October 2012

Control Plane for Embedded DSP

Richard Michael Dennen
Worcester Polytechnic Institute

WORCESTER POLYTECHNIC INSTITUTE

Control Plane for Embedded Digital Signal Processing

An Application of Networks-on-Chips for VLSI

Major Qualifying Project Report

Submitted to the Faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Richard Dennen

on October 22, 2012

Abstract

This project is sponsored by MITRE Corporation to develop a scalable and reusable control plane architecture for VLSI design. The main goal of this project is to develop a communication platform for a wide range of applications to reduce the development and testing time associated with the design of a interconnect system. Thorough research has been conducted in the area of network-on-chip designs that are suitable for these types of applications. The necessary components are built and verified in hardware description language. The deliverable components are packaged as reusable and parameterized SystemVerilog code.

Acknowledgements

Al Conti and Paul Secinaro from MITRE Corporation for advising this project. Your commitment and dedication to this project allowed to me to succeed in accomplishing the goals of this project. I thank you both for everything you have taught me in my time at MITRE. I appreciate that you committed so much time from your busy schedules to guide me through the design of this project.

Prof. Xinming Huang from Worcester Polytechnic Institute for advising this project, taking the time to ensure that my project was progressing well, offering feedback on the design and opinions in important design decisions, as well as the detailed revisions and corrections made to this report!

Prof. Sergey Makarov from Worcester Polytechnic Institute for introducing me to MITRE Corporation and encouraging me to pursue a project at this new project center.

Joseph Chapman and Adam Woodbury from MITRE Corporation for arranging this new partnership with WPI and for the advice you have offered over the course of the project.

Prof. Stephen Bitar for your advice, motivation, and support over the four years I have spent at WPI.

Joseph DiChiara and Julian de Zulueta for being a part of this incredible experience and for being great friends.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

In recent years there has been much motivation for the design of integrating many capabilities on a single chip. Motives have included smaller package sizes for more portable devices, reduced power consumption, and higher performance for real-time applications [1]. These chips incorporate the functionality of several devices into one die. They are thus called *systems-on-a-chip* (SoC). The SoCs use a higher level of integration which reduces the need for long wires and routing between many chips that often lead to long propagation delays and much power dissipation.

A system-on-a-chip approach is often used in types of embedded digital signal processing, where a system constitutes a potentially large number of interconnected devices. Each processor includes a large number of different *processing elements* that operate on the incoming signal data. Real-time processing performance is often a requirement of such devices, warranting the SoC design paradigm [1]. More recently, such systems have also been composed of predesigned logic blocks, called *intellectual property cores*, or IP cores. These are distributable designs (in the form of synthesizable RTL) that can be inserted into a SoC [2]. This greatly reduces design time for large devices and promotes reusability of designs.

Bus interfaces have been a traditional solution for control commands and the exchange of information in digital systems. A typical microprocessor accesses memory and I/O devices through a bus, often sharing these resources with other microprocessors in a multiprocessor system. While bus systems are typically straightforward designs, they come at the cost of performance and hardware resources. Busses demand a large number of wires to interconnect devices they contain. In addition, central control (arbitration) is required to ensure that data reaches its destination as well as a fairness of resource usage is enforced. Bus-based systems also have a limited capability and bandwidth for simultaneous transfers [2].

The need for alternative communication architecture became apparent as the number of devices connected in such a system increased. A traditional bus system does not offer scalability for systems in the size of potentially hundreds of connected devices. It was at this time that system designers began implementing concepts of computer networking on silicon devices, known commonly as *network-on-chip* (NoC) [2].

1.1 Background

A network-on-chip is a tool for communication between processing elements and storage devices in a system-on-chip. They are used to connect large numbers of devices that a traditional shared bus cannot. Early NoC implementations were seen in supercomputers and telecommunication devices [3]. Data moves through a NoC as it would move in a computer network. Elements in the NoC are responsible for locating the destination of transmitted data and assuring its delivery.

NoCs offer the advantages of greatly reducing wiring complexity and much improved scalability. The comparable-sized shared bus implementation could easily result in poorly scalable wiring, as the wire lengths to devices would increase with the devices' distances apart. This stands to be one of the greatest complications in making smaller chips [4]. NoC offer more compact solutions that bring routing elements closer together, offering shorter and more scalable interconnects. Excessive delays from long wires often exhibited in bus-based systems are likely avoided. Also, a NoC implementation relies less on dedicated point-to-point interconnects between devices and the bus. Process elements are connected via common switching and routing elements, over which the devices share the link utilization. Ultimately, better resource utilization is obtained [3].

1.2 Problem Statement

The MITRE Corporation sponsored this project to replace communication architecture used in the existing embedded digital signal processing chips, called a *control plane*. It is a separate data path reserved for command and control information, such as device configuration and status messages. The control plane is used to configure these waveform devices through writable registers or memories and reading information from these devices. Real-time signal is not handled by the control plane. Such data is routed through dedicated high-performance point-to-point interconnects outside of the control plane. These data paths handle constant data traffic, whereas the control plane oversees infrequent control traffic.

The previous implementation of a control plane suffered from many limitations. The new control plane must exhibit more flexibility, as the requirements having been expanding with each new project. In the past, issues such as latency and throughput were sacrificed in favor of

simplicity and faster development cycles. Future designs may require more scalable architectures with respect to these parameters. The previous control plane was also an application of a network-on-chip. However, its architecture suffered from many issues, such as high latency and poor link utilization, with a large number of interconnected devices. As the MITRE design engineers have experienced, the number of devices needed in these DSP applications will continue to rise and the systems will become increasingly more complex.

2 Network-on-Chip

Network-on-chip (NoC) is an emerging multiprocessor communication architecture in VLSI design. This type of communication replaces a traditional shared bus between the many processing elements on the chip. NoCs use routing elements and switches to pass messages between one another rather than point-to-point connections characteristic of shared busses [2]. This area of VLSI design and research heavily depends upon the development of networking technology. However, they are very distinct fields each containing distinct applications. Network-on-chip for VLSI must take many parameters into consideration, such as chip area and power consumption [4]. The diagram in Figure 1 shows an example of a network-on-chip based design using routing elements to form interconnects between many processing elements.

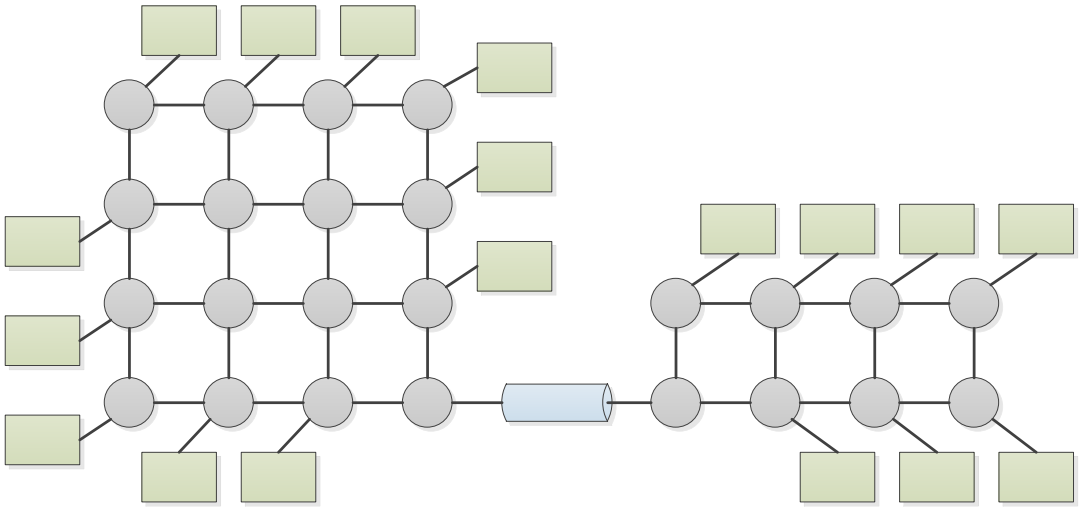

Figure 1: An example network-on-chip including routers (gray), a bridge (blue), and many IP cores (green).

A NoC implementation is defined by several parameters. These parameters are often selected to fit the needs of a design. For instance, these selections may favor low-power applications or favor high-performance systems. These parameters include (but are not limited to) the following:

- Topology
- Routing
- Switching
- Flow Control

Selections for each of these parameters carry its own advantages and disadvantages. There is no single selection of parameters that is favorable to all systems. General-purpose applications must consider the imposed limitations and the advantages gained by the selection of these parameters.

2.1 Topologies

The arrangement and interconnects between network nodes greatly effects a NoC's performance metrics. Such arrangements may send data through many nodes before reaching its destination. Other arrangements may provide more paths between nodes to minimize the number of nodes data must traverse.

2.1.1 Shared Bus

A shared bus is a popular selection for on-chip communication [2]. This is perhaps the simplest communication medium. In this method, the connected devices are divided into two categories: masters and slaves. Masters are capable of initiating requests and slaves respond to these requests. Data busses, often appearing as a collection of grouped wires, transmit data between masters and slaves [2]. In this topology, only one device can drive its data or control information at a given time, as this bus is a shared medium, as shown in Figure 2. Consequently, only one transfer can occur and there is a need for central control to ensure that data and control information are routing correctly.

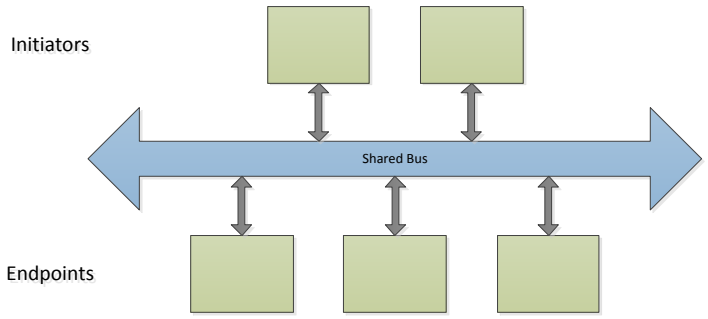

Figure 2: Shared bus topology.

Shared busses create an issue of scalability. The two prevailing strategies for a shared bus use either multiplexer-based control of the data bus or tri-states. With tri-state implementations the amount of wiring is greatly reduced, as each driver of the bus is connected directly to a bus lines through a small tri-state buffer. The buffer is high-impedance when the device is not

selected to drive the bus. Consequences of a tri-state implementation include higher power consumption and lower clock speeds from increased latency [2]. Multiplexer-based busses multiplex the masters' data and control lines out to global busses destined for the slaves. This implementation requires more logic for many multiplexers as well as much more wiring from each of the masters and slaves. This topology also suffers from degrading bandwidth when adding more devices [1]. In terms of power consumption, shared busses tend to consume more power since a bus master has to drive each of the lines connected to slave devices [4].

2.1.2 Ring Bus

A common communication topology is the ring bus [5]. In this topology routers are connected in a circle; each router node is connected to the previous node as well as the following node, as shown in Figure 3. Messages initiated on the ring are passed from node to node until the message reaches the destination node. This greatly simplifies the decision logic for router nodes resulting in smaller and faster logic.

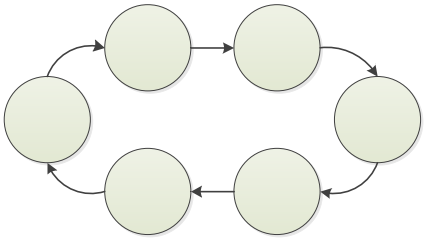

Figure 3: A ring bus.

Ring busses have the advantage of having shorter interconnects between nodes. By exploiting the nodes' spatial locality, adjacent nodes do not need long wire to connect them. Consequently, shorter wires contribute less latency resulting in higher communication speeds [1].

By nature, the ring bus can be pipelined. Each node in the bus can be considered a stage of the pipeline. This allows multiple messages to be inflight on the network at a given time. The pipelining of ring bus allows for a potentially high throughput in the system but suffers greatly in latency as the number of nodes increases.

2.1.3 Mesh Network

Another common NoC topology is a *mesh* [5]. This type of topology places network nodes in a grid-like configuration, like the network shown in Figure 4. Each node in a mesh is connected to four other nodes and likely a processing block as well. Messages are passed from node to node until reaching the node connected to the target device.

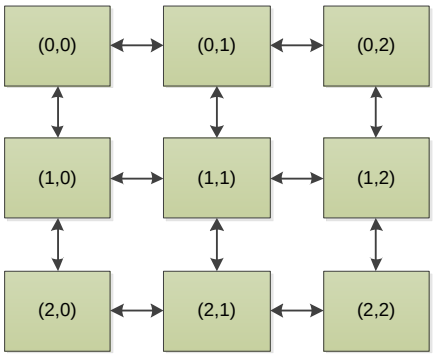

Figure 4: A mesh network.

Like the ring bus, a mesh network can have shorter connections between nodes. This eases wiring routing and placement on silicon as well as controls latency and power consumption through these links [5]. This again allows for higher operational clock speeds.

A particular advantage of the mesh topology is possibility for several messages to be inflight on separate pathways unlike the ring bus with only a single path. While this can reduce congestion and latency in delivery, it can also introduce other issues such as deadlock and out-of-order data.

2.1.4 Star Network

The *star* network topology works quite differently from the ring and mesh topologies. With the ring and the mesh, adjacent nodes are connected to one another to pass information around the network. A star network contains a central node that is connected to each other node. All network traffic must pass through this central node [6]. Figure 5 depicts the star network topology.

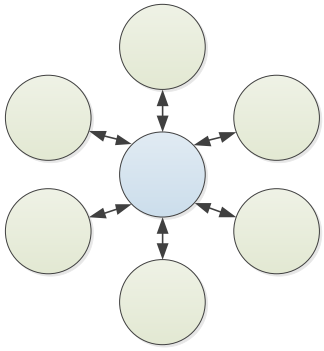

Figure 5: A star network.

Such a topology greatly simplifies routing. Peripheral nodes will always forward data to the central node. Data will only come from the central node, which eliminates issues of contention for that link. Routing in the central node requires minimal path-making decision, as it is connected to all other nodes. Despite the simpler routing, performance is heavily dependent upon traffic patterns. A busy and congested network will occupy the central node's resources needed by all inflight messages [6].

2.1.5 Comparison of Topologies

Each of the available topologies offers some advantage to NoC designers. For busses and star networks, that advantage is simplicity. The advantage of a ring or mesh network is having a better density (occupying less chip area). The chip area consumed by the NoC is a major factor in the network's scalability—how the network performs as the number of nodes increases. Due to the shorter and more local interconnects on mesh and ring networks, they are amongst the more scalable networks. Star and bus topologies quickly introduce wiring congestion in silicon and thus are limited in scalability. Ring and mesh topologies have been popular NoC choices because of the scalability [5]. However, chip area is not the only constraint on scalability. Table 1 lists the major differences between these common network topologies used in networks-on-chip.

	Complexity	Scalability	Wiring	Routing	Connection	Latency
Shared	Simple	Not scalable	Potentially long interconnects	Simple routing (arbitration)	Point-to-point	Dependent on load and traffic
Ring	Simple	Not easily scalable	Short interconnects	Simple routing; one path	Packets; forward to destination	Grows proportionally with number of nodes
Mesh	More complex than ring	Fairly scalable	Short interconnects	More complex routing than ring; multiple paths	Packets; forward to destination	Moderate; faster paths than rings
Star	Simple	Not easily scalable	Interconnects grow in length and number with number of nodes	Very simple routing	Packets; forward to destination in two hops	Congestion at the central node can lead to excessive latency

Table 1: A comparison of common NoC topologies based on important parameters.

Other factors in network topology scalability include power consumption, wiring complexity, cost, and latency [5]. The ring bus performs well in each of these categories but latency. A ring bus use short links between nodes and has considerably few links between nodes. This greatly simplifies wiring placement on-chip. Also, routing logic is nearly trivial on a ring bus. However, latency suffers greatly as a result. In the worst case, a message must traverse every node in the network.

The mesh topology reaches a fair compromise across these parameters for control planes similar to the one requested by MITRE. It shares the shorter interconnects as in the ring bus. Routing logic is more complex but not excessively taxing on hardware resources. Having many links per node introduces path diversity, allowing for shorter worst-case routing paths than the ring. Also, latency scales much better than a ring.

2.2 Routing

Routing is how nodes decide where to forward data packets. Nodes responsible for routing are typically called *routers*. Strategies for routing come in various degrees of algorithmic and hardware complexity.

2.2.1 Deterministic Routing

A set of routing strategies called *deterministic routing* uses predetermined network paths to route data through the NoC. That is, the path between any two nodes will always be the same. Deterministic routing does not take the condition of the network into account in its decision-making process. This is why it is sometimes called *oblivious routing*. With deterministic routing, simpler algorithms can be developed that require less logic [2].

2.2.1.1 XY Routing

A common type of deterministic routing is *XY routing*. This routing strategy is specific to 2D mesh topologies. Nodes can be thought to have an XY coordinate corresponding to its row and column in the mesh grid. Data is routed to adjacent nodes first in the X direction. That is, data is moved to the right column before navigating to the destination row (the Y direction).

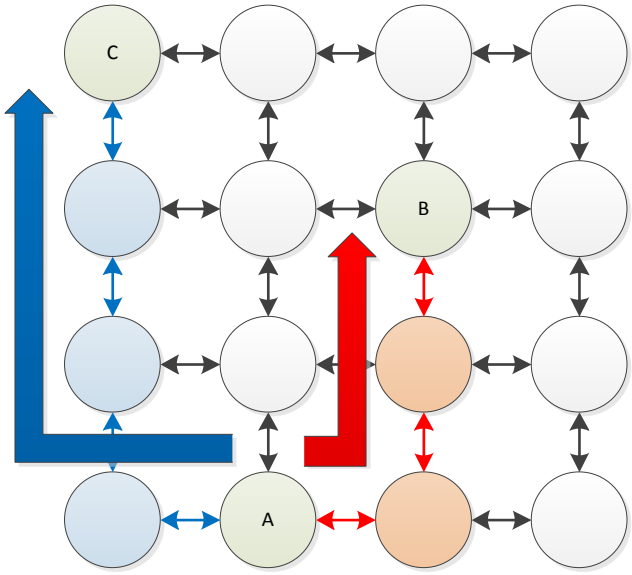

Figure 6: Routing paths by the XY algorithm for a request from A to B and from A to C.

2.2.1.2 *Source Routing*

Another type of deterministic routing is called *source routing*. This method does not require complex routing logic. Routing is determined by the initiator; each node transversal is determined and encoded in the message. This is a computationally light routing protocol but it suffers from increased message size from the overhead of routing bits [6].

2.2.2 Adaptive Routing

The other case of routing strategies is *adaptive routing*. In this class of algorithms, routers analyze the current state of the network. This information is used to find a path to the destination that is less congested than a direct path. The goal of this approach is to gain performance from reducing latency due to network congestion [3].

While in deterministic routing messages can be easily routed to the shortest paths, adaptive routing strategies do not necessarily make this guarantee. Messages can be routed along longer paths on the way to the destination in order to avoid congestion and stalls. These decisions are called *misroutes* and can have negative effects on performance.

A major issue that arises with adaptive routing is *livelock*. When livelock occurs, a message is continually misrouted, never reaching its destination. In applications such as the MITRE control plane, data loss is not acceptable and this issue would have to be resolved. Measures must be taken in the routing algorithm to ensure that all messages are able to eventually reach their destinations [3].

2.2.3 Comparison of Routing

The fundamental tradeoff between routing strategies is the trade of chip area and wire routing for performance. With deterministic routing, messages typically follow minimal paths but are susceptible to network congestion. Their routing logic is typically simpler than with adaptive routing strategies. Adaptive routing can take measures to avoid network congestion but requires more complex logic to ensure the prevention of error conditions such as livelock.

Ultimately, simple routing strategies like XY routing are favorable when hardware size is a concern and performance is not a primary goal. Depending upon traffic patterns this strategy

could still give good performance. In a latency-critical system a robust adaptive routing algorithm is likely a good choice.

2.3 Switching

While routing determines *where* messages are sent within a router it does not dictate *how* this data is sent. This is determined by the *switching* method. Two main types of switching techniques are used: *circuit switching* and *packet switching*, described in the following sections.

2.3.1 Circuit Switching

The first major technique, circuit switching, creates a link between the sender node and destination node. Before a message is transmitted, a request must be sent to allocate a physical channel between nodes. The destination node sends a notification back to the sender signaling the sender to transmit its data. The established path (or "circuit") between the nodes remains open until the message is received by the destination node. The intermediary nodes forming the connection are blocked from use in other paths until the current transfer is complete and the nodes along the path are released.

Circuit switching offers high throughput when the channel is allocated. Aside from the initial latency from the setup of the connection, there is no additional latency or stall when the channel is established [2]. Such a method favors low, infrequent traffic, possibly containing large amounts of data. This method would not be suitable for small messages, as the setup overhead would become more significant [1].

2.3.2 Packet Switching

The alternative to circuit switching is known as *packet switching*. This method of switching uses *packets* that carry the information necessary for routing. Rather than setting up the connection prior to transmission, connections are instead made as a packet progresses through the network. It is common that packets are subdivided into smaller units called *flow-control digits*, or *flits*. A packet contains a *head* flit, potentially many *body* flits, and a *tail* flit. The head flit is the beginning of the packet and contains the necessary routing information for the packet. Head flits serve to allocate router channels for the remaining flits of the packet. The final tail flit of a packet frees the router channel [3].

Three main strategies are used in packet switching: *store-and-forward*, *virtual cut-through*, and *wormhole switching* [2]. The simplest of these is store-and-forward (SAF). This technique buffers the entire received packet before transmitting it to the next node. Virtual cut-through (VCT) switching works in a similar manner. The VCT method allows flits to move to the next node as soon as space becomes available. While space is not available, flits are buffer locally.

The final method is wormhole switching (WH). In this method each flit moves one at a time to the next node. This creates a string of nodes carrying individual flits of the packet, as shown in Figure 7. By this method, flits do not accumulate at a single node when stalled. An advantage of this is that each node is not burdened by large buffer requirements [2].

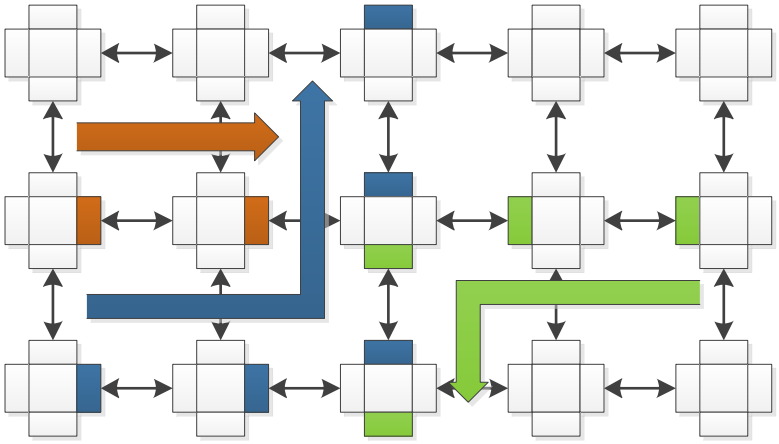

Figure 7: An example of wormhole switching.

2.3.3 Comparison of Switching

The main difference between circuit and packet switch methods is to how well they address network traffic patterns. The overhead of establishing a channel in circuit switching can be wasteful if many messages need to be sent often. This does not harm packet switching, as each packet is capable of routing itself. In packet switching packets do not enjoy the same guaranteed performance as in circuit switching. Since the link is allocated to the inflight transfer in circuit switching, no other transactions can stall it. In packet switching many packets may be contending for a common network resource.

Wormhole switching has the advantage of using less buffering, therefore reducing its footprint on the network. SAF and VCT both require full packet buffers to store backed-up data.

In wormhole switching, only a single flits need to be buffered. Wormhole and VCT switching also offer better throughput and latency as the entire packet does not need to be buffered in each node like in SAF. These two methods, however, have the potential to create more network congestion, as their packets can span many nodes, like in the example in Figure 7. The orange packet is attempting to allocate the North buffer of next router that is held by the blue packet. They also run a greater risk of producing deadlocks [2].

2.4 Flow Control

Link-level flow control is the method by which the network ensures data integrity. It is at this level data loss is prevented. This level of communication must respond appropriately to network conditions such as stalls.

2.4.1 Stall-and-Go

In a Stall-and-Go flow control based system adjacent routers signal one another when they are ready to receive data. There is a stall signal on each channel of the router directed at the incoming data link (the "upstream" router) and another coming in from the outgoing link (the "downstream" router). Routers produce a STALL signal when they are not ready to receive data otherwise a GO signal is given. When a channel is given a downstream STALL signal, the router must suspend transmission until given a GO signal.

In digital logic, such signals would be given as sequential outputs. That is, they are updated on one clock edge by the sender and observed on the next active edge by the receiver. It should be noted that this is not adequate time for the stalled node to pass on the STALL to other downstream routers in the same cycle. For this, each channel requires a small flow control unit.

The input port to a router must have the capability of buffering two flits. In normal operation (GO), only one register is needed. However, since stalls cannot be passed upstream instantaneously the stall router must have a backup register to capture this transmission [2]. Once the STALL signal is captured on an active edge, it can be simply repeated on that channel's upstream stall signal. That is, the incoming stall from the receiving end becomes the next outgoing stall on the transmitting end. This prevents data loss over the link. Figure 8 contains can example of this behavior.

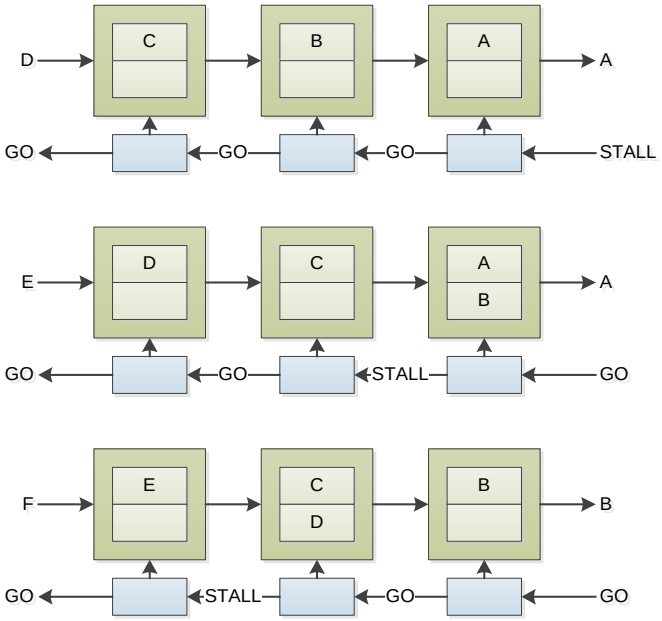

Figure 8: An example of Stall-and-Go flow control.

2.4.2 ACK-NACK

An alternative strategy to Stall-and-Go flow control is ACK-NACK flow control. This method places the burden of buffering on the transmitter rather than the receiver. The transmitting node must keep the flits that it transmits until receiving an *acknowledgement* (ACK) from the receiver. If a *negative acknowledgement* (NACK) is received, all flits buffered that have received an NACK must be retransmitted [2].

2.4.3 Comparison of Flow Control

The main difference between the two described methods is buffer overhead. Since these flow control units must be a part of each link in the network it is important to reduce their contribution to chip area and power dissipation. The Stall-and-Go method only requires two flit buffers and a simple state machine to maintain data integrity. The ACK-NACK requires more significant buffering requirements to retain several transmitted flits. Also, ACK-NACK potentially retransmits flits that were not lost or corrupted thus adding to the delivery latency. Stall-and-Go introduces no additional latency penalties from stall recovery [2].

2.5 Complications of Networks-on-Chip

Each of the discussed strategies offers many strengths to a NoC but can also introduce vulnerabilities and complications. Each of these shortcomings can have detrimental effects on the functionality of the system or on the integrity of transmitted data. Measures must be taken to address these issues in order for a NoC to be reliable.

2.5.1 Deadlock

A common threat to data integrity in a network-on-chip is *deadlock*. Deadlock occurs when contending packets prevent one another from advancing indefinitely. One should consider two cars facing one another on a narrow road. Neither car can continue until the other moves out of the way. Deadlock can suspend the operation of large portions of the NoC and is thus unacceptable in any NoC-based system. In general, a NoC designer must be acquainted with two types of deadlock: *routing-dependent deadlock* and *message-dependent deadlock* [7].

2.5.1.1 Routing-dependent Deadlock

A *routing-dependent deadlock* condition is caused by contention of network resources as a consequence of message routing [7]. Such deadlocks are purely a consequence of the routing algorithm, network topology, or both. A popularly referenced example of deadlock is depicted in Figure 9. In this example four packets form a square of adjacent nodes in a mesh network. Each packet wants to cycle counterclockwise. However, the channels necessary to do so are already allocated. They will not be available for the requesting device until the new channel is allocated. This produces a deadlock as none of the packets will ever be able to advance.

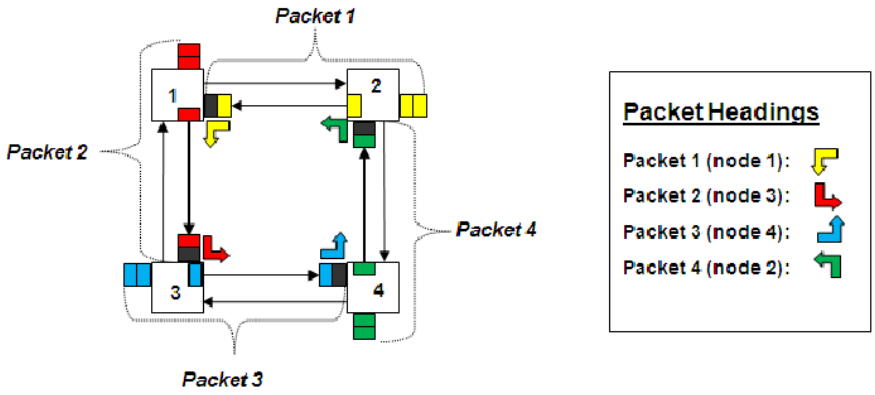

Figure 9: A classic example of deadlock [8].

The possibility of a deadlock on a mesh network can be predicted with the *Turn Model*. This method relates the dimension of the mesh to the number of illegal turns (ie. North-East, South-West). For an *n*-dimensional mesh, the number of illegal turns must be at least be $n(n-1)$. The Turn Model proves that on a 2D mesh network is deadlock-free, as four of the eight possible turns are illegal, as shown in Figure 10 [8]. Since packets must be routed in the X-direction first, the turns South-East and North-West are not legal, for example.

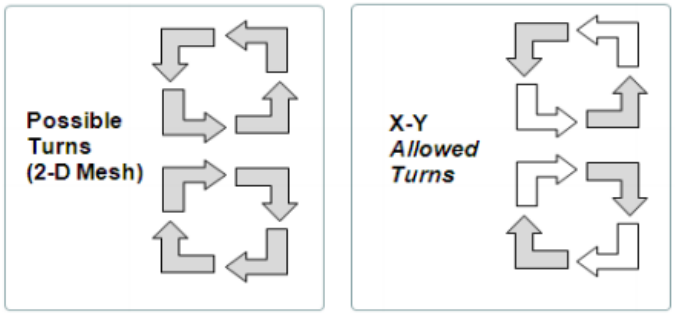

Figure 10: Allowed turns in XY routing [8].

2.5.1.2 Message-dependent Deadlock

Message-dependent deadlock is the consequence of dependences in higher levels of the NoC protocols [9]. In particular, this occurs from collisions between different types of messages used by network devices [7]. One such example of message-based deadlock is called a *request-response dependency* [9]. This issue arises from an endpoint's response message being stalled in the network by a request message. For instance, in the situation where an initiator is sending a message to an endpoint while the endpoint is responding to a previous message. The endpoint cannot *consume*—or take the message out of the network—until it has transmitted it response message due to buffering requirements. The incoming request to this endpoint is now stalled in the network. Switching implementations such as wormhole switching exacerbate this issue as many routers and channels are blocked until this message can be consumed. In the situation where a second device is sending a message that requires the channels occupied by the first initiator's request. Additionally, the first endpoint's response requires the channel held by this new message. Ultimately, the message from the other device blocks the first endpoint's response. The response message prevents the request from being consumed. The request message prevents the other message from proceeding that is blocking the response [7]. This forms a circular

dependency between the progresses of the three message, guaranteeing deadlock, as shown in Figure 11.

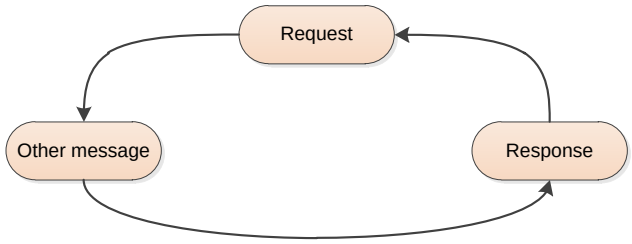

Figure 11: An example of dependencies between network messages.

Typically one of the four following solutions is used to avoid message-based deadlock: physically separate networks, virtual networks, buffer-sizing, end-to-end flow control [7] [9]. The first two solutions approach the issue of message-based deadlock by dividing the data flow of request and response messages. This is referred as *strict ordering* [9]. The last two solutions attempt to use buffering to remove the possibility of deadlock.

The buffer sizing solution works on a principle known as the *consumption assumption* [9]. The assumption states that an endpoint must consume all messages it receives. The ideal solution to this problem is an infinitely large buffer to store all incoming messages. Removing the messages from the network deallocate the link resources used by the message and therefore avoid these messages from causing deadlocks with messages dependent upon these links [7]. No such buffer is possible in a real-world application, however, so buffers can only be made adequately large to store incoming messages before responding. This practice is very costly in terms of chip area from the significant buffer requirement and is unknown in network-on-chip applications [9].

End-to-end flow control is also based on the consumption assumption. In particular, credit-based end-to-end flow control dictates that for each connection between a pair of devices a number of credits are issued informing the sender of how many messages the receiver is capable of receiving. As a result, more effective and practical buffer sizes can be attained. However, cost associated with maintaining the credit system in is a burden on the network interface hardware [9].

Using physically separate networks avoids the issue of these dependencies entirely [7]. Since requests and responses do not have to contend for the same resources there are no potential

dependencies and thus no chance of deadlock. Of course, this comes at the expense of additional hardware to construct the second network. A physically separate network resembles the network in Figure 12.

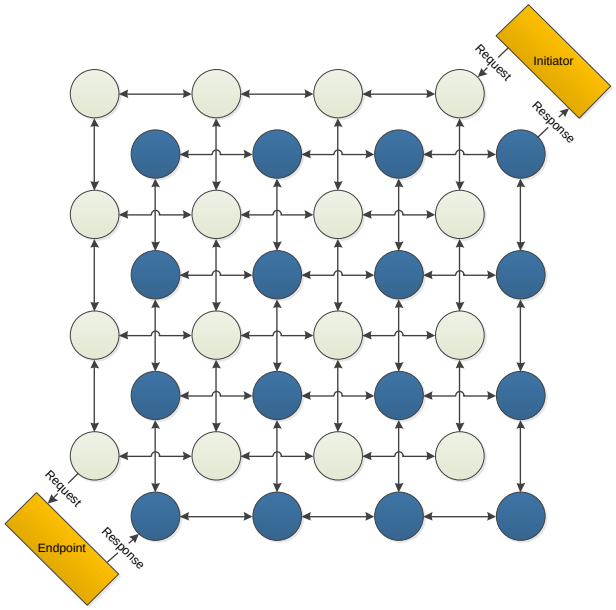

Figure 12: Physically-separate request and response networks to break message-based deadlock.

The other type of strict ordering uses separate virtual channels rather than separate physical channels. In this system a physical channel is divided into several virtual channels. Each virtual channel contains a buffer for storing a portion of a message (such as a flit). Each message type is assigned a virtual channel. The virtual channels composing a physical channel are multiplexed such that one virtual channel drives the physical transmission medium. This requires additional arbitration logic to ensure that the virtual channels are fairly assigned to the physical channel [6]. Like the physically separate network virtual channel cannot use the resources associated with another channel, as they are strictly allocated for that particular resource. Additionally, link utilization suffers from the inability of the channel to transmit simultaneously [9].

2.5.2 Livelock

Livelock is an issue that arises in adaptive routing systems. A packet in livelock will be continually routing along a path that will never reach its destination. This could be in response to patterns of congestion in the path to the destination node.

Livelock can be resolved in a number of ways. One simple way to avoid this complication is by using a *time-to-live* (TTL) counter in packets. The counter is decremented at each node traversed. When the counter reaches zero, the packet is discarded. This prevents the packet from wasting network resources. Another solution to livelock is to introduce age-based priority rules. As the packet circles it destination, it will eventually preempt the offending traffic and reach its destination [6].

2.5.3 Starvation

In NoC and bus-based systems some devices may have the ability to initiate enough requests to effectively block another device's requests from being accepted. This is called *starvation*. This is a great vulnerability in purely priority-based systems [6]. Starvation can be avoided by algorithms that consider *fairness*, should as a *round-robin* system. In a round-robin system, the last granted source is set to the lowest priority [2]. This prevents the same source from dominating the request process.

Another way of preventing a starvation is *time division multiple access* (TDMA). This method gives each source the chance to transmit for a certain fixed amount of time. Each source transmits in a fixed order. This ensures that all sources have a chance to access network resources [1]. A disadvantage of this system is that sources without data to send are still allocated bandwidth. This reduces the link utilization of the given channel.

3 Requirements and Specifications

The previous MITRE CRB control plane is lack of many desirable features for reusable IP. Such issues included a lack of documentation, lack of expandability, and strict limitations on system parameters and characteristics. The sponsors from MITRE Corporation designed this project to build a new IP package for a control plane to suit the needs of both current projects was well as future projects.

3.1 General Goals

A major requirement for this project is the development of significant documentation. The CRB IP included minimal documentation. It had basic descriptions of the intent of the written RTL code but was difficult for others to understand. No formal specification or user guide was provided. Consequently, the new control plane was required to be thoroughly documented in RTL in form of comments as well as in a formal specification sheet and user guide to assist in deployment. This specification was to include detailed descriptions of component design as well as detailed descriptions of the non-standard protocols and interfaces.

The MITRE Corporation is interested in a general-purpose control plane platform. This new control plane is not intended for a specific application or project. It must be flexible for a wide range of applications. For example, the control plane has to support both high-performance applications as well as low-power applications. As a result, parameters such as power dissipation, wiring delays and latency, and chip area had to be considered in the design.

Another limitation of the previous CRB design is that it only allowed for a single initiator. Only one device could initiate requests onto the bus. The new MITRE control plane is required to allow for not only multiple initiators but for a large number of initiators. The single initiator limitation from the CRB could potentially be an issue in many future projects at MITRE. The new control plane was required to support many initiators to remove this limitation on future systems.

The sponsor’s vision of the new control plane includes a priority on the reuse of this IP package. The aforementioned considerations each contributed to the reusability of the new implementation. The control plane is required to be designed with the ability to add additional interface protocols. The sponsor also requests a convenient and well-documented method for

adding other protocol layers over the NoC implementation. This is intended to reduce the design and verification resource needed to modify existing IP (either in the control plane or interconnected IP cores) to conform to new protocols used in future designs.

Another major requirement of the project is the verification of all IPs used in the control plane package. This step is to ensure that any future project to include the control plane IP can be assured that the IP behaves according to specification. This is intended to alleviate design engineers who will be using this IP from the burden of verifying this interconnect logic in each design. The control plane IP is intended to be distributed with detailed tests and test plans that demonstrate its functionality and validation.

3.2 Control Plane Specifications

The design goals for the new control plane in Section 3.1 describe the general goals for the new system and their motivations. These goals shaped a list of specific requirements that guided the design methodology for this IP package. The following list enumerates these design requirements.

- *Scalability* – The control plane must be adaptable for a wide range of applications including but not limited to high-performance systems and low-power applications. The design should allow for a multilayered system.
- *Size* – The system must support a large number of devices; up to 1024 endpoints and initiators.
- *Performance* – Efforts to control congestion and excessive delays due to wiring must be taken to ensure desirable performance and scalability.
- *Reliability* – Reliability is possible the most important aspect of this system. It is imperative that no data is lost and all transactions complete. The system must not be vulnerable to complications such as *deadlock*.
- *Interfaces* – The control plane must offer the capability to support at least the following standard interfaces to ensure compatibility for commonly used devices:
    - MITRE OpenCore Protocol (OCP) Memory[1]
    - Serial Peripheral Interface (SPI)

[1] MITRE implements a limited subset of the OCP Memory protocol, as described in [12].

  - "SRAM" interface – providing a basic address-data interface
  - AMBA 2.0 Advanced High-Performance Bus (AHB)

  It should be noted that the Control Plane is not to be limited to neither a particular interface nor overlying protocol.

- *Error Reporting* – A mechanism must be provided for endpoint devices to provide detailed error reporting to requesting initiators.
- *Documentation* – The system is to be documented both in RTL code as well extensively in a separate specifications document. This is intended to promote the reuse of the final deliverable.
- *Verification* – Full and thorough verification of this system is greatly important. Guaranteeing correctness of operation prior to deployment will significantly reduce design time of future projects and reduce the required man-hours for adequate verification. A set of tests must be provided with the system demonstrating its correct operation as well as recovery from errors and rare or unforeseen corner cases.

3.3 Project Deliverables

The planned deliverables for the new control plane are the following:

- An IP package encapsulating the functionality of the control plane.
- A specification document describing the major functional components and their interfaces.
- A detailed test plan and set of testbenches demonstrating the functionality and correctness of the IP package.
- A user guide to facilitate using and extending the delivered IP package.
- A code repository of all files used to create and test these deliverables.
- A briefing to the MITRE E536 department on the development and use of this IP package.

4 Methodology

Careful consideration is taken to ensure that the specifications for the control plane would be sufficient for MITRE's applications and projects in the near future. Each design alternative presented in Chapter 0 was considered for how well it may satisfy the sponsor's needs. This is to ensure finding a sufficient and adaptive solution for on-chip communication architecture.

4.1 Topology

Multiple topologies were made possible in this control plane to comply with the sponsor's requirements. The MITRE control plane was designed to support several *local* mesh networks. The overall structure of the network—the *global* topology—was allowed to be configured by connecting local meshes with a *bridge* component.

The mesh topology was selected because of its balanced parameters with respect to scalability. This topology places routing elements relatively close together, reducing wiring latencies and place-and-routing congestions in synthesis. The mesh topology also offers simpler position-based routing algorithms that simplify hardware design and reduce chip area costs.

The use of a multi-layered topology allowed groups of related devices to be grouped together in an efficient fashion. By dividing into many meshes, the dimensions can be configured in a more efficient fashion. For example, without local meshes the whole NoC would be included in a single mesh. Devices would have to be placed in a larger number of rows and columns, potentially increasing message latencies. Additionally, unused locations in the mesh would be wastefully allocated and synthesized. Hardware efficient can be gained by dividing the NoC into several meshes of related devices hardware efficient can be gained. As these local meshes will likely see infrequent traffic with devices in other groups (meshes) the need for a higher-performance links between mesh routers is unnecessary. Therefore, the system could benefit from connecting these meshes with lower-performance bridge components.

4.2 Routing

The XY routing algorithm was selected for routing within local meshes. Each node in the NoC was decided to be assigned a unique identifier including the following information: the *mesh identifier*, the *x-coordinate*, and the *y-coordinate*.

The local mesh routing algorithm was selected to be simple and fast. The simplest of routing algorithms come from the class of deterministic algorithms. XY routing offers a simple routing strategy that can be efficiently mapped into hardware. Advantageously, XY routing eliminates the possibility of routing-based deadlock.

Adaptive routing strategies have the potential to offer better latency from avoiding congested network paths but introduce many complications. Network devices would have to be concerned with out-of-order received packets, deadlock conditions, and also livelock. Additional hardware in the routers and network interfaces would be needed to prevent or correct these issues. Additional data such as sequence numbers and time-to-live fields would be necessary in packets, increasing the message overhead.

The selection of XY routing greatly simplified the role of the router component. The router routes packets based on a small target address field in the packet. For better latency and buffer performance, this target address field was constrained to be contained in the head flit of the packet. The address field contained the local mesh's identifier number as well as the X and Y locations of the target node. As this implies, the router was only responsible for routing within the local mesh. The router component was effectively unaware of the rest of the NoC to which it was connected. This system-level awareness was built into other NoC components.

4.2.1 Bridge

Since the router components could only route within their local meshes an additional component was needed to route data globally. The bridge component was used as a channel between two meshes. Packets entering the bridge from one mesh were passed to the mesh on the other end of the bridge. These packets passing through the bridge were translated to into the addresses known in new local mesh. As a consequence the bridge needed to be aware of the overall system. The target address used in the local mesh addressed the bridge on that network. Additional information, such as the original sender or the request address, was used to find the next bridge to traverse or the destination device. Such routing could be easily achieved through look-up tables (LUT). These look-up tables are generated at design time, likely by a software deployment tool. The simplicity of this solution came at the expense of cost in terms of hardware and chip area. The network diagram in Figure 13 shows an example of the request translation process.

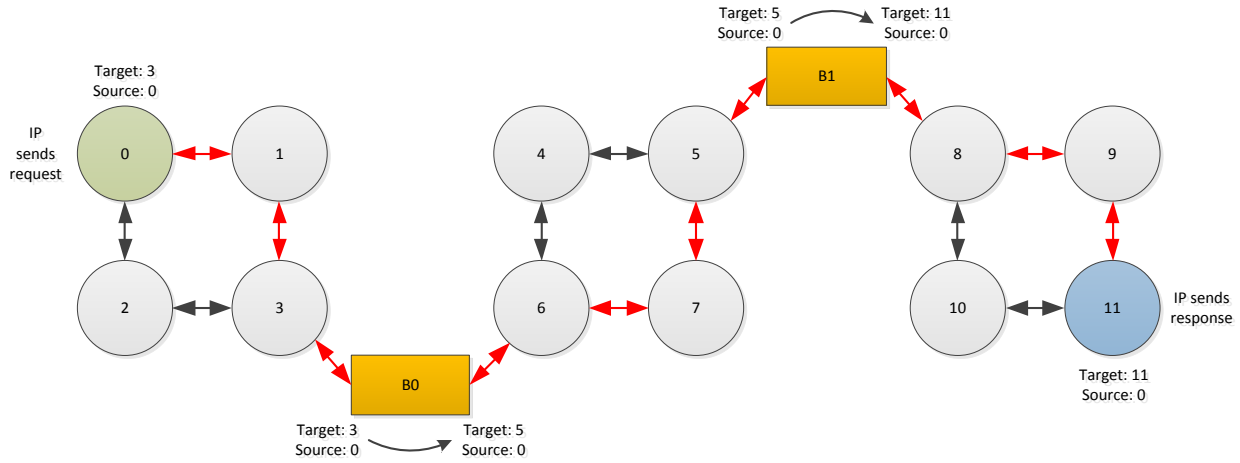

Figure 13: An example of address translation for initiator requests across a bridge.

A concern for the architecture of a bridge component is the ability for a response packet to return to the initiator device. In order for a packet to be able to return to the sender, the sender must be known. This was decided to be encoded in the packet. A separate *Source* would be used to encode the initiator's address. Just as the bridge needed a lookup table to find the next target for a request, it also needs a lookup table for responses. This lookup table, however, uses the *Source* fields rather than the base address. The diagram in Figure 14 shows an example of this response translation process.

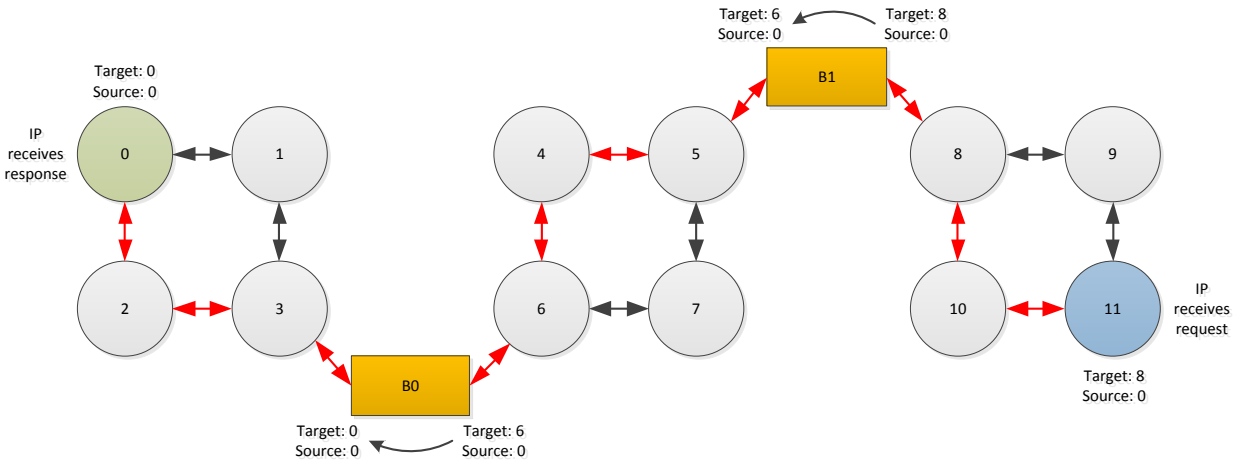

Figure 14: An example of address translation for endpoint responses across a bridge.

4.3 Switching

When the selection for the switching strategy was made the advantages of circuit and packet switching were matched with the sponsor's requirements. It is determined that traffic on the NoC would be considerably infrequent and have potentially large messages. Such a traffic pattern matched the advantages of circuit switching. This type of switching offers better throughput to long messages as the channel is allocated prior to transmission. However, packet switching is selected to simplify hardware. It was determined to be simpler to allow packets to allocate network and channel resources as they progressed through the NoC rather than pre-allocate them as in circuit switching.

The type of packet switching that was selected for the MITRE control plane was wormhole switching. This type of switching was selected because it offered a low buffering cost per router. As each channel of each router would require these packet buffers this buffering cost could become quite significant in the chip area of the entire NoC. Wormhole switching also allowed for better throughput than the SAF and VCT methods. The wormhole switching was implemented without virtual channels. Virtual channels were not implemented in order to avoid the additional complexities of virtual channel allocation. This was motivated by an effort to keep the underlying NoC components simple.

4.4 Flow Control

The most appropriate method of link-level flow control for the control plane NoC was determined to be the Stall-and-Go method. It was selected because of its low overhead implementation and high degree of reliability. Stall-and-Go had the advantage of have low buffering costs which was important in controlling the chip area consumed by the NoC. This was an important consideration as the flow control buffering unit was used on each input link in the NoC.

Additionally, Stall-and-Go offered good stall recovery. In Stall-and-Go, recovering for a stall did not incur additional latency, allowing for fast stall recovery. Unlike the other flow control option ACK-NACK, the Stall-and-Go method did not require the retransmission of packets.

4.5 Network Interface

IP cores needed a way to send and receive data on the NoC. This capability was built into the network interface. IP cores connected to the NoC via dedicated links on the router components called the *local* link. It was through this link that data entered and exited the NoC. Each endpoint or initiator on the NoC was assigned a router on one of the local meshes. For endpoint IP cores the addresses in the endpoint's address range on the NoC would translate to the address router to which the endpoint was connected. When a router received a packet destined for its own address, the packet would be routed to the local port. This system differed from the previous MITRE control plane that implemented the network interface inside the router component. Such an implementation limited the flexibility of the router component as it needed to be connected to an IP cores (consequently required the development of repeater component). Also, only one protocol—OCP Memory—was implemented on the CRB.

A requirement of the new MITRE control plane was to allow for several standard interfaces connected to IP cores as well as the easy integration of additional protocols in the futures. This was facilitated by decoupling the network interface from the router component as described previously. Additionally, the network interface was divided into two components: the *packet processor* and the *protocol adapter*. The network is illustrated in Figure 15.

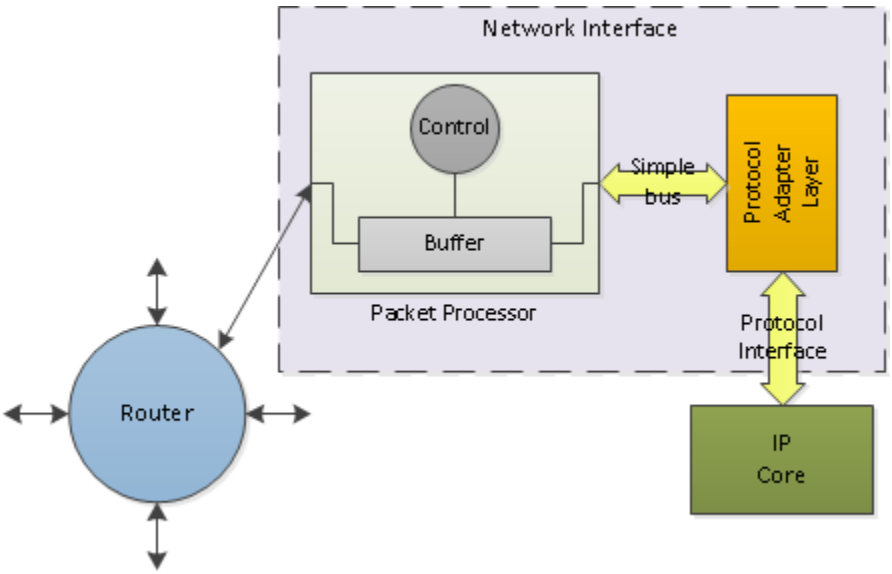

Figure 15: Composition of the Control Plane network interface (NI).

4.5.1 Packet Processor

The objective when designing the network interface was to make the details of the NoC transparent to the IP core. This was implemented in the packet processor layer. This layer of the system provides two buffers: one for data entering the IP core from the NoC and another for data exiting the IP entering the network. As seen in Figure 15, the packet processor connects to the protocol-specific component called the *protocol adapter* by a simple shared bus. This bus offers a wide data bus for passing the raw packet into the protocol adapter for further processing. A simple handshaking mechanism was provided to allow the protocol adapter to stall incoming data.

4.5.2 Protocol Adapter

The protocol adapter was the layer of the network interface that directly interfaced with IP cores. It was at this level that data would ultimately be exchanged between the network and the IP core. This functionality was isolated to this component to allow for easier and faster integration of additional protocols to the control plane. Design engineers were intended to use the packet processor component to connect to the NoC and follow a basic template for packet generation. The design engineer is responsible for designing the IP interface protocol as well as control logic for the packet processor bus.

As mentioned previously, the lower level packet processor removed many aspects of the underlying network, such as flit headers and stall signals. Other aspects such as packet structure and network node locations, however, were still known to the protocol adapter. Such knowledge of the network was necessary in the packet generation process for initiator IP cores. Such protocol adapters needed the capability to interpret a request address from the IP core and produce a network address that routed the generated packet to the destination node.

The concept of a separate protocol adapter allowed the concept of custom packet structures to become simpler to implement. The packet bits not used in routing were ignored by the routers and bridges. This data was simply passed along to the next routing element. Similarly, this data was not affected by the packet processor. Packet processors were given the potential to process additional application-specific information.

4.6 Packet Structure

The new MITRE control plane offers a flexible packet structure. In order for simple, general-purpose routing elements to be designed, some restrictions had to be imposed, however. Figure 16 illustrates the general packet structure of a packet used in the new NoC control plane. It should be noted that the fields necessary to routing were positioned at the beginning of the packet for minimizing routing buffer costs.

Figure 16: General packet structure for the MITRE control plane.

The first fixed field in the control plane packet is the *Target* field. This field is a *mesh address*, meaning it contains a mesh identifier number, an X location, and a Y location. This address identifies the node to which the packet is being routed on the current local mesh. This field is positioned at the beginning of the packet so that routers only have to read the head flit in order to allocate the correct channel on which to transmit the packet. The *Source* field identifies the mesh address of the original sender of the packet. This field is used by endpoints and bridges to generate the target address of a response message. The *Type* is used to identify the type of network packet. This system implements two packet types: *requests* and *responses*.

The next two fields are the *base address* and *local address.* This system is very similar to that used in the previous MITRE control plane implementation. Each unique base address identifies a single endpoint IP core. The local address acts as the request address for the endpoint device. This address field is required to be sized according to the largest device address space. The motivation for such as scheme is for smaller decoder logic for the lookup tables for initiator protocol adapters and bridges when determining the new network address from the request address. This makes it necessary for these routing elements to only buffer the base address and not the local address.

The remaining bits of the packet are left open for application-specific use. Bits that would likely reside in these bits are operation codes, read and write data, and error values. By not imposing requirements on these bits additional protocols and functions can be added in the future. For example, a new command can be implemented to perform a write to the target device without generating a response packet. Imposing such packet structure limitations would have

hindered much future expansion and alternative use of the MITRE control plane. With reuse and expandability at the forefront of the project requirements, such restrictions are avoided.

5 Implementation and Design

This section provides a detailed view of the underlying architecture of the control plane. The methodology developed in Section 4 is implemented with a bottom-up approach. The fundamental components are built and tested first. After successful testing of one layer the next layer is then built over it.

5.1 Flow Control Buffer

The Flow Control Buffer (FCB) is used to ensure that no data would is lost across data links. The buffer uses the Stall-and-Go type flow control to control traffic through the buffer. In this implementation, the buffer is designed with two flit registers. Under normal operation (no stalls) the *forward* register receives flit transmitted across the input flit bus, or link. This register is used as the output register of the block as well. The second register, the *save* register, holds flits that would otherwise be lost in the stall. The block diagram in Figure 17 shows the implementation of the Flow Control Buffer.

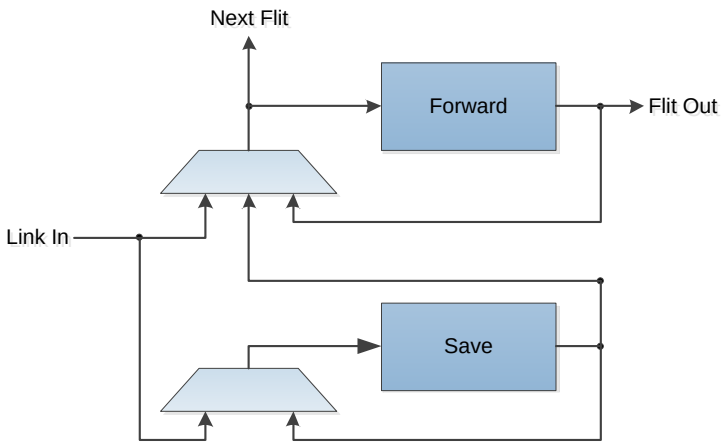

Figure 17: Block diagram of Flow Control Buffer.

The need for the component arose from the complications of passing the STALL signals through sequential logic. When a router is signaled STALL from a downstream (receiving) router, the STALL signal is not processed by the stalled router until it has transmitted its flit. Consequently, the downstream router must have buffer space to receive this transmitted packet. This is the purpose of the Save register. The Save register stores the value of the incoming flit when the channel is initially stalled. The value in the Forward register is retained until a GO

signal is received, at which point the flit has been accepted and stored by the downstream router. It is also at this point that the Save register is copied into the Forward register.

Aside from its buffer responsibilities the Full Control Buffer must also remember its stall state as well as propagate stall information upstream. A simple state machine retains the stall state of buffer. It contains a single input: the downstream stall signal from the router to which the buffer's output is connected. The state machine always progresses to the STALL state when the input is STALL and always transitions to GO when the input value is GO. The state transition diagram is shown in Figure 18.

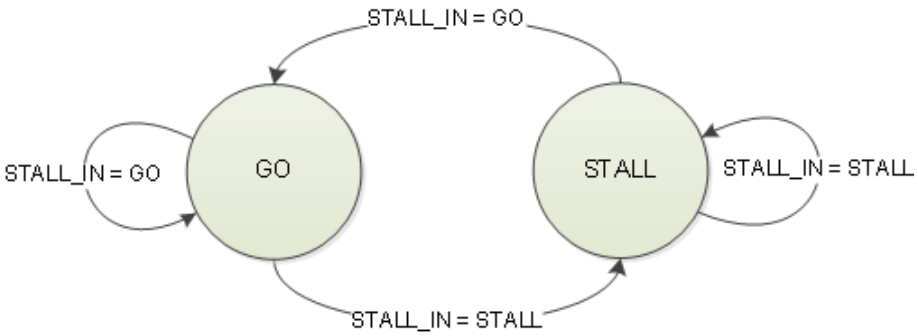

Figure 18: State transition diagram for the Flow Control Buffer state machine.

Upstream stalls are an important part of the design of the Flow Control Buffer. When a STALL is received from a downstream router, both registers in the buffer are now occupied and the buffer cannot store any more data. Consequently, the buffer must indicate to the upstream router that no more data can be accepted so a STALL signal is generated back. When the buffer is given the GO signal the save register is now empty. The upstream router can now be signaled that data can be transmitted again. Given these facts, the upstream stall signal can simply be the downstream stall signal. Since it is a registered (sequential) output the downstream input value will not appear on the upstream output at the next clock edge.

A special functionality is built into the Flow Control Buffer. An additional output is provided for use in the router component. This is a combinatorial output supplying the next flit to be transmitted. According to the stall values and current state, the next flit could be either the current flit (in the Forward register), the flit in the Save register, or the flit on the input link. An updated version of the Flow Control Buffer block diagram is shown in Figure 19 reflecting this additional output and the stall state.

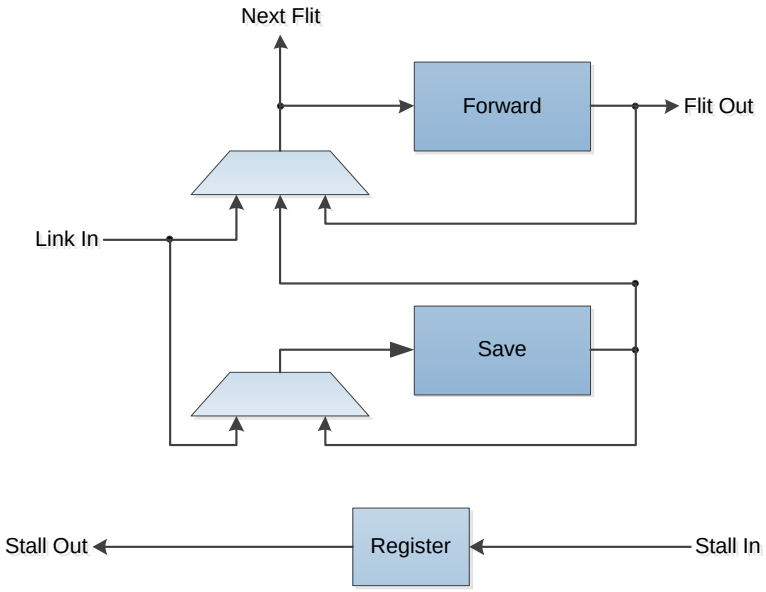

Figure 19: Final block diagram of the Full Control Buffer.

5.2 Router

The Router component is the fundamental building block of the Control Plane. This component is responsible for transporting packets within local meshes. It is shown in Section 5.3 that this component can serve over utilities as well.

5.2.1 Architecture

Given a local mesh topology each router was designed to connect to four adjacent routers as well as to connect to an IP core through a dedicated fifth link, as shown in Figure 20. Each link was designed to support bidirectional traffic. That is, a router can simultaneously transmit and receive on a link.

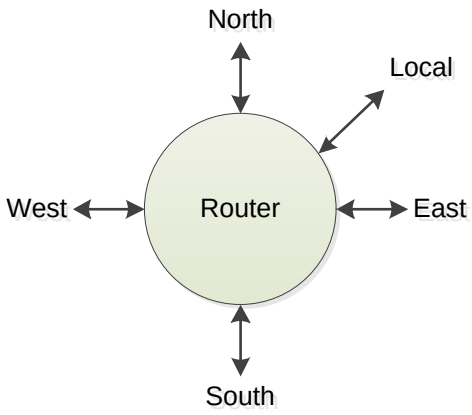

Figure 20: Router component interconnects.

The Router component is designed with five bi-directional ports. There is a total of five input channels and five output channels. Each of these output channels is capable of establishing a channel. It is the input channels' role to request an output channel. An output channel accepts the input channel's request when it is available to make a connection. From here forth, the terms *inport* and *outport* are used to describe router input channels and output channels, respectively.

Transmission channels within the Router are configured through the *switch*. The switch is responsible for routing the appropriate data and control signals between connected inports and outports. This component allows all five outport channels to have simultaneous transmissions. The block diagram in Figure 21 illustrates the interconnection between inports and outports through the switch.

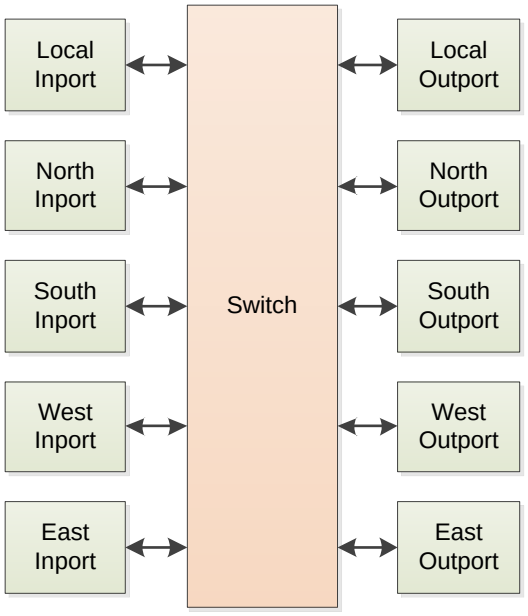

Figure 21: Routing interconnects and transmission paths.

5.2.2 Interfaces

The Router component interfaces with other Routers through the flit bus links. Two types of information are carried over these links. These are the flit busses and the stall signals. The flit busses carry the divided packet bits for transmission. The stall signals are control signals that indicate whether the receiving node is capable of receiving packet data. It should be noted that these two different kinds of signals travel in opposite directions, as shown in Figure 22. For each router port, the transmit link of one serves as the receive link of the adjacent router port.

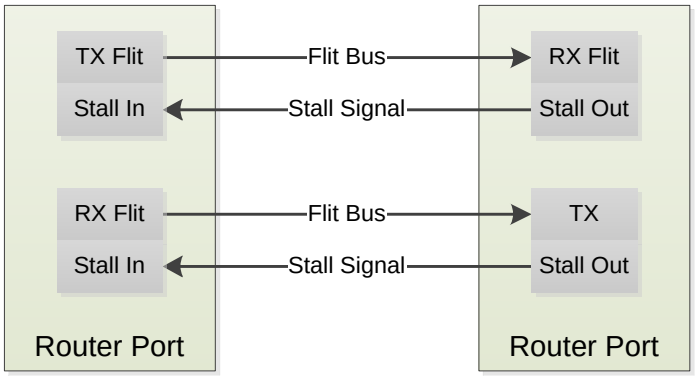

Figure 22: Links between adjacent routers.

5.2.3 Router Inport

A router *inport* is a port through which data enters the router. It consists of an input flit link, an FCB, and a target decoder. The FCB is used to maintain data integrity across the link input. The target decoder is used to determine through which outport the new packet should be routed.

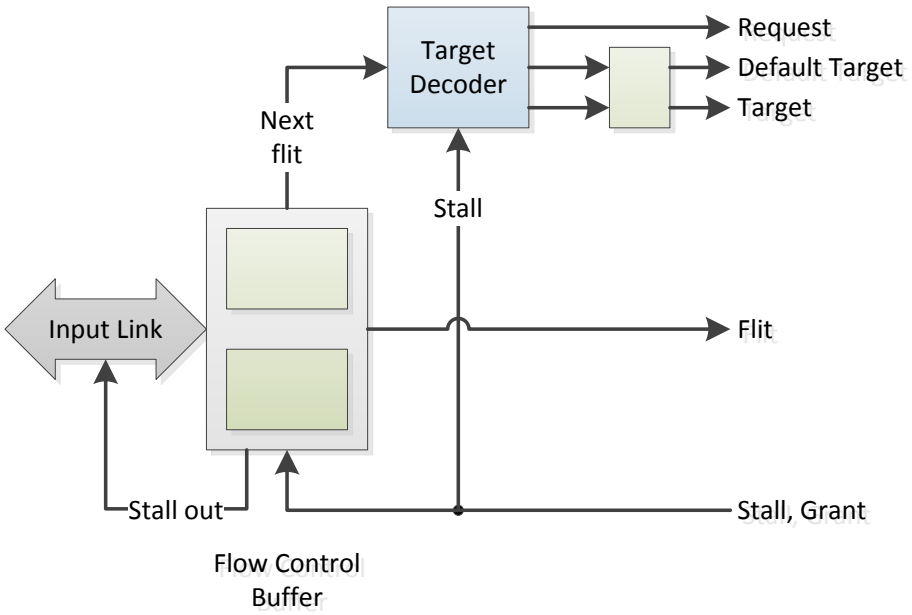

Figure 23 : Block diagram of the router inport.

The Router inport is designed to decode the target port with a single flit. This was done to prevent additional buffering needing and avoid the additional latency associated with buffering additional flits. The NextFlit output of the FCB is used to provide the routing information to the target decoder. This allows two packets to be processed back-to-back, preventing a one-cycle between requests. The target decoder produces a request vector, a target selector, and a default target selector. The request vector produces five parallel bits, each one assigned to issue a request to each of the five Router outports. The target selector is used to provide the switch with the desired outport. The default target selector is used to override the target selector. Instead of a routing an outports control signals (stalls and grants) back to the inport default signal values are routed instead. This behavior is desired when no request is made (no packet has entered the inport). Otherwise, an inport would receive control data from any outport. If that outport had an

established channel its signals would reach the inport despite having not requested it. This could result in the propagation of unnecessary stalls in upstream routers and links.

It should be noted that the Request output is a combinational output. That is, it responds immediately to a change in its inputs, such as the NextFlit value. This allows the request to be generated in the same clock cycle, allowing the requested outport to have time to grant the request on the next clock edge. The Target and DefaultTarget selector signals are registered outputs, however. They will not be updated until the next active clock edge. The registers are necessary to retain these values for the duration of the transmission. These registers are not updated until the beginning of another transaction. This is indicated by the NextFlit being either a HEAD flit or NULL flit. Additionally, the incoming Stall signal must be a GO value. This is due to the fact that the Target output provides the Switch with routing information for the outport's control information. Updating this register preemptively routes incorrect control information back to the inport. One can consider the case when the inport is stalled when the next flit is a HEAD flit. It is connected to Outport 0 as shown in Figure 24. The next flit is bound for Outport 3. If Outport 0 is driving STALL and Outport 3 is driving GO then the requesting inport will receive the GO from the next request. This will tell the inport that its stored TAIL flit has been accepted. Since Outport 0 is still stalled the TAIL flit is not accepted and data is lost. This situation is demonstrated in Figure 25.

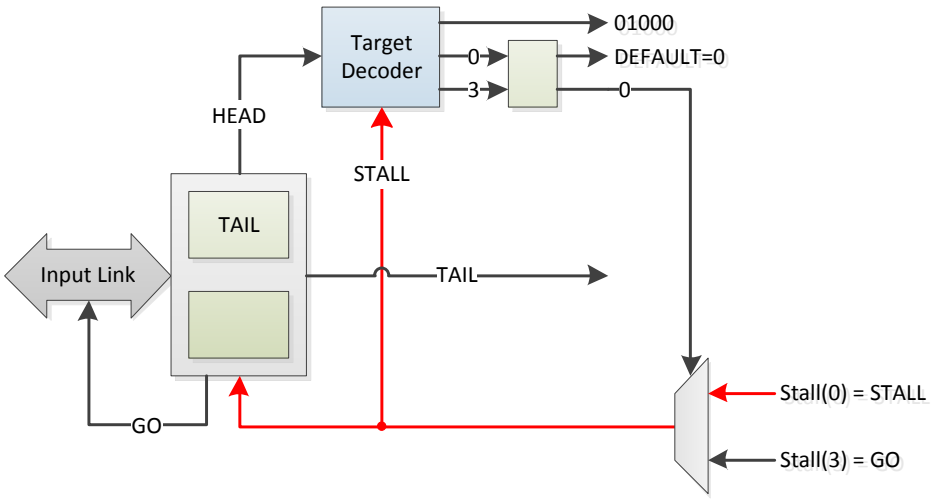

Figure 24: An example of a misuse of the target selector.

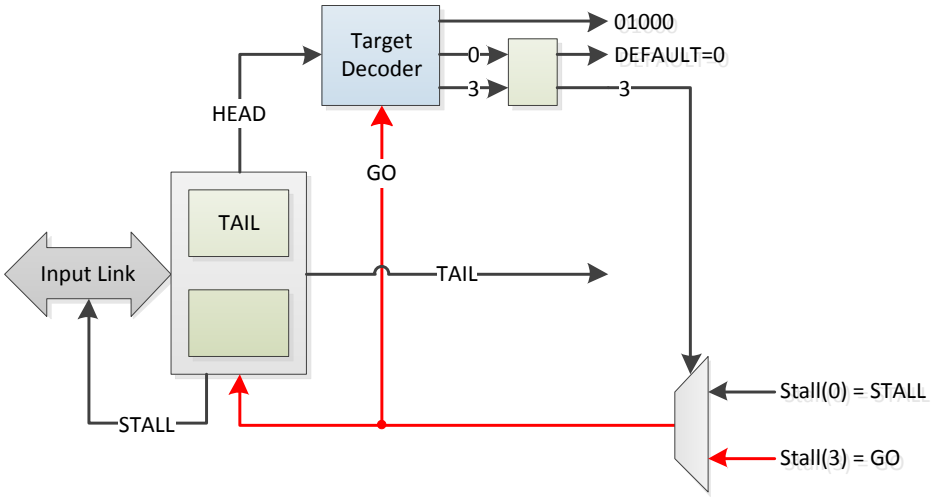

Figure 25: The result of the previous example.

5.2.4 Router Outport

The router *outport* is a port through which data exits the router. More importantly this component controls channel allocations within a router. An outport consists of two subcomponents: a round-robin arbiter and an output register, as shown in Figure 26. The output register serves as a registered output of both the outport component and the router component. Since these flit busses are used to connect to input links of other routers, the registered output reduces the total combinatorial delays, allowing for higher clock frequencies. This is common design practice in digital systems.

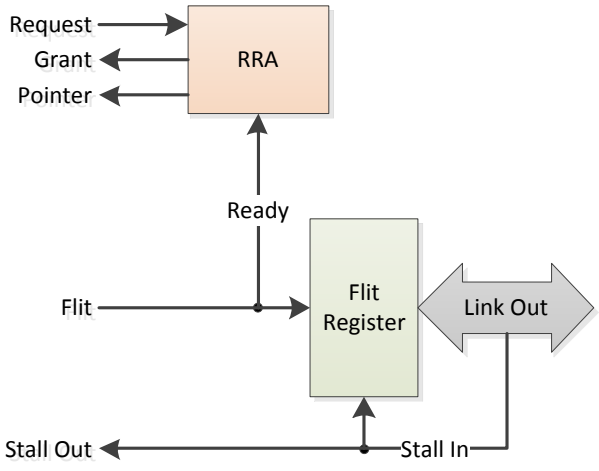

Figure 26: Block diagram of a router outport.

The round-robin arbiter component is the component responsible for channel allocation. It implements a *fair* scheduling algorithm that services each requesting device in order. More specifically, the requests of devices from the previously serviced devices are ignored in favor of the current and future devices in the order. In particular, the next device in the order requesting access will be granted [10]. A simpler implementation is a fixed-priority system where each input is assigned a priority. In this situation, if a high priority device makes frequent requests to a channel the lower priority devices can suffer *starvation*. The round-robin implementation ensures all devices will eventually be serviced.

The arbiter component receives a *request* vector and a *ready* signal and produces a *grant* vector, a *group select* signal, and a *pointer*. The request vector is generated in the Router switch. Each bit in the vector corresponds to one of the requesting devices—in the instance of the Router these are inports. The Ready signal determines when arbitration should occur. The Grant vector generated contains one bit per requesting device. These bits serve as an acknowledgement to the requesting device that it has been selected. The group select signal is used to indicate that no devices were selected. This occurs when no devices are requesting the outport channel. The Pointer signal acts as a selector in the Router switch to route data from the granted inport. The group select acts as an override in the switch to provide default signals to the outport.

The round-robin arbiter in the router outport is implemented using a mask register and a simple priority encoder, as seen in Figure 27. The mask register remembers the history of the arbiter's allocations. The register contains one bit for each requesting device (inport). If the bit is set (equal to a logic 1) then the device's request is acknowledged by the arbiter. If the bit is cleared (equal to a logic 0) the device's request is ignored. The mask register is updated after each channel allocation. The device requested and the devices preceding it in the allocation order are *masked*, or set to 0 in the mask register. An AND operation is performed between the corresponding mask bits and request bits. When a bit in the mask register is zero, the resulting bit will always be zero according to Boolean algebra. With these other devices removed from the final request vector a simple priority encoder can be used to select the next device with a request. If the final request vector has no requesting devices then the unmasked request vector (the original input vector) is used instead.

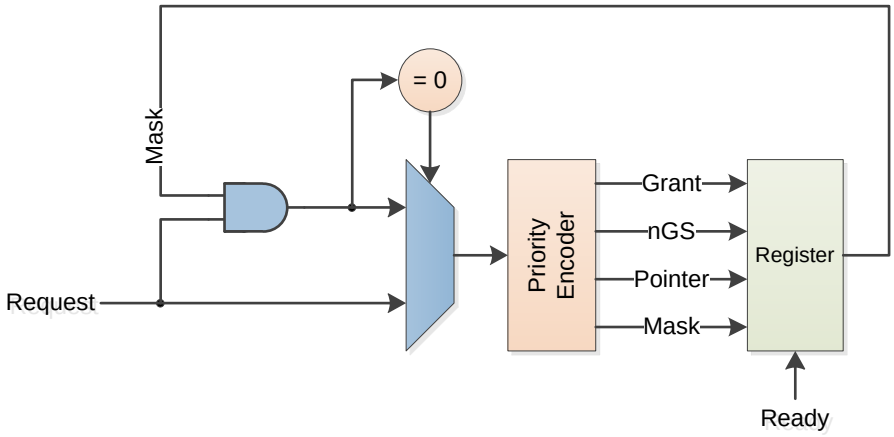

Figure 27: Block diagram of a mask-based round-robin arbiter.

The block diagram in Figure 27 shows that the outputs of the round-robin arbiter are also registered. In addition to offering better timing performance the registered outputs retain the values of the last arbitration. The Ready signal acts as clock enable to the registers. As the masking logic and priority encoder are combinational, as new requests are issued their output values will change. These changes should neither be used externally nor captured in the register. Registering the new values on the Ready signal ensures that the correct information is driven in the interacting components and that they remain stable for the duration of the transmission. The Ready signal is asserted when a TAIL or NULL flit comes in on the flit input bus and when the output link is not stalled.

5.2.5 Router Switch

The Router switch is used to produce arbitrary connection from outport to any inport. This includes switching of signals from the inport to the outport as well as signals from the outport to the inport. Between channels is maintained by the two kinds of selectors: Targets and Pointers. Target selectors switch values from the five outports back to an inport. Pointer selectors switch values from the five inports to an outport.

Flit data is multiplexed inside the switch. There a multiplexer for each outport that selects between the flit busses supplied by each of the five inports. A sixth flit bus carrying a null flit (all zeros) is included in the switch multiplexers. If the group select signal is active in the outport the null flit is output from the multiplexer instead. Figure 28 shows the multiplexer equivalent of this

circuit. Since each outport has one of these multiplexer circuits five simultaneous transmissions are possible.

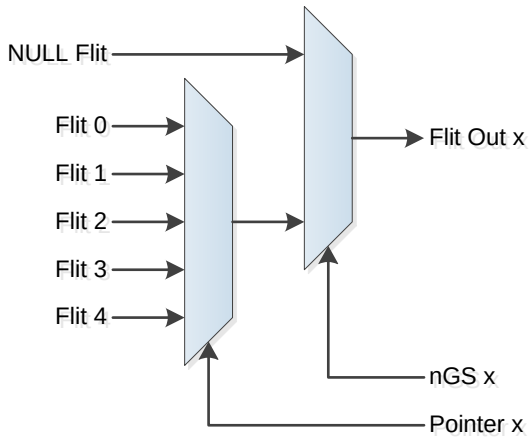

Figure 28: Flit bus multiplexer in the router switch.

Stall signals are passed from the outports back to the inports in a similar way. Each inport must have a means to read the incoming stall signals from the outport to able to suspend transmission on a downstream stall. Stall signals are routed back using the Target selector from the inport. The Default Target selector bypasses this value when active, providing the default GO signal. This selector is asserted when the inport is not transmitting. The equivalent circuit is shown in Figure 29.

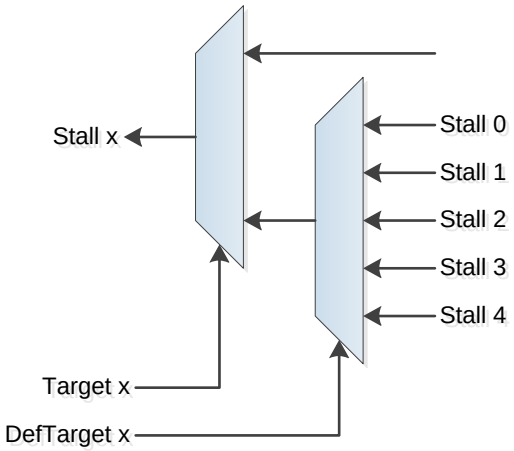

Figure 29: Circuit equivalent of a multiplexer-based stall signal switching.

The Grant signals are switched in a slightly different manner. Each outport has a Grant vector that is passed into the switch. It has one bit for each of the inports. From the perspective of the inport only the one bit for itself is relevant. Therefore, the multiplexer for an inport takes

its respective bit from each of the outport grant vectors. When the inport is not making a request or transmitting the Default Target selector select the inactive level of the grant signal (nGRANT). The equivalent circuit is shown in Figure 30.

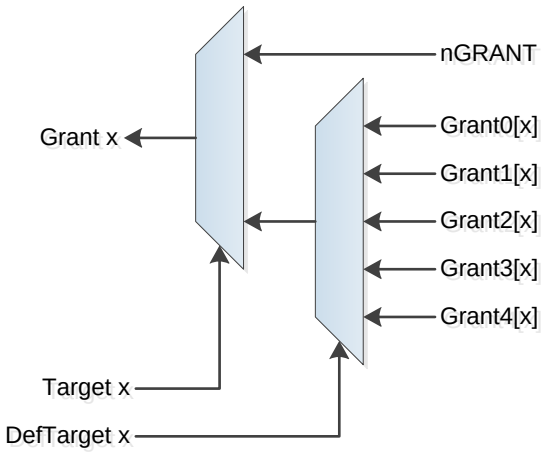

Figure 30: Circuit equivalent of the grant signal switching.

The request signal vectors are simply remapped before reaching the outports. Each inport drives five request lines: one for each outport. Across the five inports there are twenty five request lines. Only five of these lines are relevant to an outport. Those five lines of the requests lines bound for that outport. An example of such is that Outport 3 would receive the Request(3) bits from each of the inports. The position of the bit in the vector indicates to the Outport which device is requesting. For example, Bit 1 of an outport's request vector is sourced by Inport 1. Figure 31 shows the circuit equivalent for this switching block.

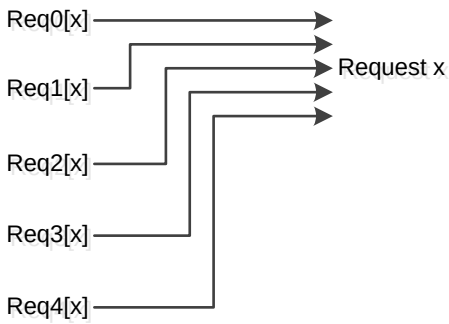

Figure 31: Outport request signal generation.

5.2.6 Separate Data Path Router

During the development of the Bridge test system, it was discovered that the Router-based test meshes suffered from a fatal flaw. The test system exposed that mesh's susceptibility to message-base deadlock, as described in Section 2.5.1.2 Message-dependent Deadlock. This flaw does not appear in the original Router test since the simulation models did not introduce the concept of request and response messages since it was out of the purview of the Router component. In order to correct this issue the data paths for requests and response had to be separate to avoid dependencies [9].

Either separate physical networks or virtual channels in the router channels are used to avoid message-based deadlock. The most common solution is the use of virtual channels [7]. Despite this fact, separate physical network were selected for the MITRE Control Plane. This is an important design decision in the project. Physical-separate data paths are chosen to prevent excessive project delays from redesigning and verifying the new virtual channel router component.

A new component is used to break this message-based deadlock. The component called the *Separate Data Path Router*, or SDP Router, was built to include two Router components. Each of the five links of a mesh router is expanded to include the signals for both the request router and the response router. This is done as an abstraction of underlying hardware. Additionally, combining the data busses significantly reduces the already burdensome number of ports in the RTL description of the SDP Router. From here further, the term *Router* describes the SDP Router and the term *Single Router* describes the individual router component.

5.3 Bridge

The Bridge component acts as a gateway between adjacent local meshes. Messages from one of the meshes enter the Bridge and are translated into the opposite mesh's topology. It acts as a target translator from one mesh to another. Additionally, Bridges can perform *global routing* where Routers can only perform *local routing*. Bridges can determine paths from one mesh to another mesh of any degree of separation. These components are designed to translate the target mesh address to the nearest bridge, which in turn routes to the next nearest bridge. This continues until the packet reaches the destination device within its local mesh.

In order for the Bridge component to work there has to be separate data paths similar the Router. While this increases the hardware cost of the Bridge component it also simplifies the target translation process. Due to the packet structure, translation is different in requests than in responses.

The structure of a bridge consists of an input shift register with flow control, a loadable output shift register, lookup tables and other decoding logic, as illustrated in Figure 32. Packet data enters the Bridge through the input link and the FCB. The input shift register must be full before translation occurs. The output shift register is loaded on the next cycle when the head of packet is in the front of the shift register. At this point, the packet fields are used to produce the target field. It should be noted that the output shift register includes an extra buffer position to capture the next flit in the packet, which is contained in the FCB. The output shift register is enabled to shift out at this time. It will stop shifting when a tail or null flit is detected at the front of the register.

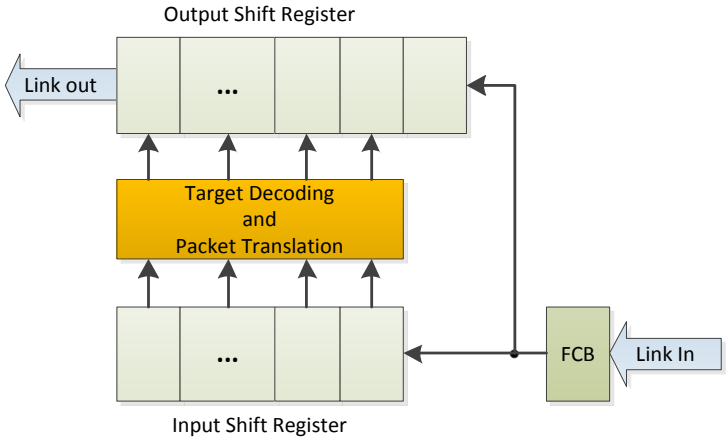

Figure 32: Block diagram of the Bridge component.

The request data path in the Bridge uses the base address field in the packet. The base address acts as a unique identifier for the device. This address is a global address—it is understood throughout the network. A lookup table is used to find the next bridge or router to which to route the current packet. The base address is translated into the XY location of this intermediary device which is concatenated with the mesh identifier of the output link's mesh to produce the new Target address. This address is loaded into the output shift register rather than the previous. This process is shown in Figure 33.

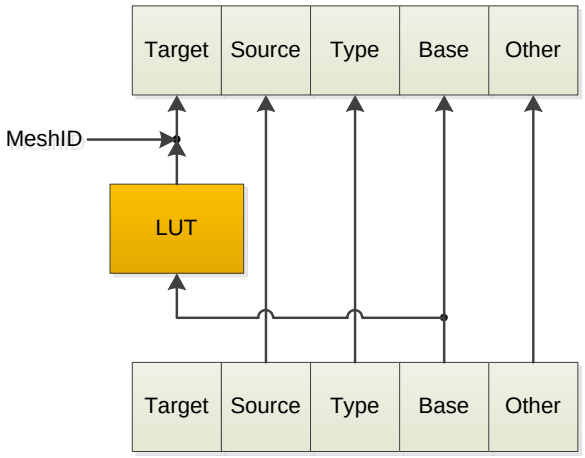

Figure 33: Request data path for the Bridge component.

The response data path of the Bridge component routes uses the Source field of the packet. This greatly reduces both the buffering requirements and the latency through the Bridge. In particular, the mesh identifier of the Source field is usually sufficient. The mesh identifier is compared with the output side's mesh address. If the Source address is in this mesh, the Source field is copied into the Target field of the output shift register. Otherwise, a software-generated[2] look-up table specific to the containing bridge is used to find a bridge that will route to this mesh. Again, the output link's mesh address is concatenated with the resulting XY location from the lookup table. This functionality is shown in Figure 34.

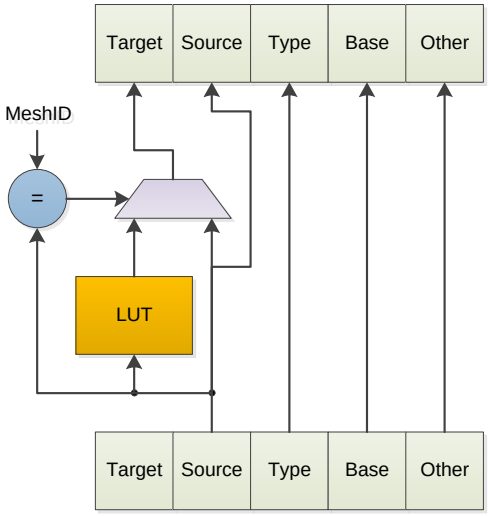

Figure 34: Response data path for the Bridge component.

[2] These look-up tables should be generated by a deployment tool to ensure consistency with the overall network. The deployment tool is planned as a long-term goal of the project.

5.4 Packet Processor

The Packet Processor (PP) is the first layer of the network interface (NI). Its primary purpose is to form a layer of abstraction over the flit-based transmission architecture. The Packet Processor interfaces between the bus protocol-specific level (which interfaces with the target IP) and the NoC through a simple shared bus. As shown in Figure 35, the Packet Processor contains an interface to a Protocol Adapter, an interface to the NoC, and two shift registers.

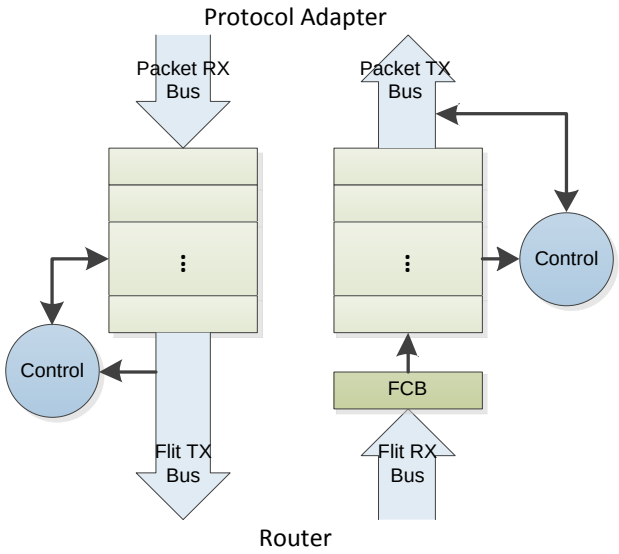

Figure 35: Block diagram for the Packet Processor.

The Packet Processor serves as a packet serializer and deserializer. That is, the receiving side of the Packet Processor accepts a *flat* packet from the Protocol Adapter. The entire packet is transmitted over this shared bus to Packet Processor block. Internally this path contains a shift register buffer. The Packet Processor loads the packet data from the bus into its flit registers. The flits are shifted into the connected Router's Local port.

The transmitting side of the Packet Processor contains another shift register to collect the packets coming in from the network. Once the front of the packet reaches the front of the shift register,[3] a request is generated on the transmit bus to the protocol adapter. The shift register is preceded by a Flow Control Buffer. This is necessary since the Packet Processor needs to be able to capture a flit inflight when the stall signal is asserted.

[3] The request generation uses sequential logic, so the flit register just before the front is checked. The request signal will be generated at the same time that the front of the packet reaches the front of the shift register. This prevents a dead cycle between the arrival of the packet and the request generation.

Despite the simplicity of this block, considerations are made in regards to flow control. When the Packet Processor transmits an entire packet up to the next level, it must be sure that there is buffer space available. The flow control is handled by the bus protocol between the two layers. The bus carries three signals: the data bus, a request (REQ) line, and an acknowledge (ACK) line. When the Packet Processor has an entire packet to transmit, it asserts the REQ line. It must wait for the ACK line to be asserted by the Protocol Adapter. It may take many cycles for the ACK to be sent so the data must be retained in the shift registers. Therefore, a stall is introduced to incoming flit link. This stall is only generated when a request is active and the ACK is not asserted. This prevents the NoC from transmitting a packet before the Packet Processor has room to store it. Additionally, the shift register only shifts when a request is not active. Once a request is generated, the register is suspended. The flow chart in Figure 36 describes the control flow of the transmission process.

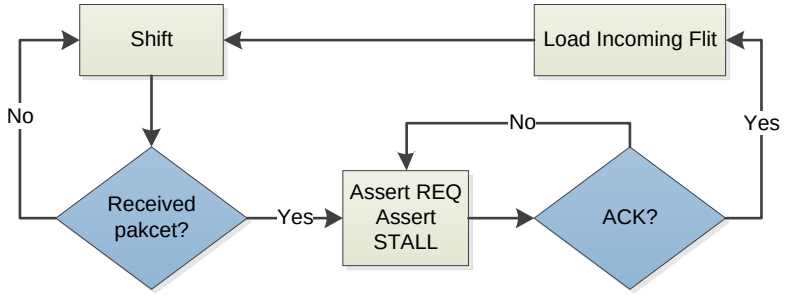

Figure 36: Flow diagram of the Packet Processor transmit bus request cycle.

The Packet Processor's receiving bus functions in a similar manner. This control flow is documented in Figure 37. Data is passed from the Protocol Adapter to the Packet Processor through another shared bus. When the Protocol Adapter sends data it similarly asserts its REQ line and waits for an ACK from the Packet Processor. The Packet Processor will accept the Protocol Adapter's request when the output shift register (to the NoC) is empty. This prevents data loss in the inflight packet. When the request is accepted, the output shift register is loaded with the incoming packet.

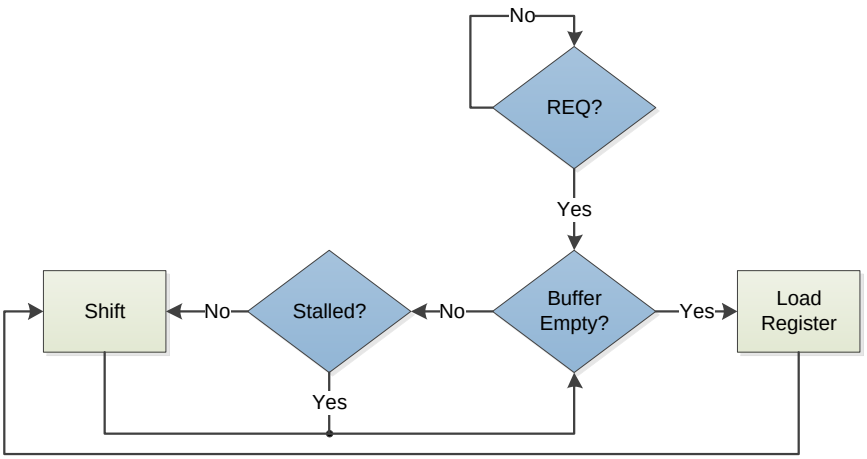

Figure 37: Flow diagram of the Packet Processor receive bus request cycle.

5.5 OCP Protocol Adapters

The Protocol Adapter (PA) layer completes the Control Plane's Network Interface level. The PA components connect the routers of the interconnect network to the IP cores. Each Protocol Adapter implements a certain bus protocol, such as OCP or AHB. This side of the Protocol Adapter connects to the IP core. The Protocol Adapters connect to routers through the transmit and receive busses of Packet Processor layer. These connections are shown in Figure 38.

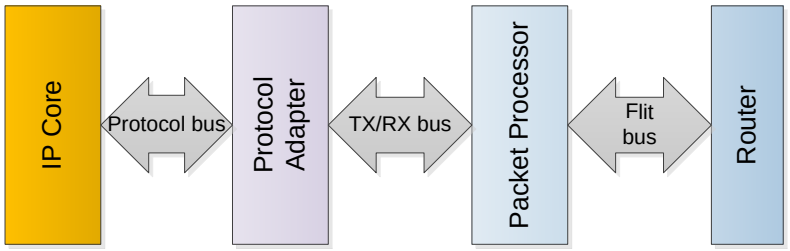

Figure 38: Layered structure of the Network Interface.

Protocol Adapters are divided into two general categories of *endpoint adapters* and *initiator adapters*. Endpoint adapters connect to endpoint IP cores, or slave devices on the network. The endpoint adapters act locally as *master* devices to the connected IP core. Requests coming in from the network are translated into the implemented bus protocol's request type, as shown in Figure 39.

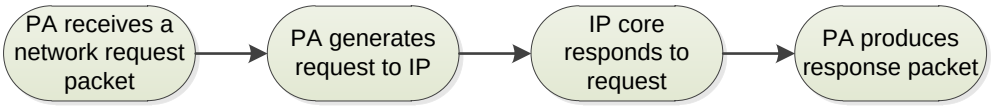

Figure 39: Control flow of request generation through endpoint Protocol Adapters.

Initiator adapters connect to initiator IP cores, or master devices on the network. These adapters are locally *slaves* of their connected IP cores. The IP cores issues a request to these adapters and the adapters generate a corresponding packet to transmit to the network, as illustrated in Figure 40.

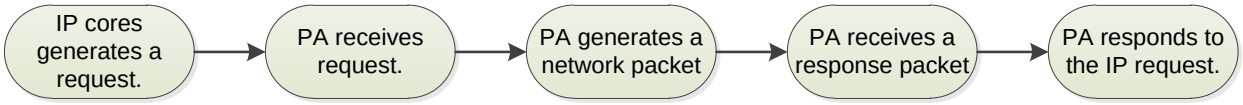

Figure 40: Control flow of request generation though initiator Protocol Adapters.

It is important to note that the Protocol Adapter is actually responsible for defining two distinct protocols. The first protocol defined is that of the bus interface it implements. This protocol only dictates the transaction with the IP core. The second protocol is a packet protocol that determines how this information is transmitted across the network to target devices. It is likely that in particular system that all of the packet protocols will be compatible with one another.

5.5.1 Packet Protocol

The MITRE Control Plane is currently built for a single overlying protocol. It supports basic read, write, and no-operation (NOP) operations from initiator devices to endpoint devices. The packet structure is extended to include a local address field, an operation field (OP), a parameterized data field, and an error field, as shown in Figure 41. All requests generate responses in protocol. The *Error* field allows the initiator to know if its request was successfully serviced.

Figure 41: Packet structure for the MITRE OCP Memory protocol adapters.

This packet protocol supports three basic operations: Read, Write, and No Operation, or NOP. The read and write operations can be directly related to the OCP read and write operations. The Write operation issues an address and data to store at the given address. The Read operation

requests the data at the given address. The encodings of the supported operations are found in Table 2.

Operation Name	Encoding	Function
OP_NOP	00	Performs no operation. The endpoint device produces a response packet with ERR_NONE.
OP_WRITE	01	Performs an OCP write to the target device.
OP_READ	10	Performs an OCP read from the target device. Slave data is returned.

Table 2: Supported operations through the OCP Memory Protocol Adapter.

An error field is included in the packet protocol to satisfy the sponsor's request for descriptive error handling. This field gives the initiator information about the success of its request. This is particularly important for read operations, where a failure can result in invalid data returned. It is important that initiator receives this information so it may retry the operation to receive valid data. With the limit error reporting the OCP protocol, the error reports in this protocol are also limited. When an OCP request is made, the PA can only report error when an error response is given (a response of SRESP_ERROR or SRESP_FAIL) [11]. However, no other information is given regarding the error [12]. The more descriptive errors of this protocol originate from the endpoint itself. These include timeout and misroute errors, as shown in Table 3.

Name	Encoding	Function
ERR_NONE	000	No error occurred; the operation was successful.
ERR_FAIL	001	The OCP IP responded with ERROR or FAIL.
ERR_TIMEOUT	010	The maximum duration of the OCP transfer elapsed the transfer was terminated.
ERR_INVAL_OP	011	An unsupported operation or invalid operation was issued.
ERR_INVAL_TAR	100	The request packet was misrouted to this endpoint.

Table 3: Error codes supported in the OCP Memory Protocol Adapter.

5.5.2 Implementation

Only an OCP-based endpoint Protocol Adapter is built and tested due to time constraints of the project. It was designed with a state machine to control the flow data and translate the packet-based requests into OCP requests to the target IP core, as shown in Figure 42.

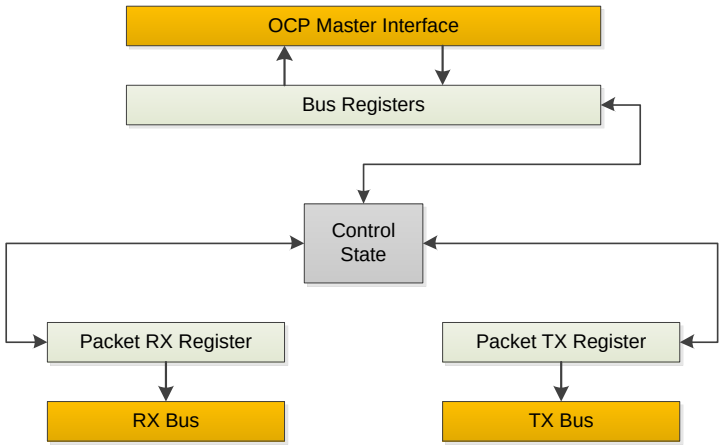

Figure 42: State-based packet request to OCP request translation in the Protocol Adapter.

This Endpoint PA initially waits for an incoming packet from the attached Packet Processor. It acknowledges the request upon receiving it and will buffer the incoming packet. The packet is checked to determine if an OCP request is necessary for this packet. NOP packets do not need to make an OCP request. In fact, NOP requests cannot be made on an OCP bus, as no such OCP command exists on the OCP Memory profile [12]. Bad packets also do not generate OCP requests. These include misrouted packets and invalid operations, as not all OCP commands are supported on this protocol. For the packets that do not generate OCP requests, a response packet is immediately generated and transmitted through the Packet Processor. Packets that do initiate OCP requests to the IP must obey the OCP Memory protocol. A request is put on the bus until accepted by the slave (IP). The command-accept terminates a write operation. Read operations are not terminated until the slave gives a valid response [12]. Both of these phases are capable of *timing out*, or reaching a maximum elapsed time for the transaction. Upon timeout, the Protocol Adapter applies a reset to the slave IP core to correct a possible error that is stalling the slave device. This prevents the endpoint from being permanently stalled and potentially causing deadlock on the network. Its functionality is described by the flowchart in Figure 43.

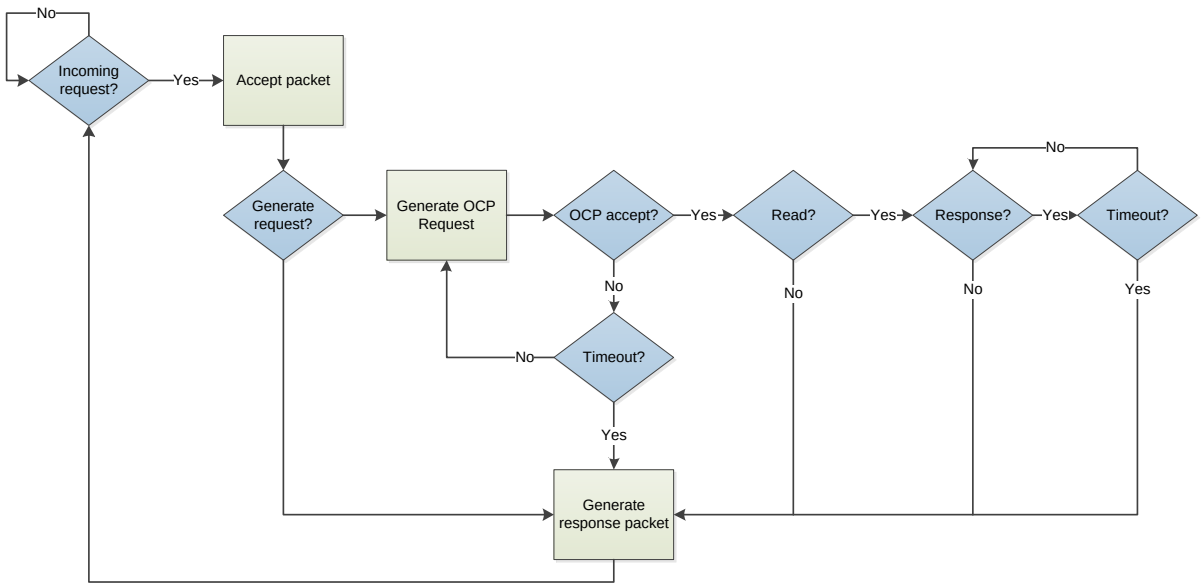

Figure 43: Flow chart of the OCP Memory endpoint protocol adapter state machine.

6 Testing and Verification

Verification is a fundamental component of chip design. Since chip fabrication is a long and expensive process, it is very important to thoroughly test RTL designs in simulations. The intent is to greatly reduce the possibility of unforeseen errors by exhaustively testing the design under as many different conditions as possible.

6.1 Testing Methods

Each of the Control Plane test systems, or *testbenches*, followed a similar style. Each test system consisted of a *device under test*, or *DUT*, at least one *bus functional model* (BFM), and a *monitor*, as shown in Figure 44. The component being tested is called the device under test, or DUT. The DUT is connected to one or more *bus functional models*, or BFMs. The purpose of a BFM is to generate stimulus for the DUT. A BFM is a simulation model used to model devices that would interface with the DUT. Their designs are greatly simplified by not having to be synthesizable. Additionally, they are only responsible for implementing a compatible interface to the DUT component [13]. BFM components were designed to the produce large amounts of randomized data. Randomization helps identify unforeseen scenarios that may expose flaws in the hardware design.

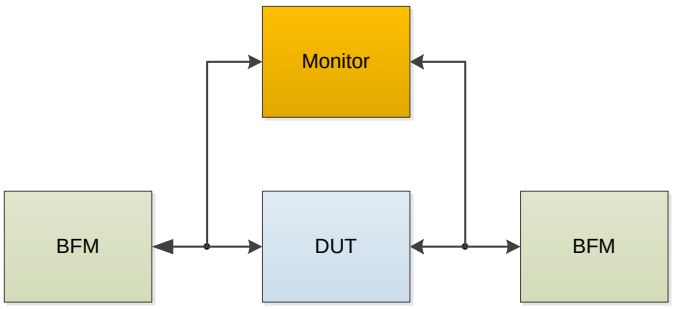

Figure 44: General structure of a testbench.

Monitor components were used determine the success or failure of the simulation. In the earlier test systems, namely the router subcomponents and the FCB component, the monitor was used to read the stimulus to the DUT from the BFMs and the outputs of the DUT. Consistency checks were used to verify that the DUT was responding correctly to the BFM's for these lower-level designs. For higher-level designs the monitor had a much more passive role. In these types of tests the DUT was composed of many interconnected network devices simulating a small

NoC. The BFMs were used to emulate real endpoints and initiators. At this level of testing the monitor was not used for signal (bit) level testing and checking. Instead, BFMs issued messages to the monitor via a *mailbox*, a common thread synchronization tool in software. These messages were representative of the actual requests and responses being issued on the NoC interconnects between the BFM devices. The monitor uses these status messages to determine whether a message has been delivered successfully. Additionally, tests using test networks included an inactivity timer to determine detect deadlock conditions. For each clock cycle that the monitor does not receive a status message it increments the inactivity timer. If a status message is received on that cycle the timer is reset to zero. This is similar in behavior to a "watchdog timer" on microprocessors.

A desirable feature of such testbenches is their *self-checking* capabilities. RTL simulators often produce output waveforms from the design simulation. It would be tedious and error-prone to check the success and failure of the each transaction. This task is delegated to testing components such as BFMs and monitors. This is especially useful for running long tests includes potentially thousands to millions of iterations.

6.1.1 Queues

The Control Plane test systems utilized some of the higher-level simulation constructs available in the SystemVerilog language. Many of these constructs offer quick solutions to problems that would otherwise require additional design considerations. One such example is the SystemVerilog *queue*. A queue is a dynamic array of either a specified or unspecified (unbounded) size. Elements can be added and removed from a queue at any time, as it is dynamic. Methods are provided in this type of queue to add and remove elements of the queue in a particular order, such as removing from the front and adding to the back [13]. SystemVerilog queues behave like hardware FIFO (First-in, First-out) queues [14]. Consequently, they serve as a quick substitute in simulation models. The Control Plane tests utilize queues in a similar capacity.

6.1.2 Mailbox

A fundamental element of the Control Plane verification systems is the *mailbox*. This SystemVerilog construct allows communication between different threads. A mailbox is a FIFO

queue of messages that is capable of synchronizing accesses from multiple threads, ensuring data integrity [15]. BFMs in the Control Plane test systems use mailboxes to communicate system events such as sending packets, receiving packets, and errors. This allows the BFMs to generate random stimulus independent of the rest of the testbench but also to allow the system to predict the outcomes of its actions. Special event messages are transmitted to the mailbox in the monitor from BFMs. The messages used in the mailbox can have any data type [13]. Since SystemVerilog includes object-oriented extensions to Verilog, mailboxes can use classes that contain system event information as the message data type. The monitor uses the mailbox to predict and confirm the results high-level transactions in the system.

6.1.3 Interfaces

An *interface* is a SystemVerilog construct that bundles many signals into a single entity. It is similar to a VHDL record and a structure and computer programming. Interfaces are used to ease connecting devices together by grouping related signals into a single port of a module. Unlike a VHDL record, interfaces can contain both inputs and outputs. The directions of these signals can differ between different modules using *modports* [15]. For instance, a bus interface could have master and slave modport where the address bus is an output in the master modport and an input in the slave modport. Interfaces can also contain tasks and functions just like modules [15].

6.2 Mesh Subsystem Tests

The mesh subsystems tests were incorporated at all levels of the Control Plane design. These served as a practical test environment for each component. Ultimately, these tests were important for ensuring that the newest level of RTL would interact correctly with the previous level of RTL. As each level was added, new BFMs were adapted from previous ones to implement the interface of the next level component.

6.2.1 General Structure

Each mesh subsystem test consists of a mesh subsystem (the DUT), several initiator and endpoint BFMs, and a monitor. The block diagram in Figure 45 illustrates this configuration.

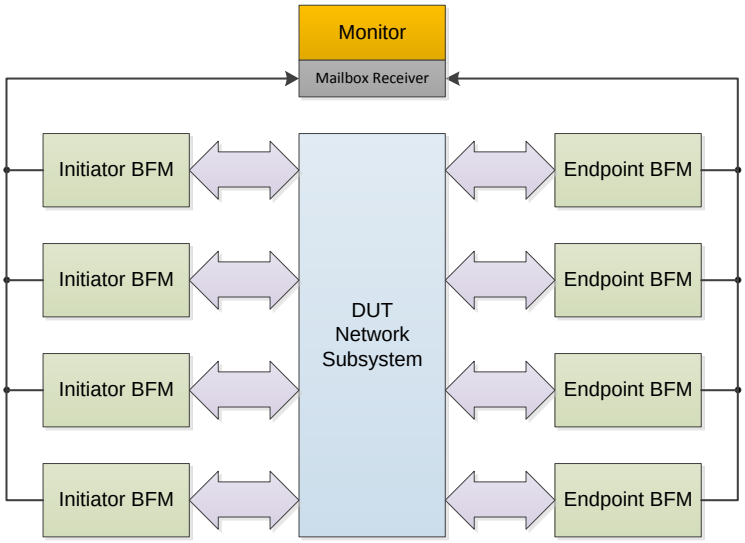

Figure 45: General structure of a mesh subsystem test.

The structure of the testbench simplifies the role of the monitor. The monitor is not responsible for implementing any particular interface to communicate with the BFMs and the DUT components. Instead, the BFMs act as both transmitters and receivers (like the IP cores would). The BFMs use a shared mailbox interface (as shown in Figure 45) to communicate transaction events to the monitor. Such events include transmitting and receiving requests, transmitting and receiving response, and error packets. In the case of the BFMs for the tests prior to the Protocol Adapter layer, each response is generated at the packet level. Since the monitor does not have to conform to a new interface from one test to the next the same monitor was each for each of those tests.

The mailbox interface uses the SystemVerilog `interface` construct. The interface defines a mailbox with a parameterized message type. Two modports are defined: `send` for BFMs and `receive` for the monitor. Each modport implements the appropriate functions and tasks to read from and post to the mailbox [14]. This implementation circumvents the restriction in the SystemVerilog language that prohibits the use of mailboxes to as ports into a module or program[4]. Interfaces, however, can be used as ports [15].

[4] The term program refers to the SystemVerilog construct called `program`.

6.2.2 Packet-based Tests

Before reaching the IP core level interface the subsystem test BFMs generated network stimulus as packets. Packets are the message type processed in the components below the Protocol Adapter level. Consequently, packets are generated at the BFM simulation model level in these tests.

At these levels the Control Plane is protocol-agnostic. That is, the packet and message structure defined at this level is used for routing and transport. For example, the Routers and Bridges do not process the application-specific bits that the Protocol Adapters would use, as shown in Figure 46. These bits were randomly generated for the purpose of transmitting more unique packets. For this fact, it was simpler for the endpoint BFMs to *echo* the request to the sender, or send back the same data packet. Using this scheme the initiator BFM can predict the exact the response packet it should expect to receive from the target endpoint device BFM.

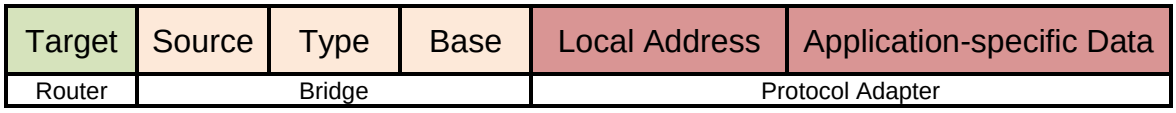

Figure 46: Packet fields according to network level and protocol depth.

The initiator BFMs in these tests were designed to perform *non-blocking* requests. That is, the BFM did not have wait for a response to the request before issuing the next request. This was done to apply additional stress to the network DUT without requiring more BFMs and a bigger network. Each time a packet was generated the initiator BFM both reported the transmitted packet to monitor and also stored the expect response in an unbounded SystemVerilog queue. The receiver portion of the initiator BFM searches the queue for the incoming response packet. If a matching packet is found it is removed from the queue. If it is not found, it is sent to the monitor as an error. The flowchart in Figure 47 reflects this functionality.

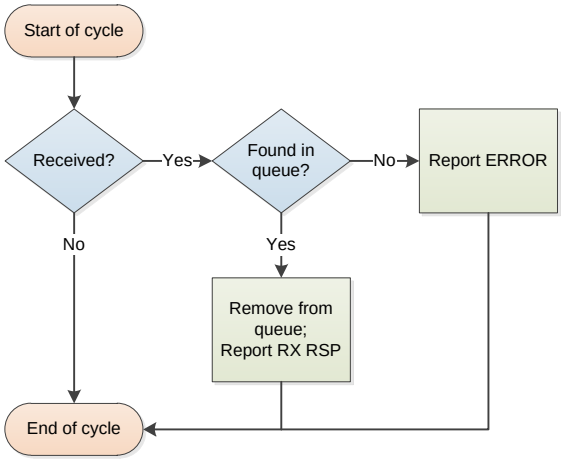

Figure 47: Mesh subsystem initiator BFM receive flowchart.

The monitor is responsible for reporting and maintaining the status of the system. This component is responsible for receiving status messages from initiator and endpoint BFMs about transmitted and received packets as well as detected errors. The monitor also maintains a count of inflight packets for each initiator. When an initiator BFM reports a transmitted packet the monitor increments the count. When the initiator BFM reports the correct response the monitor decrements that BFM's packet count. Once all initiators have report that they have finishing transmitting their packet (with a DONE message) the monitor waits for the inflight counts to be zero, indicating all packets were transmitted and received correctly. When this condition is met, an output bit is set to signal the top-level testbench module that the simulation has ended successfully, as shown in the flowcharts in Figure 48 and Figure 49. At the end of the simulation the monitor prints simulation statistics to the terminal on which it executes.

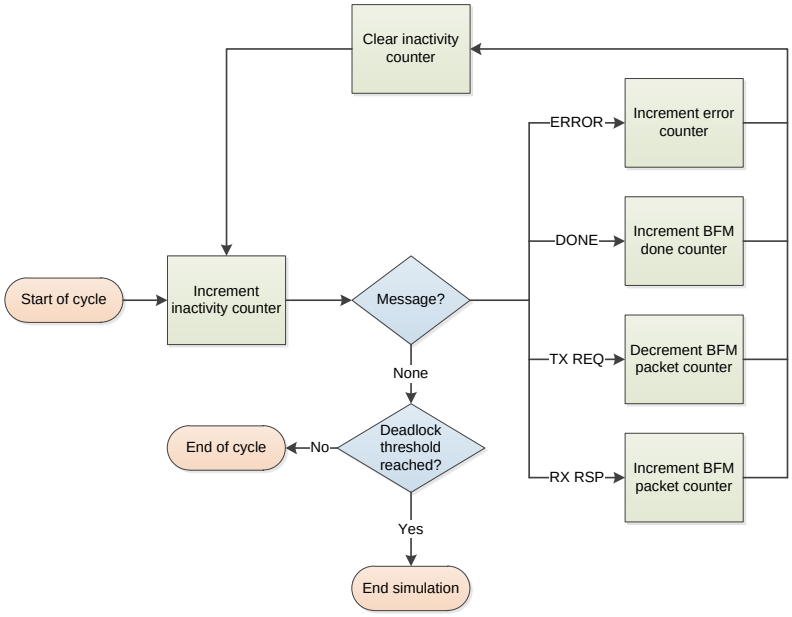

Figure 48: Flowchart of the mesh subsystem monitor's message processing system.

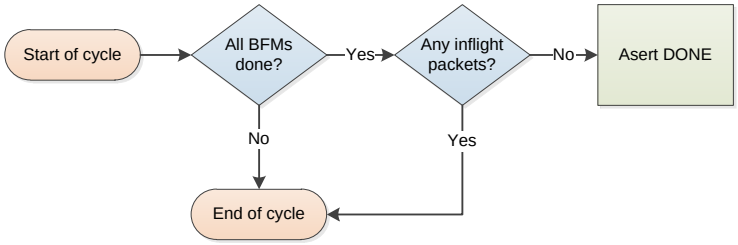

Figure 49: Flowchart of the mesh subsystem monitor's completion detection.

These testbenches also incorporate a simulation timeout. Errors in the simulation can cause the simulation to run indefinitely. The timeout imposes a maximum number of simulation clock cycles before the simulation is terminated, even without finishing or having an indication of an error. This is to prevent excessive waste of server time to run the simulation as well as prevent a scarce simulation tool license from being used by the simulation. This is implemented with two concurrent threads in the top level testbench. One thread suspends until the *Done* signal from the monitor is asserted. The other waits for the maximum number of simulation clock cycles to occur. The end of each thread ends the simulation.

6.2.3 Request-based Tests

The Protocol Adapter test introduces new level of complexity in the Control Plane test systems. The previous components like the Router, Bridge, and Packet Processor all operate on

packets in the network. The BFMs used in these tests are able to generate packets when simulation the request and response cycles of the connected IP cores. In many cases these randomized packets are serialized into and out of the network in the component's mesh test. The Protocol Adapter test introduces a new stage of the request-response cycle that includes the OCP Memory request process.

The OCP Memory BFM simulating the OCP-based IP core ultimately receives an OCP request rather than a Control Plane network packet. In the previous tests the endpoint BFMs (the IP core sink) receives a packet. The received packet is transmitted to the monitor through the mail interface to report successfully receiving the request packet. Additionally, the expected response packet is transmitted in order to verify the packet the initiator receives. Since the OCP BFM does not receive a packet, it only receives certain information about the transfer, such as the operation and data (for writes). Other packet information such as the source (initiator identifier) and target (endpoint identifier) is not transmitted to the BFM. Another particular issue is when the request packet does not generate an OCP request, such as packet including an invalid operation or a NOP. The OCP BFM does not receive a request from such packets. Consequently, they cannot report to the monitor the appropriate response packet.

Modifications to the mesh subsystem test architecture allow this structure to work for such tests. These include modifications to the use of the monitor as well as to the BFMs. The OCP BFM reports the results to all OCP transactions the monitor. This BFM is adapted from an existing MITRE verification BFM for the CRB IP. It randomly chooses how to respond to requests. It can choose to respond to the request successfully or with an error. Additionally, it randomly decides whether to timeout the transfer. The OCP BFM reports the expected error field to the monitor upon completion of the transfer. The monitor stores this error field in a location associated with the sender BFM.

Two issues exist with this scheme: the OCP BFM is not aware of its sender's identity and request packets that do not form OCP requests are ignored. The first of these issues is addressed by the modifying the initiator BFM. The initiator sends its BFM identifier, a parameter used to identify the BFM in the monitor, in the Local Address field of the packet. This address is sent to the OCP IP in all transactions. This guarantees that the OCP BFM can identify its sender. The initiator BFM addresses the second issue by storing parameters such as the Operation field and

Target field. The initiator checks the operation generated for the OCP BFM for operations that would not generate an OCP request. For these operations a special message is sent to the monitor that indicates that the value stored in the associated Error field in the monitor is not valid for this request. The initiator instead uses this information to predict the correct endpoint response value, such as `ERR_INVAL_OP` for invalid operations and `ERR_NONE` for NOP operations. The BFM transmits with this special message a Boolean value indicating success or failure to the monitor. The regular response-receive message is sent to the monitor for all other operations, as the expected response is transmitted to the monitor by the OCP BFM.

7 Conclusion

This project set out to develop a scalable and reusable control plane NoC architecture for use by the MITRE Corporation's VLSI design group. An emphasis was placed on the ability to support a large number of devices to meet the growing demands of current embedded digital signal processing chips.

7.1 Summary of Project Contributions

The work from this project provided the project sponsor with a developed network-on-chip architecture for their Control Plane design. This architecture dictated the parameters of the network such as routing strategies and switching techniques. These parameters were selected mainly in accordance to design complexity, scalability to large numbers of devices, and chip area. The architecture was also designed to avoid network vulnerabilities such as deadlock. This was addressed at the architectural levels to avoid the need to address these concerns in higher-level protocols.

The network elements were developed from the established network architecture. Each component was built according to strict specifications determined by the network architecture. These components were designed with a bottom-up approach. That is, the lowest levels of the architecture were developed first. These layers included the link-level flow control buffers and the router component. The design of low-level components established a specification of next level.

The network interface level of the NoC was developed to provide a standard interface to IP cores and external devices. This was an important step to maintaining compatibility with the connected devices. This prevented hardware designers from having to develop custom hardware to interface with Control Plane's network. Ultimately, it preserved the reusability of the IP cores, as they could use standard bus interfaces that could work in other systems. The reusability of the network interface was maintained by dividing it into two layers. The lower layer of the network interface, the Packet Processor, retained the link-level access to the network. The low-level functionality of the Packet Processor was abstracted through a pair of point-to-point busses for communication with the next layer. The Protocol Adapter, the higher level of the network interface, implemented the target bus protocol for the IP cores to communicate. The details of the

NoC were mostly obscured in this layer. It was in this level that the higher level protocol, such as request operations and data transfer, were implemented.

These details of the network were unknown to the lower levels of the network to achieve simpler hardware as well as to maintain reusability. The routing elements were designed agnostic to these protocols and served as only a transportation mechanism for network messages. By decoupling specific protocols from the underlying communication interconnect (the system of routing elements) the Control Plane was not limited to any particular message structure or protocol. The underlying network components were generic to any network-on-chip based application. These protocol-specific designs were encapsulated in the highest level the network design so that new protocols could easily be introduced into the design without requiring the redesign of the underlying network components.

Each of the network components were thoroughly tested using RTL simulators. Realistic test systems were developed that simulated a network structure connected to simulation models acting as the transacting IP cores. Exhaustive tests were performed to verify the integrity such test systems. In particular, these test systems assured that all messages were delivered successfully. Deadlock detection was also built into these tests to ensure that deadlock-freedom was maintained by each successive level. These tests were performed to avoid the need to test the network components in the future designs in which they will be deployed. One of the motivations for the sponsorship of this project was to create a reusable tool that did not require significant redesign and additional verification.

7.2 Future Work and Improvements

Development of the MITRE Control Plane is planned to continue after the conclusion of this project. With the majority of the underlying network architecture a functional collection of IP is now available. Using the specifications of the components, in particular the Packet Processor, additional Protocol Adapters can be built to conform the Control Plane's architecture and provide support to additional bus protocols that have not been developed during the course of this project. The existing OCP Memory Protocol Adapter serves as a reference to interfacing with the Packet Processor's busses to the network as well as the state-based approach to request translation to the IP core protocol level.

The project sponsor also seeks the development of a deployment system for the Control Plane. Deployment can be quite cumbersome for large NoCs. The deployment tool would likely be a software tool or a set of scripts used to generate RTL for the system based a set of given parameters, such as link bandwidth, number of mesh layers, IP cores, and their bus interfaces. Such a tool would instantiate the network of routers, bridge, and respective interfaces to connect the appropriate devices. This would greatly reduce development time associated with debugging missing interconnects or incorrect routes between routers. The deployment tool should also generate test code for verifying the resulting RTL. Just as a human can make errors in deployment a computer program can as well. Verification of this generated RTL is also important for reducing the potential errors for the system before fabrication.

A user guide is planned for after the deployment tool is completed. The sponsor wants the user guide to give detailed instructions for using the Control Plane components and deploying a network into a design. It is planned for the user guide to include both manual deployment as well as automated deployment. This document should describe the parameters and interfaces used in each component. The final section of the document should also describe the process of using the software deployment tool.

A proposed feature of the Control Plane was to implement more compact address spaces for target devices. With the current implementation, the request address from an IP core includes the base address and the local address in the packet structure. As the number of endpoint devices increases, more base address bits are needed to address them. When considering microprocessors that only have a limited number of address lines (e.g. 32-bit) this greatly restricts the address space of target devices. For instance, in a system with 100 endpoint devices, a base address of seven bits is needed. If one endpoint device contains a 2GB (2^{31} bytes) address space, thirty-one address lines are required. This could exceed the physical address space limit of an initiator device. Future protocol adapters will likely rectify this problem by including address decoders that resolve initiator IP cores' request address into the base and local address format used by the Control Plane. This solution reduces such initiators' restrictions from such an address space while allowing the Control Plane to operate on a simpler address format to simplify routing and request logic. Also, this localizes the additional chip area penalty from the address decoder to the initiator Protocol Adapters and not repeated in the bridge and endpoint components.

References

[1] A. A. Jerraya and W. Wolf, Multiprocessor Systems-on-Chips, San Francisco, CA: Morgan Kaufmann, 2005.

[2] S. Pasricha and N. Dutt, On-Chip Communication Architectures: System on Chip Interconnect, Burlington, MA: Morgan Kaufmann, 2008.

[3] W. J. Dally and B. Towles, Principles and Practices of Interconnection Networks, San Francisco, CA: Morgan Kaufmann, 2004.

[4] I. Cidon and I. Keidar, "Zooming in on Network-on-Chip Architectures".

[5] B. Grot and S. W. Keckler, "Scalable On-Chip Interconnect Topogies," in *2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2008.

[6] V. Rantala, T. Lehtonen and J. Plosila, "Network on Chip Routing Algorithms," Turku, Finland, 2006.

[7] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, G. De Micheli and L. Raffo, "Designing Message-Dependent Deadlock Free Networks on Chips for Application-Specific Systems on Chips," 2006.

[8] R. Holsmark, "Deadlock Free Routing in Mesh Networks on Chip with Regions," Linkoping, Sweden, 2009.

[9] A. Hansson, K. Goossens and A. Radulescu, "Avoiding Message-Dependent Deadlock in Network-Based Systems on Chip," Hindawi Publishing Corporation, 2006.

[10] M. Weber, "Arbiters: Design Ideas and Coding Styles," Boston, 2001.

[11] OCP Internation Partnership, "Open Core Protocol Specification".

[12] E. C. Whitney, "Applied Common Interfacing Techniques using OCP," in *SNUG*, Boston, 2009.

[13] G. Tumbush and C. Spear, SystemVerilog for Verification, Springer, 2012.

[14] S. Sutherland, "Modeling FIFO Communication Channels Using SystemVerilog Interfaces," in *SNUG*, Boston, 2004.

[15] S. Sutherland, S. Davidmann and P. Flake, SystemVerilog for Design: A Guide to Using SystemVerilog for Hardware Design and Modeling, New York: Spring Science+Businessm Media, LLC, 2006.

Appendix A Terms and Abbreviations

A.1 Field Terminology

ACK	Acknowledge
AHB	Advanced High-Performance Bus
AMBA	Advanced Microcontroller Bus Architecture
BFM	Bus Functional Model
DSP	Digital Signal Processing
DUT	Device Under Test
FIFO	First-in, first-out
Flit	Flow Control Unit
Head	First flit in a packet
IP	Intellectual Property core
LUT	Look-up Table
NI	Network Interface
NoC	Network-on-chip
NOP	No-operation – an operation code that indicates that no action is taken.
OCP	OpenCore Protocol
RTL	Register Transfer Language
SAF	Store-and-forward switching
SDP	Separate Data Path
SoC	System-on-a-chip
SPI	Serial Peripheral Interface
Tail	Last Flit in a packet
VCT	Virtual cut-through switching
VLSI	Very-large Scale Integration
WH	Wormhole switching

A.2 MITRE Control Plane Terms

CP	Control Plane
CRB	Control Ring Bus
FCB	Flow Control Buffer
Inport	A portion of the router that receives packet data.
Null flit	A flit carrying no message-related data. Its header contains the decimal value 0. These are transmitted on idle links.
Outport	A portion of the router that transmits packet data.
PA	Protocol Adapter
Pointer	Switch selector for routing signals from an inport to an outport.
PP	Packet Processor
SDP	Separate Data Path
Target	Switch selector for routing signals from an outport to an inport.