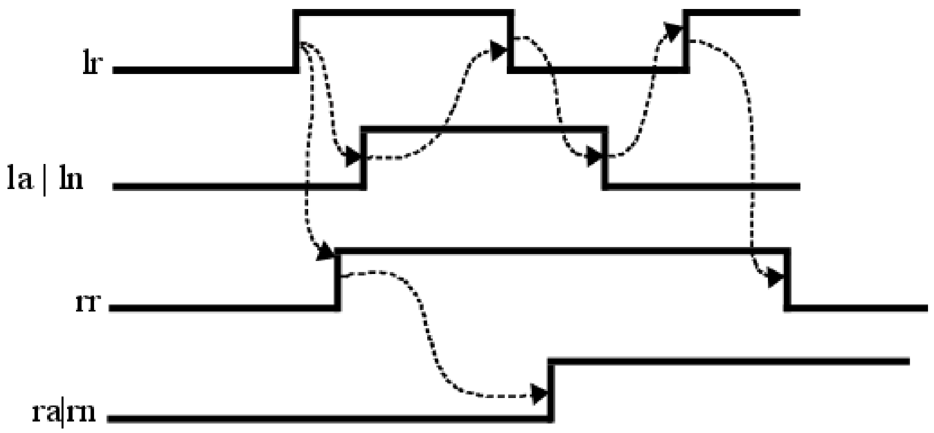


Figure 4.14. 4- to 2-phase converter signal transition order

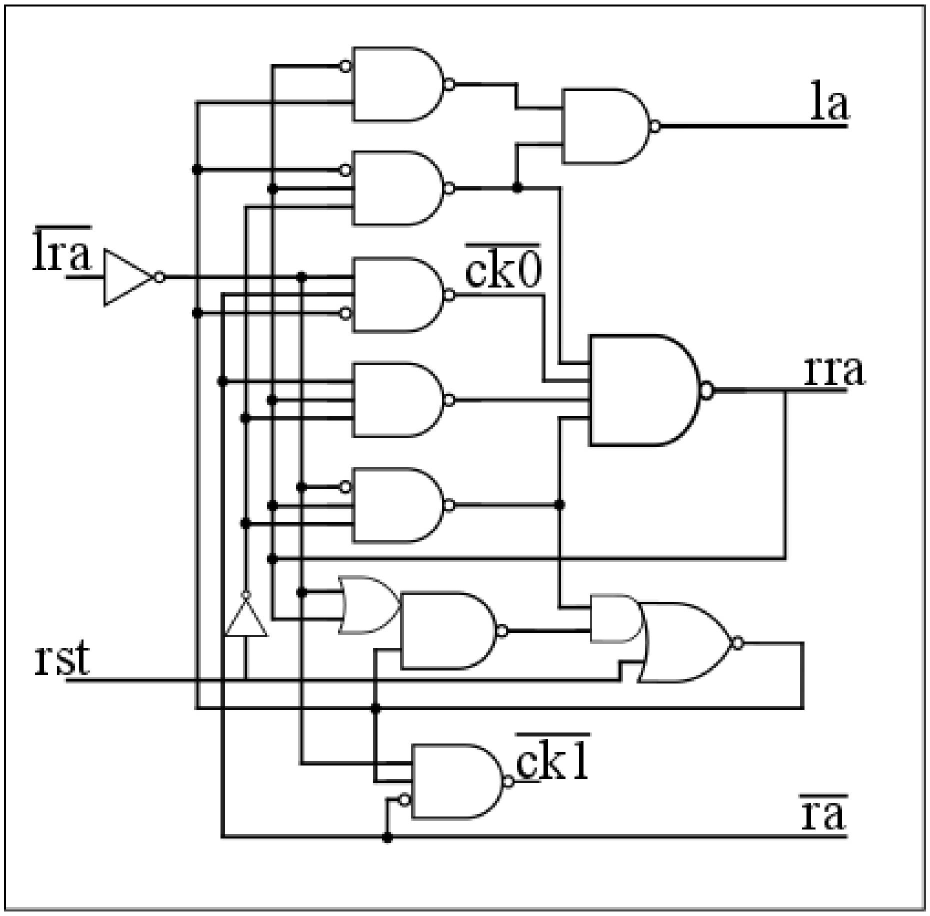


Figure 4.15. 4- to 2-phase converter circuit

$$
\begin{gathered}
\text{lr+} \mapsto \text{lrn+} \prec \text{la+};\\
\text{lr 2} \mapsto \text{lra 2} \prec \text{ln+};\\
\text{rra+} \mapsto \text{lrn-} \prec \text{rn+};\\
\text{rrn+} \mapsto \text{lra 3} \prec \text{ra+};\\
\text{la+} \mapsto \text{lrn-} \prec \text{lr 2};\\
\text{ln+} \mapsto \text{lra 3} \prec \text{lr 4};
\end{gathered}
$$

Figure 4.16. Relative timing constraints for 4- to 2-phase converter

```
agent   SPEC    =   rst.SPEC1 ;
agent   SPEC1   =   lr.SPEC2 ;
agent   SPEC2   =                                          'rr.SPEC3 ;
agent   SPEC3   =                              'la.SPEC4 ;
agent   SPEC4   =   lr.SPEC5   +   ra.SPEC9 ;
agent   SPEC9   =   lr.SPEC8 ;
agent   SPEC8   =                              'la.SPEC1 ;
agent   SPEC5   =                  ra.SPEC8  + 'la.SPEC7 ;
agent   SPEC7   =   lr.SPEC6   +   ra.SPEC1 ;
```

Figure 4.17. 4- to 2-phase converter circuit ccs specification

$$\begin{gathered} rr+ \mapsto rr1+ \prec ra\_+; \\ y0+ \mapsto y00+ \prec rry0-; \\ rr+ \mapsto rr1+ \prec rry0-; \\ rr+ \mapsto rry0+ \prec ck0-; \end{gathered}$$

Figure 4.18. Relative timing constraints for 4- to 2-phase converter circuit

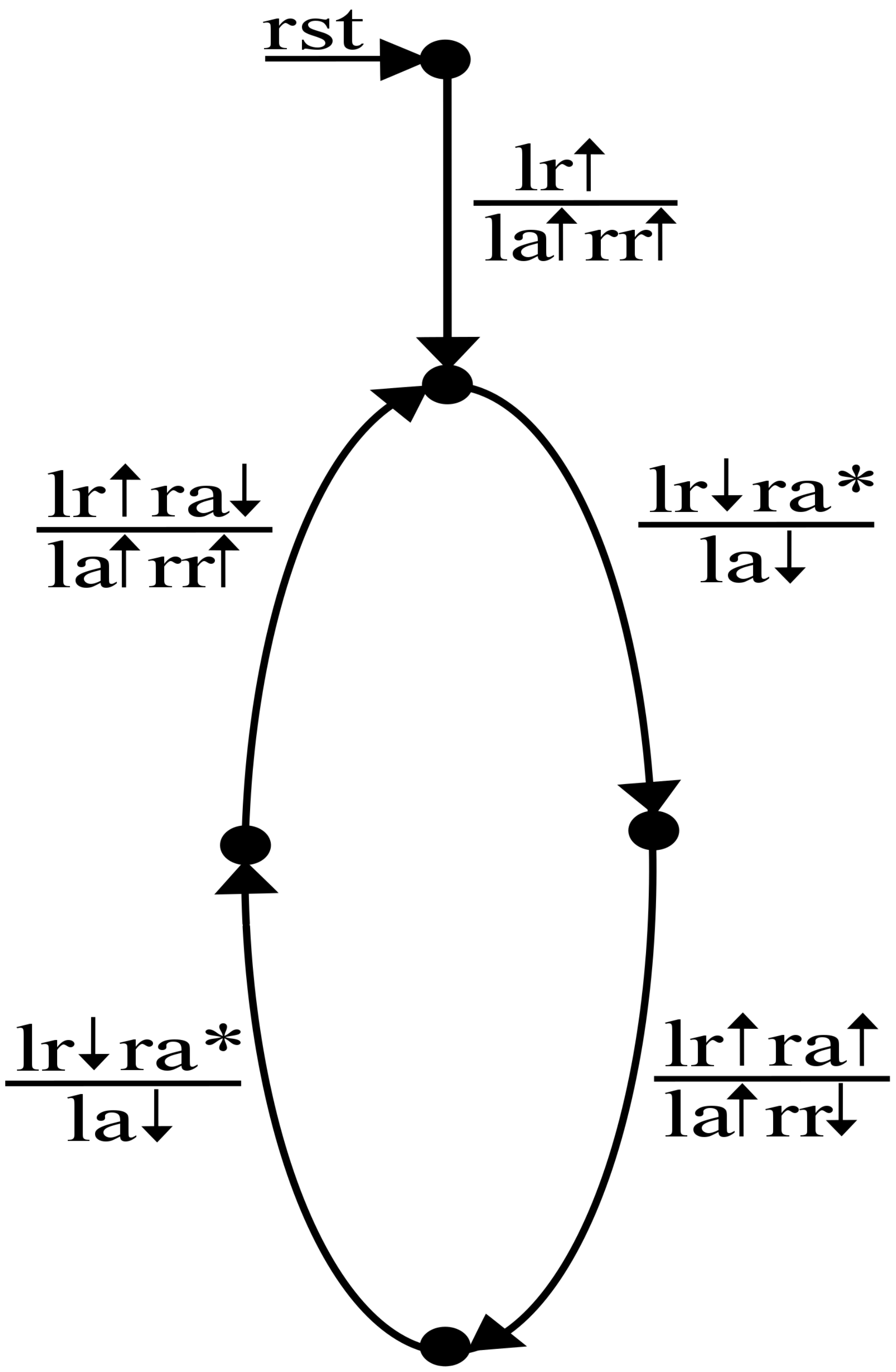


Figure 4.19. 4- to 2-phase converter circuit MEAT specification

CHAPTER 5

ACCURATE SIMULATION APPROACH

Current network simulators target cycle accurate performance evaluation while providing customization for network topology, and scalability for large number of nodes [40, 50, 51]. However, they often lack fidelity for RTL accurate temporal reporting and network power evaluation. Routers and channels are replaced by model abstractions, thereby sacrificing accuracy for run-time improvement. While analyzing NoC architectures that employ complex transceiver and router circuits, and unorthodox line links, accuracy becomes the prime consideration for the simulator. In this work, a circuit-accurate NoC simulator that uses a Verilog simulation environment for network modeling, and a C++ wrapper for traffic generation and performance analysis, was built by Mac J. Wibbels, Masters student under Prof. Kenneth Stevens. The bridge between two domains is built using a SystemVerilog domain programming interface (DPI). The developed modular simulation engine consists of three main components: a traffic generator, network interface, and traffic analyzer. Each component is configurable and extensible to different routing algorithms, network topologies and configurations, and report data. The network simulation is performed by a Verilog simulator on circuits that have been fully designed and validated. This allows highly accurate latency and power results, and for the designer to observe specific behavior characteristics (such as arbitration) that are challenging or of specific interest to design quality.

5.1 Network fabric configuration

The network fabric becomes the device under test (DUT) for the simulator. The simulator generates stimulus to the Verilog DUT and evaluates the results of the simulation. This specific design fabric consists of fully implemented routers for each differing design configuration. These are arrayed and interconnected with emulated links characterized by SPICE simulations [13]. Characterized values for wire delays and energies of RC and transmission line (TL) links in the 65nm node are obtained from [13]. These models can be optimized for energy or latency, or for different metal lines. This provides additional flexibility to our simulator as we can easily switch between different wire models. The links are emulated in the network fabric as Verilog delay statements on each wire of the link based on the values from [13].

5.2 Simulation setup

Network hardware simulation is performed using the standard VLSI tool flow. Circuits are specified in Verilog, synthesized using Synopsys Design Compiler. ModelSim is used for functional verification, using back-annotated parasitic SDF values. Synopsys Primetime PX is used to perform static timing analysis (STA) as well as generate power numbers using the value change dump (VCD) and standard parasitic exchange format (SPEF) files from the simulations. Interconnect characteristics are obtained from SPICE simulations as described above. The traffic generation model is implemented using a high level language (C++). The traffic generator is designed to accommodate a variety of generation patterns such as nearest neighbor and uniform random [9]. Workloads, including random permutations, are implemented. A routing table is used by the traffic generator to convert source and destination indexes to router specific packet address bits. The routing table can be customized to use module interconnect delays or hop count as metrics to evaluate the optimal packet path. Additional routing table implementations can be substituted to evaluate other routing protocols. Messages are sent and tracked through the network using a unique identifier payload (data bits). The identifier payload is appended with additional data bits to achieve a user-specified activity factor. The message queue operates on an extensible message class. The queue stores and timestamps flits and messages that enter and leave the network. The traffic generator is parametrized for bandwidth injection rate.

The simulation consists of two phases, a warm-up phase that is tasked with placing the network in a load state that is within the operating range for provided traffic, and an evaluation phase. These each consist of a time window sufficient to inject w warm up messages into the network, and e evaluation messages into the network. A cool-down phase, which drains queued messages without adding additional traffic, is optional. The traffic generator will iterate over different simulation loads when given a starting and ending injection rate. The network interface module houses all the basic functions to interact with the DUT by introducing and removing flits from the network. It is implemented in SystemVerilog. Traffic is passed to the Verilog environment through a socket interface using DPI function calls. These calls communicate with the core switch and merge interfaces of each router. The simulations used are configured as follows. Each simulation has a fixed injection period, at which point a new message is injected using uniform random message distribution. Messages are configured to contain 10 flits. Injection period start and end values IP_s and IP_e are provided, along with the number of simulations N. Warm up and evaluation times T_w and T_e are also provided. The injection period for the simulation in terms of picoseconds per byte $IP_{ps/b}$ is calculated as $IP_{ps/b} = 1/(IP_s + i((IP_e - IP_s)/N))$. This value is used to calculate the number of messages in the warm up as $M_w = T_w/(IP_{ps/b} \times M_{sz})$ where M_{sz} is the number of bytes per message. The number of messages in the evaluation is calculated as $M_e = T_e/(IP_{ps/b} \times M_{sz})$.

CHAPTER 6

EVALUATION

Four different network topologies are built: mesh with RC wires (MeshRC), torus with RC (TorusRC) wires on wrap lines, torus with transmission Lines (TorusTL) on wrap lines, and folded torus with RC wires (FTorus). Spacing between nodes and network wire length are based on fixed die size of 8×8 and 64 routers. Link delay and energy are calculated by predetermined values based on length for both diffusive and transmission lines as shown in Tables 6.1, 6.2, and 6.3 [13]. The short wire between two routers on a 2-dimensional network is considered to be a 1 mm long diffusive RC wire. Long interconnects or wrap lines in toroidal networks are considered to be 7 mm long. FTorus uses diffusive RC wires 1.75 mm in length between any two adjacent nodes or routers.

6.1 Latency evaluation

The traffic generator and network interface modules feed data to the traffic analyzer module based on Verilog simulation results of the network fabric. This module reads the unique message and flit identification codes to verify reception of each message and to calculate flit and message latency. The simulator uses real workload data, which provides a rich set of packet evaluation metrics. The simple interface and modularity of the traffic analyzer allows extended classes to evaluate additional metrics. This allows reporting of critical data such as link utilization, energy, data activity factors, and hop-to-hop latency for each flit and message.

Complete activity of the routers and interconnect are written to a VCD file during simulation. Router power is calculated using *Primetime PX*. Interconnect power is calculated by a custom C++ program that parses the generated VCD file and computes interconnect transition count. The interconnect transition count is multiplied by SPICE extracted energy measurement of the interconnects shown in Tables 6.1, 6.2, and 6.3, to achieve precise interconnect power estimation. The traffic analyzer reports the injection rate, warm-up message count, messages injected during the evaluation phase, total message hop count, and average hop count per message. The minimum, average, and maximum latencies for flits and messages from source to destination, as well as between routers in the network, are also reported.

6.2 Results – direct translation

Figure 6.1 plots average message latencies versus message injection rate by changing the message injection rate against the number of messages ($warmup + evaluation$) on a linear scale, described in Section 5.2, for all the developed network topologies. From initial results in Figure 6.1, an observation can be made that torus with transmission lines saturates later when compared to other Torus topologies with higher link delays. The figure also shows that mesh networks saturate much later when compared to all the other toroidal networks. Results in Figure 6.1 are obtained by a simple translation from Junbok's designs to support higher radix networks [43].

Direct translation of 3-port routers to 5-port routers results in better cycle time for routers without virtual channels when compared to routers with virtual channels. This is due to no change in pipelining for the routers without virtual channels and the routers with virtual channels. This resulted in a large cycle time penalty for virtual channel routers due to the extra complexity added due to the virtual channel logic, as shown in Figure 6.2. Simple routing logic of the routers makes maximum cycle time of the routers as the main bottleneck for network latency.

Also the message latency is calculated from the point the message is put into the queue to the point it is received out of the network. Message latency for the network alone saturates at a point where the network works at its maximum message accepting limit and stays there even for higher injection rates. This latency is the message transmission latency and is shown in Figure 6.3 for 8×8 MeshRC.

Further optimization of cycle times in a router with virtual channels can give better latency benefits for toroidal networks, when compared to that of mesh.

6.3 Results – repipelining

A smaller network size of 4×4 and 16 routers were developed for MeshRC and TorusTL topologies with 2 mm short links and 14 mm long wrap lines. The routers used in TorusTL were repipelined to achieve comparable cycle times with that of routers used in MeshRC, as shown in Figure 6.4. Figure 6.5 shows average message latencies per hop versus message injection rate, varying the injection rate and total messages on the same scale as used in Section 6.2. These results show that with comparable cycle times, TorusTL saturates much later when compared to MeshRC and results in higher throughput when compared to MeshRC.

Table 6.1. Transmission line (TL) evaluation for 7 mm link

Transmission Line (opt)	Latency (ps)	Energy (pJ/bit)	Pitch (μm)	Eye Height (V)	Eye Width (ps)
TL1 (energy)	338.3	0.54	2.8	0.88	339.6
TL2 (latency)	292.9	0.60	7.6	0.85	337.9

Table 6.2. Repeated RC evaluation for 7 mm link

Diffusive Wire (metal)	Latency (ps)	Energy (pJ/bit)	Repeater Size (μm)
RC1 (M1)	963.8	1.08	3.6
RC2 (M5)	775.5	0.78	3.6

Table 6.3. Repeated RC evaluation for 1 mm link

Diffusive Wire (metal)	Latency (ps)	Energy (pJ/bit)	Repeater Size (μm)
RC1 (M1)	137.7	0.154	3.6
RC2 (M5)	110.8	0.111	3.6

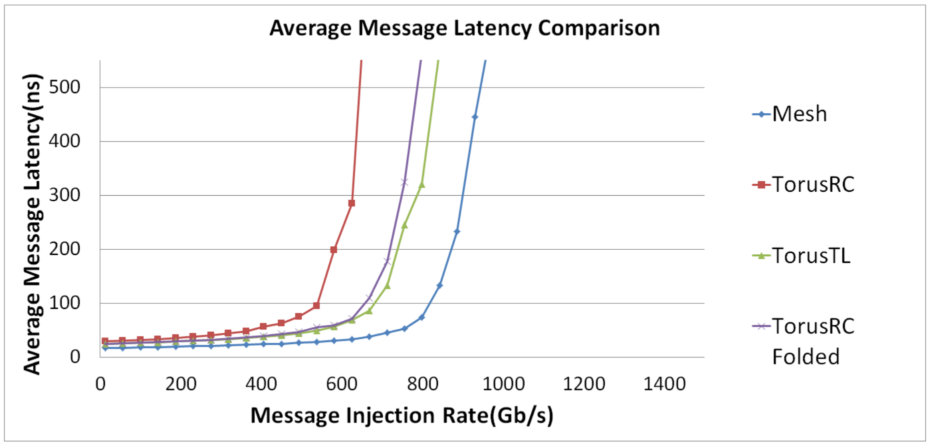


Figure 6.1. Average message latency comparison without repipelining for 8×8 networks

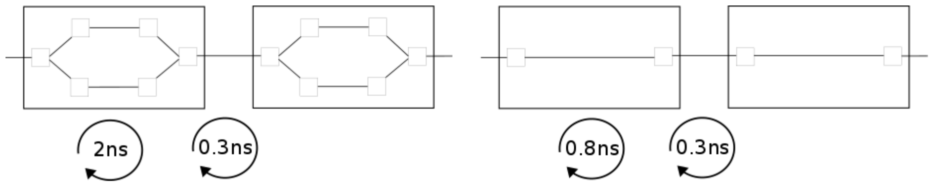


Figure 6.2. Maximum cycle time of routers with and without virtual channels

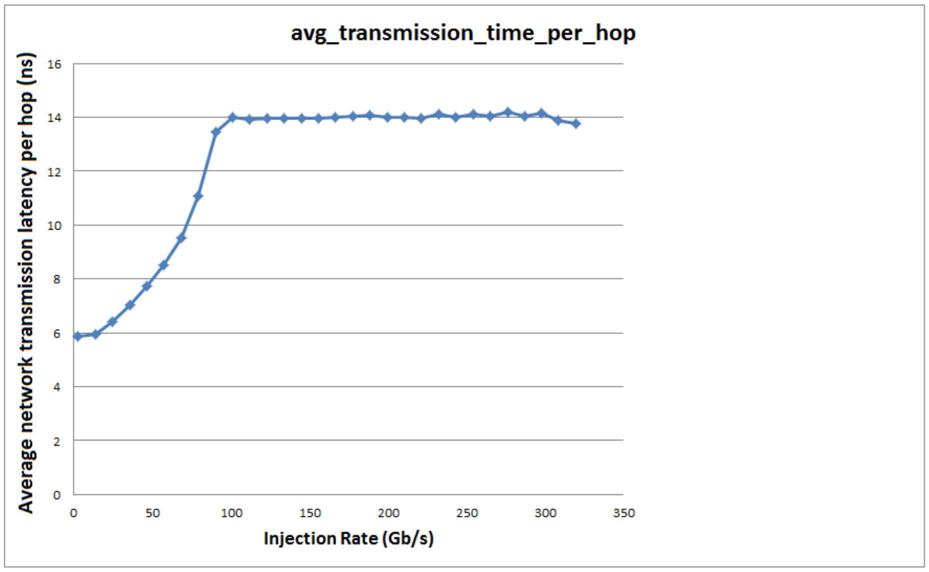


Figure 6.3. Message transmission latency for 8×8 MeshRC

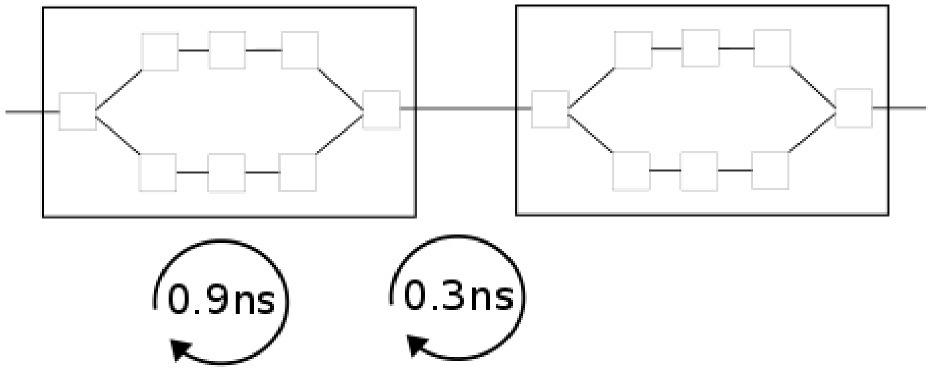


Figure 6.4. Maximum cycle time of routers with virtual channels post repipelining

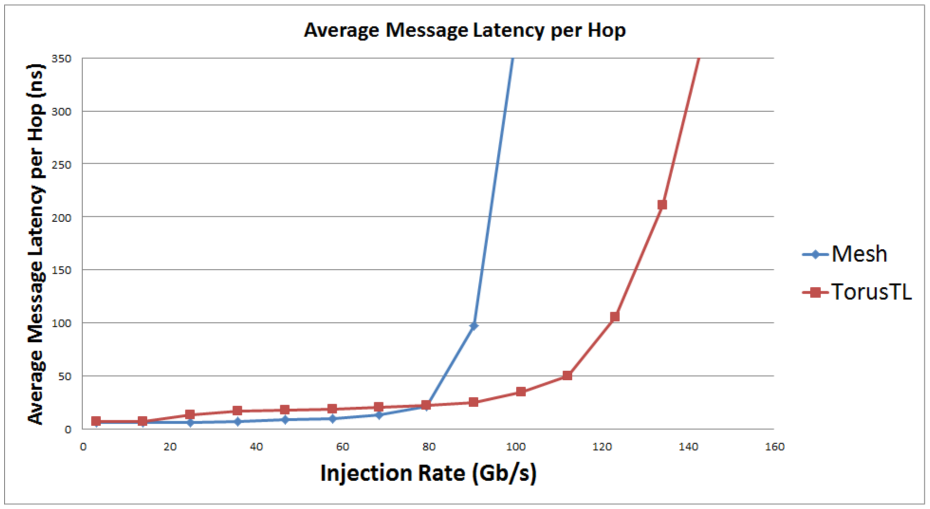


Figure 6.5. Average message latency comparison with repipelining for 4×4 networks

CHAPTER 7

FUTURE WORK

Asynchronous routers can be integrated with source asynchronous signaling (SAS) [52]. Synchronous versions of the routers can be developed using elastic half buffers [53]. This work can be extended to more complex topologies such as a hypercube or butterfly etc. [9]. The simulator can be integrated with different traffic models such as B-Model and SynFull etc. [54, 55]. These asynchronous router designs should also be evaluated against a synchronous version for power and performance.

CHAPTER 8

CONCLUSION

This work researches NoC's efficiency through simplicity. It starts with a simple T-router-based NoC designed by Junbok You [43]. It incorporates source routing and single flit packets. It directly translates the T-routers to higher radix (5-port routers). This research employs carefully designed high speed relative timed routers for an asynchronous NoC.

This work implements a fixed die size 8×8 mesh topology using the 5-port router. Switch and merge modules are directly extended from 3-ported to support 5-ported routers. It uses two MSBs of the packet as part of direction decoding bits. It employs bit rotation logic to traverse the route from source to destination with encoded return address. This work uses a uniform random XY routing algorithm to route messages from source to destination.

This work extends a 2D mesh with RC wires (MeshRC) to 3D topologies such as torus with RC (TorusRC) wires on wrap lines, torus with transmission lines (TorusTL) on wrap lines, and folded torus with RC wires (FTorus). The link delays and energies are calculated by predetermined values based on length for both diffusive and transmission lines. Wire delay values are directly integrated into network simulator. The simulator also implements XY routing onto these 3D topologies. This work examines deadlock scenarios and develops routers with virtual channels and negative acknowledgement signals to break the cycles, and hence getting rid of deadlocks.

This work studies the trade-off between reduction in hop count versus slowdown due to extra logic complexity for toroidal networks. It evaluates the message latencies for all the four developed networks with fully implemented routers. This thesis focuses on network simplicity, and how adding additional complexity affects the performance and area of a design.

This work examines how repipelining of the routers with virtual channels helps toroidal networks achieve better latencies and throughput when compared to mesh networks. This repipelining adds significant area and power to the design, but significantly improves performance. This work also talks about an accurate NoC modeling through a simulator that takes into account the effects of on-die implementation, on network performance, through the use of synthesized Verilog modules and interconnect models.

REFERENCES

[1] S. Furber, "Future trends in soc interconnect," in *VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT). 2005 IEEE VLSI-TSA International Symposium on*, April 2005, pp. 295–298.

[2] C. Grecu, P. Pande, A. Ivanov, and R. Saleh, "A scalable communication-centric soc interconnect architecture," in *Quality Electronic Design, 2004. Proceedings. 5th International Symposium on*, 2004, pp. 343–348.

[3] R. Ho, K. Mai, and M. Horowitz, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, Apr 2001.

[4] S. Sharma, C. Mukherjee, and A. Gambhir, "A comparison of network-on-chip and buses."

[5] A. Lastovetsky, *Distributed Memory Multiprocessors.* John Wiley and Sons, Inc., 2004, pp. 95–139. [Online]. Available: http://dx.doi.org/10.1002/0471654167.ch4

[6] D. H. Lawrie, "Memory-Processor Connection Networks," Ph.D. dissertation, University of Illinois at Champaign-Urbana, February 1973.

[7] C. H. Sequin, "Doubly Twisted Torus Networks for VLSI Processor Arrays," in *Proceedings of 8th Annual Symposium on Computer Architecture*, IEEE. New York, NY: IEEE Press, 1981, pp. 471–480.

[8] K. W. Doty, "New Designs for Dense Processor Interconnection Networks," *IEEE Transactions on Computers*, vol. C-33, no. 5, pp. 447–450, May 1984.

[9] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks.* Elsevier, 2004.

[10] W. Zhang, L. Hou, J. Wang, S. Geng, and W. Wu, "Comparison research between xy and odd-even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip," in *2009 WRI Global Congress on Intelligent Systems*, vol. 3, May 2009, pp. 329–333.

[11] M. Li, Q.-A. Zeng, and W.-B. Jone, "Dyxy: A proximity congestion-aware deadlock-free dynamic routing method for network on chip," in *Proceedings of the 43rd Annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: ACM, 2006, pp. 849–852. [Online]. Available: http://doi.acm.org/10.1145/1146909.1147125

[12] M. Dehyadgari, M. Nickray, A. Afzali-kusha, and Z. Navabi, "Evaluation of pseudo adaptive xy routing using an object oriented model for noc," in *2005 International Conference on Microelectronics*, Dec 2005, pp. 5 pp.–.

[13] S. Das, "Techniques for fast, energy efficient on-die global communication," Ph.D. dissertation, The University of Utah, 2016.

[14] D. Xiang, Y. Pan, Q. Wang, and Z. Chen, "Deadlock-free fully adaptive routing in 2-dimensional tori based on new virtual network partitioning scheme," in *Distributed Computing Systems, 2008. ICDCS '08. The 28th International Conference on*, June 2008, pp. 454–461.

[15] G. Chen, M. A. Anders, H. Kaul, S. K. Satpathy, S. K. Mathew, S. K. Hsu, A. Agarwal, R. K. Krishnamurthy, V. De, and S. Borkar, "A 340 mv-to-0.9 v 20.2 tb/s source-synchronous hybrid packet/circuit-switched 16 x00d7; 16 network-on-chip in 22 nm tri-gate cmos," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 59–67, Jan 2015.

[16] A. Joshi, B. Kim, and V. Stojanovic, "Designing energy-efficient low-diameter on-chip networks with equalized interconnects," in *17th IEEE Symposium on High Performance Interconnects, 2009. HOTI 2009.*, Aug 2009, pp. 3–12.

[17] V. S. Vij, "Algorithms and methodology to design asynchronous circuits using synchronous cad tools and flows," Ph.D. dissertation, The University of Utah, 2013.

[18] K. S. Stevens, P. Golani, and P. A. Beerel, "Energy and performance models for synchronous and asynchronous communication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, pp. 369–382, March 2011.

[19] K. Stevens, R. Ginosar, and S. Rotem, "Relative timing [asynchronous design]," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 1, pp. 129–140, Feb 2003.

[20] J. Cortadella, M. Kishinevsky, S. M. Burns, and K. Stevens, "Synthesis of asynchronous control circuits with automatically generated relative timing assumptions," in *Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, Nov 1999, pp. 324–331.

[21] K. S. Stevens, S. V. Robison, and A. L. Davis, "The Post Office – Communication Support for Distributed Ensemble Architectures," in *Proceedings of 6th International Conference on Distributed Computing Systems*, May 1986, pp. 160 – 166, best paper award.

[22] K. S. Stevens, A. L. Davis, and W. S. Coates, "The Post Office Experience: Designing a Large Asynchronous Chip," in *Proceedings of the 26th Hawaii International Conference on System Sciences*, January 1993, pp. 409–418.

[23] T. Bjerregaard and J. Sparso, "Implementation of guaranteed services in the mango clockless network-on-chip," *IEE Proceedings on Computers and Digital Techniques*, vol. 153, no. 4, pp. 217–229, July 2006.

[24] A. Sheibanyrad and A. Greiner, "Two efficient synchronous & asynchronous converters well-suited for networks-on-chip in GALS architectures," *Integration, the VLSI Journal*, vol. 41, no. 1, pp. 17–26, 2008.

[25] G. Campobello, M. Castano, C. Ciofi, and D. Mangano, "GALS Networks on Chip: A New Solution for Asynchronous Delay-Insensitive Links," in *Design, Automation and Test in Europe*, Mar 2006, pp. 160–165.

[26] C. Koch-Hofer, M. Renaudin, Y. Thonnart, and P. Vivet, "ASC, a SystemC Extension for Modeling Asynchronous Systems, and Its Application to an Asynchronous NoC," in *International Symposium on Networks-on-Chip*, 2007, pp. 295–306.

[27] L. Xin and C.-S. Choy, "An asynchronous router with multicast support in noc," in *Proceedings of the Fifth IASTED International Conference on Circuits, Signals and Systems*, ser. CSS '07. Anaheim, CA, USA: ACTA Press, 2007, pp. 59–63. [Online]. Available: http://dl.acm.org/citation.cfm?id=1659760.1659774

[28] X.-T. Tran, Y. Thonnart, J. Durupt, V. Beroulle, and C. Robach, "A design-for-test implementation of an asynchronous network-on-chip architecture and its associated test pattern generation and application," in *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 149–158. [Online]. Available: http://dl.acm.org/citation.cfm?id=1397757.1397995

[29] G. Gopalakrishnan, P. Kudva, and E. Brunvand, "Peephole optimization of asynchronous macromodule networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 30–37, March 1999.

[30] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Efficient link capacity and QoS design for network-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2006, pp. 9–14.

[31] D. Rostislav, V. Vishnyakov, E. Friedman, and R. Ginosar, "An asynchronous router for multiple service levels networks on chip," in *Proceedings. 11th IEEE International Symposium on Asynchronous Circuits and Systems, 2005. ASYNC 2005.*, March 2005, pp. 44–53.

[32] T. Krishna, A. Kumar, P. Chiang, M. Erez, and L.-S. Peh, "NoC with Near-Ideal Express Virtual Channels Using Global-Line Communication," in *Symposium on High Performance Interconnects.* IEEE, 2008, pp. 11–20.

[33] M. N. Horak, S. M. Nowick, M. Carlberg, and U. Vishkin, "A low-overhead asynchronous interconnection network for gals chip multiprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 494–507, April 2011.

[34] G. Gill, S. S. Attarde, G. Lacourba, and S. M. Nowick, "A low-latency adaptive asynchronous interconnection network using bi-modal router nodes," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, May 2011, pp. 193–200.

[35] G. Faldamis, W. Jiang, G. Gill, and S. M. Nowick, "A low-latency asynchronous interconnection network with early arbitration resolution," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, Jan 2014, pp. 329–336.

[36] A. Ghiribaldi, D. Bertozzi, and S. M. Nowick, "A transition-signaling bundled data noc switch architecture for cost-effective gals multicore systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 332–337.

[37] A. Elshennawy and S. P. Khatri, "An asynchronous network-on-chip router with low standby power," in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, Oct 2014, pp. 394–399.

[38] W. Jiang, K. Bhardwaj, G. Lacourba, and S. M. Nowick, "A lightweight early arbitration method for low-latency asynchronous 2d-mesh noc's," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, June 2015, pp. 1–6.

[39] T. Krishna, C. H. O. Chen, W.-C. Kwon, and L.-S. Peh, “Smart: Single-cycle multihop traversals over a shared network on chip,” *IEEE Micro*, vol. 34, no. 3, pp. 43–56, May 2014.

[40] D. Gebhardt, J. You, and K. Stevens, “Design of an energy-efficient asynchronous noc and its optimization tools for heterogeneous socs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1387–1399, Sept 2011.

[41] G. Chen, M. Anders, H. Kaul, S. Satpathy, S. Mathew, S. Hsu, A. Agarwal, R. Krishnamurthy, V. De, and S. Borkar, “A 340 mv-to-0.9 v 20.2 tb/s source-synchronous hybrid packet/circuit-switched 16 $\times$ 16 network-on-chip in 22 nm tri-gate cmos,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 59–67, Jan 2015.

[42] E. Kasapaki and J. Sparso, “Argo: A time-elastic time-division-multiplexed noc using asynchronous routers,” in *2014 20th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2014, pp. 45–52.

[43] J. You, “Design and optimization of asynchronous network-on-chip,” Ph.D. dissertation, The University of Utah, 2011.

[44] C. Mead and L. Conway, *Introduction to VLSI Systems*, ser. Computer Science. Reading, MA: Addison-Wesley, 1980.

[45] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design – A Systems Perspective.* Kluwer Academic Publishers, 2001.

[46] V. S. Vij and K. S. Stevens, “Automatic addition of reset in asynchronous sequential control circuits,” in *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2013, pp. 374–379.

[47] V. S. Vij, R. P. Gudla, and K. S. Stevens, “Interfacing synchronous and asynchronous domains for open core protocol,” in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, Jan 2014, pp. 282–287.

[48] G. Birtwistle and K. S. Stevens, “The Family of 4-phase Latch Protocols,” in *14th International Symposium on Asynchronous Circuits and Systems.* IEEE, April 2008, pp. 71–82.

[49] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, “Virtual channels in networks on chip: Implementation and evaluation on hermes noc,” in *2005 18th Symposium on Integrated Circuits and Systems Design*, Sept 2005, pp. 178–183.

[50] F. Fazzino, M. Palesi, and D. Patti, “Noxim: Network-on-chip simulator,” *URL: http://sourceforge. net/projects/noxim*, 2008.

[51] N. Jiang, D. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. Shaw, J. Kim, and W. Dally, “A detailed and flexible cycle-accurate network-on-chip simulator,” in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2013, pp. 86–96.

[52] S. Das, V. Vij, and K. S. Stevens, “SAS: Source Asynchronous Signaling Protocol for Asynchronous Handshake Communication Free from Wire Delay Overhead,” in *International Symposium on Asynchronous Circuits and Systems.* IEEE, May 2013, pp. 107–114.

[53] J. You, Y. Xu, H. Han, and K. S. Stevens, “Performance Evaluation of Elastic GALS Interfaces and Network Fabric,” *Electronic Notes in Theoretical Computer Science*, vol. 200, no. 1, pp. 17–32, February 2008, elsevier.

[54] M. Badr and N. E. Jerger, “Synfull: Synthetic traffic models capturing cache coherent behaviour,” in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, June 2014, pp. 109–120.

[55] D. Gebhardt, J. You, and K. S. Stevens, “Link pipelining strategies for an application-specific asynchronous noc,” in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, May 2011, pp. 185–192.