Design and Implementation of a Wormhole Router Supporting Multicast for Networks on Chip

B E I Y I N
Design and Implementation of a Wormhole Router Supporting Multicast for Networks on Chip

Master of Science Thesis
In Electronic System Design

by

Bei Yin

Stockholm, May 2005
Abstract

Packet switched networks are being proposed as a global communication architecture for future System-on-Chip (SoC) designs. Buses (a single bus, segmented buses or a hierarchy of buses) do not scale with the system size in bandwidth and clocking frequency. However, a bus is very efficient in broadcasting. A network allows many more concurrent transactions, but it does not directly support multicast. As the system size scales up to exploit the chip capacity, multicasting needs to be efficiently supported.

The objective of this thesis project is to design and implement a wormhole router supporting multicast for Network-on-Chip (NoC). Wormhole routing is a network flow control mechanism which decomposes a packet into smaller flits and delivers the flits in a pipelined fashion. It has good performance and small buffering requirements. In order to provide performance predictability, a multicast protocol is designed. It consists of establishment, transmission and release phases. A multicast group can choose to reserve virtual channels during establishment and has priority on link bandwidth arbitration.

The implementations are at the RT level using VHDL, and they are synthesizable. First, based on a canonical wormhole router model, a unicast wormhole router is implemented and validated. Then, based on the unicast wormhole router design, the multicast protocol is implemented. The resulting router supports both unicast and multicast traffic.

We test the performance of multicasting by constructing a linear array network and a mesh network. In both networks, unicast traffic and multicast traffic are injected. Our results show that the multicast scheme is efficient and does not affect the unicast performance significantly if the network is not saturated. The implementation overhead for multicast is high. In addition, the control path is the performance bottleneck and should be optimized.
Acknowledgments

This Master Thesis project has been accomplished at the Laboratory of Electronics and Computer Systems (LECS) in Department of Microelectronics and Information Technology (IMIT) of Royal Institute of Technology (KTH) in Stockholm, Sweden. Many people have contributed in one way or another to the development of this thesis.

I am very grateful to my supervisor Zhonghai Lu's instructions and supports in all stages of the thesis work. He gave me the opportunity to do this challenging and interesting project. I would like to thank Prof. Axel Jantsch and Doc. Johnny Öberg for their constant guidance. They are very nice people. I would like to say thanks to my parents and Haige for always believing in me, supporting me and giving me the chance to realize what I want. Finally, many thanks to all my friends in Stockholm. We shared a lot of great moments that I will never forget.
## Contents

1 Background .......................................................... 1  
  1.1 Network-on-Chip ............................................... 1  
  1.2 Wormhole switching .......................................... 2  
  1.3 XY routing .................................................... 4  
  1.4 Motivation and problem statement ......................... 5  
  1.5 Structure of thesis ........................................ 5  

2 Unicast ............................................................... 7  
  2.1 Unicast model ................................................ 7  
  2.2 Implementation ............................................. 11  
    2.2.1 Overview ............................................... 11  
    2.2.2 Data path ............................................. 14  
    2.2.3 Control path ......................................... 17  
    2.2.4 Admission and sink .................................. 22  
  2.3 Validation .................................................... 23  

3 Multicast ............................................................ 32  
  3.1 Multicast scheme .......................................... 32  
  3.2 Deadlock avoidance ...................................... 36  
  3.3 Implementation ........................................... 37  
    3.3.1 Overview ........................................... 37  
    3.3.2 Data path .......................................... 41  
    3.3.3 Control path ...................................... 46  
    3.3.4 Admission and sink ................................ 49  
  3.4 Experimental results .................................. 49  

4 Summary ............................................................ 57  
  4.1 Conclusion ................................................ 58  
  4.2 Future work ............................................... 58  

A Code structure .................................................. 59
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B</strong> Power</td>
<td>61</td>
</tr>
<tr>
<td><strong>C</strong> Synthesis area results</td>
<td>63</td>
</tr>
<tr>
<td>C.1 Unicast router with decoupled admission</td>
<td>63</td>
</tr>
<tr>
<td>C.2 Unicast router with coupled admission</td>
<td>65</td>
</tr>
<tr>
<td>C.3 Unicast router with decoupled admission optimized for timing</td>
<td>67</td>
</tr>
<tr>
<td>C.4 Unicast router with coupled admission optimized for timing</td>
<td>69</td>
</tr>
</tbody>
</table>
List of Figures

1.1 A 2D mesh ........................................... 2
1.2 Wormhole switching ................................ 3
1.3 Deadlock ............................................. 4
1.4 Virtual channels ................................. 4

2.1 A canonical unicast wormhole router ............ 7
2.2 Router processing flow ............................. 8
2.3 The coupled admission model ...................... 9
2.4 The ideal sink model ............................... 10
2.5 The p-sink model .................................... 10
2.6 Definitions of the unicast flits ................... 12
2.7 The implementation of the main part of a unicast router ... 13
2.8 The implementation of the crossbar in the decoupled admission 15
2.9 The implementation of the crossbar in the coupled admission . 16
2.10 The FSM of the unicast router ..................... 18
2.11 Nodes numbers in the mesh ....................... 19
2.12 The implementation of the admission part ........ 22
2.13 The implementation of the sink part ............... 23
2.14 Queuing buffers ................................. 24
2.15 Packet latency without source queuing time ...... 25
2.16 Power spectrum of any one sequence of numbers .... 25
2.17 Correlation of any two sequences of numbers .... 26
2.18 Test results of the ideal sink model ............... 27
2.19 Test results of the p-sink model ................. 28
2.20 Test results of the p-sink under the locality traffic ..... 29
2.21 Test results of the p-sink with the coupled admission .... 30
2.22 Test results of the p-sink with the coupled admission under the locality traffic .................. 30

3.1 Tree-based multicasting ........................... 32
3.2 Path-based multicasting ........................... 33
LIST OF FIGURES

3.3 Multicast packet format ........................................ 33
3.4 A cycle in a group .................................................. 36
3.5 Deadlock in sinking ................................................ 37
3.6 A multicast wormhole router model .............................. 38
3.7 Definitions of the flits ............................................ 39
3.8 The implementation of the main part of a multicast router 40
3.9 A multicast record in the flit type process module .......... 41
3.10 Flow of the flit type process module 1 ........................ 42
3.11 Flow of the flit type process module 2 ........................ 43
3.12 A multicast record in the multicast record module ........ 46
3.13 The FSM of the multicast router ............................... 47
3.14 Packet latency with source queuing time ..................... 50
3.15 Test in a linear array ............................................ 50
3.16 Test results of the pure unicast and the pure multicast in a
  linear array ............................................................. 51
3.17 Test in a mesh ...................................................... 52
3.18 Test results of 8.7% multicast traffic in a mesh .......... 53
3.19 Test results of 16.2% multicast traffic in a mesh .......... 54
3.20 Test results of 28.5% multicast traffic in a mesh .......... 54
3.21 Test results of 16.2% multicast traffic without lane reservation
  in a mesh ................................................................. 55
3.22 Test results of 16.2% multicast traffic with lane reservation in
  a mesh ................................................................. 56

A.1 The structure of VHDL files ...................................... 60

B.1 Power of the unicast wormhole router under the uniform traffic
  [4] ................................................................. 61
B.2 Power of the unicast wormhole router under the locality traffic
  [4] ................................................................. 62
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Minimum average latency</td>
<td>28</td>
</tr>
<tr>
<td>2.2</td>
<td>Synthesis results of the unicast wormhole router</td>
<td>31</td>
</tr>
<tr>
<td>2.3</td>
<td>Synthesis results of the unicast wormhole router when optimized for timing in the control path</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>The flits releasing the resources or not</td>
<td>48</td>
</tr>
<tr>
<td>3.2</td>
<td>The flits going through the admission or the sink</td>
<td>49</td>
</tr>
</tbody>
</table>
Chapter 1

Background

1.1 Network-on-Chip

More and more processor cores and large reusable components have been integrated on a single silicon die, which has become known under the label System-on-Chip (SoC). Buses and point-to-point connections were the main means to connect the components. But as silicon technology advances further, problems related to buses have appeared. First, buses do not scale as the number of communication partners connected becomes higher. Second, long and global wires and buses become undesirable due to their low and unpredictable performance, high power consumption and noise phenomenon. Third, due to the unpredictability of the communication performance, designing and verifying a large bus based communication networks is very hard. Fourth, every system has a different communication structure, making its reuse difficult. So researches in systematic approaches to the design of the communication part of SoC are needed at all levels from the physical to the architectural to the operating system and the application level. So Network-on-Chip (NoC) is used mostly in a very broad meaning, encompassing the hardware communication infrastructure, the middleware, operating system communication services, the design methodology and tools to map applications onto a NoC. All these together can be called a NoC platform [5].

The NoC architecture, provides the communication infrastructure for the resources. In this way it is possible to develop the hardware of resources independently as stand-alone blocks and create the NoC by connecting the resources as elements in the network. Moreover, the scalable and configurable network is a flexible platform that can be adapted to the needs of different workloads, while maintaining the generality of application develop-
ment methods and practices. A NoC consists of resources and routers that are directly connected to each other, so that resources are able to communicate with each other. A resource is a computation or a storage unit such as a processor core or a memory. Routers route and buffer messages between resources from their sources to the destinations in the network. Each router is connected to four other neighboring routers through input and output channels. A channel consists of two one-directional point-to-point buses between two routers or between a resource and a router. A NoC with a mesh topology is shown in figure 1.1, where $R$ is resource, $S$ is router and Arrow is the channel.

![A 2D mesh](image)

Figure 1.1: A 2D mesh

### 1.2 Wormhole switching

Nodes in a direct network communicate by passing messages from one node to another. A message enters the network from a source node and is routed towards its destination through a series of intermediate nodes. Four types of switching techniques are usually used for this purpose: circuit switching, packet switching, virtual cut-through switching, and wormhole switching.

In circuit switching, a dedicated path is established between the source and the destination before the data transfer initiates. The message is never blocked during transfer. In order to improve performance, packet switching is used. In packet switching, a message is divided into packets that are independently routed towards their destination. The destination address is encoded in the header of each packet. The entire packet is stored at
CHAPTER 1. BACKGROUND

every intermediate node and then forwarded to the next node in its path. In order to reduce the time to store the packets at each node, virtual cut-through switching is introduced. In this technique, a message is stored at an intermediate node only if the next channel required is occupied by another packet.

Figure 1.2: Wormhole switching

Wormhole switching is a variant of the virtual cut-through technique that avoids the need for large buffers for saving messages. In wormhole switching, a packet is transmitted between the nodes in units of flits, the smallest units of a message on which flow control can be performed. The head flit of a message contains all the necessary routing information and all the others flits contain the data elements. The flits of the message are transmitted through the network in a pipelined fashion. It is shown in figure 1.2. The main advantage of wormhole switching derives from the pipelined message flow, since transmission latency is insensitive to the distance between the source and the destination. Moreover, since the message moves flit by flit across the network, each node needs to store only one flit. The reduction of buffer requirements at each node has a major effect on the cost and the size of systems. The main disadvantage of wormhole switching comes from the fact that only the head flit has the routing information. If the head flit can not advance in the network due to resource contention, all the trailing flits will be also blocked along the path and these blocked messages can block other messages. This reduces network performance drastically and this chained blocking can also lead to deadlock, as shown in figure 1.3. In this situation, each message wants to turn left, but the buffer that it wants to use is occupied by another one. Messages wait for each other in a cycle and hence no message can advance any further. Prevention of deadlock in wormhole switching is usually accomplished by a suitable choice of routing algorithm that selectively prohibits the message from taking all the available paths, thus preventing cycles in the network [13].

Virtual Channels (VCs) can be used in wormhole-switched networks to prevent deadlock and reduce effects of chained blocking. A virtual channel is a logical abstraction of a physical lane. All the virtual channels associated
with a physical channel have individual flit buffers and are time-multiplexed for message transmission using the physical channel. An example is shown in figure 1.4.

1.3 XY routing

Routing algorithms can be classified as deterministic or adaptive according to the path selection process. In a deterministic routing, the path from the source to the destination is determined by the current node address and the destination node address. For the same source-destination pair, all
packets follow the same path. To avoid network congestion and enhance fault
tolerance, it is preferred that the routing algorithms provide paths according
to the dynamic network and traffic conditions. This kind of algorithms is
called adaptive routing algorithms.

Dimension-order routing is a deterministic routing scheme in which the
selected path traverses network dimensions in sequence. The network dimen-
sions are arranged in a predetermined monotonic order. Dimension-order
routing in two-dimensional meshes is called XY routing. The two dimen-
sions of a mesh are labeled as X and Y. Because cycles can not be formed
with XY routing, it is deadlock-free.

1.4 Motivation and problem statement

Network-on-Chip is being proposed as a global communication platform for
future SoC applications. It has a lot of merits: scalability, reusability, etc.
Buses (a single bus, segmented buses or a hierarchy of buses) do not scale
with the system size in bandwidth and clocking frequency. However, it is
very efficient in broadcasting. Although a network allows many more concur-
rent transactions, implementing multicast may be inefficient if not addressed
properly. The normal multicasting scheme transfers multicast packets by
routers in the same way as unicast packets. This scheme does not guarantee
Quality-of-Service (QoS). Based on this observation, a hardware support-
ing QoS multicast scheme is proposed, and a synthesizable RTL design is
developed and implemented in this thesis.

1.5 Structure of thesis

- Chapter 2
  The canonical model of unicast is introduced. Then a unicast wormhole
  router is designed and implemented in RT level. Moreover, experiments
  are conducted to validate the design.

- Chapter 3
  A novel hardware supporting QoS multicast scheme is proposed. Some
  situations resulting in deadlocks are discussed, and methods to avoid
  them are brought up. After that a multicast wormhole router is de-
  signed and implemented based on the unicast router in RT level. In
  the end, some tests have been done to show the merits.
• Chapter 4

First, the thesis work is summarized. Then conclusions are drawn and future work is discussed.
Chapter 2

Unicast

2.1 Unicast model

Figure 2.1: A canonical unicast wormhole router

Figure 2.1 shows a canonical wormhole router architecture with virtual channels at inputs [10]. It has $p$ Physical Channels (PCs) and $v$ lanes per
PC. A packet passes the router through four stages: Routing, Lane allocation, Flit scheduling and Arbitration [2]. Consider that a packet consists of three flits, one head flit, one body flit and one tail flit. When the head flit arrives, according to its flit type field, the lane sends out the routing request and enters routing stage. Then the routing logic calculates the output path by its destination field according to routing algorithm. After receiving the grant from the routing logic, the lane sends out the lane allocation request and goes to lane allocation stage. The lane allocator finds out an available lane on its output path in the next hop and associates that lane to the lane in this hop together, so that other packets can not use them until this association is released. If the lane allocation succeeds, the lane will receive a grant from the lane allocator, send out a arbitration request, go to the arbitration stage and wait for the transfer; otherwise the lane will stay in the allocation stage. The arbitration includes two levels. The first level is about the lanes sharing the same physical channel to the crossbar. The second level is for the crossbar traversal. If one lane wins all the two levels of the arbitration, the lane will receive a grant from the arbitration. Then the flit in this lane is transmitted through the crossbar which is controlled by the arbitration. When the data or tail flit arrives, according to its flit type field, the lane enters the flit scheduling stage. If there is a buffer available in the next hop lane associated by the head flit, a grant will be sent out from the flit scheduler to the lane and the lane will enter the arbitration stage and perform the same as the head flit, otherwise it will stay here till success. When the tail flit leaves the router, the router will release the lane association. Then the head flits of other packets can use the lane. The flits from different packets will not be interleaved in a lane since the lane is kept being associated to the previous lane while transmitting the whole packet. This flow is shown in figure 2.2.

![Diagram](image)

**Figure 2.2: Router processing flow**

Credits are passed between adjacent routers in order to keep buffers statuses of lanes used in the lane allocation and the flit scheduling. When a flit succeeds in the lane allocator or the flit scheduler, one buffer in the next hop lane will be used. So buffer usage status will be increased in this router. If a flit goes out from the next hop lane, one buffer of that lane will be empty.
So a credit plus signal is sent back to this router to decease the buffer usage status.

In the admission part, there are a lot of lanes used for admitting packets into the network. Their length is the maximum number of flits that a packet can contain. If an uploading lane is available, a packet will be split into smaller flits and the flits are put into one uploading lane in order. Then the packet will also pass the router through four stages: **Routing**, **Lane allocation**, **Flit scheduling** and **Arbitration**, the same as the lanes at the input channels (channels are connected to outputs of other routers). The model in which the packet in one admission lane can be routed to any output channel, is called ideal admission model or decoupled admission model, which was shown in figure 2.1. If the packet in one admission lane can only be routed to one specific output channel, the model is called coupled admission model. It is shown in figure 2.3 [2]. This scheme alleviates the complexity of the router, especially in the crossbar, and achieves approximately the same performance as the decoupled admission model. This has been proved by Zhonghai Lu in his paper [2] and will be proved in RTL design in this chapter.

![Diagram of the coupled admission model](image_url)

**Figure 2.3: The coupled admission model**

In the sink part, there are many lanes used for sinking packets from the network. Their length is the maximum number of flits that a packet can contain, equal to the length of the admission lanes. When the head flit of a packet arrives at the destination router, the lane allocator associates the lane to one sink lane. Then the head flit bypasses the crossbar and enters
its sink lane. When the body flit or the tail flit of that packet arrives, it enters the corresponding sink lane. When all flits of a packet have arrived, the whole packet is composed and then delivered out of the network. The model in which each lane at the input channels has its own sink lane, is called ideal sink model, which is shown in figure 2.4. In this model, there are \( p \times v \) sink lanes. If there are only \( p \) sink lanes and all the lanes at the input channels share them for sinking, the model is called \( p \)-sink model. Figure 2.5 shows this model [3]. The \( p \)-sink model reduces the complexity of the router without decreasing the performance significantly. This has been shown in [3] and is presented as an RTL design in this chapter.

### 2.2 Implementation

#### 2.2.1 Overview

According to the canonical model mentioned in the previous section, a unicast wormhole router is designed and implemented.

Since a unicast packet consists of one head flit, some body flits and one tail flit, here are three kinds of flits: the head flit, the body flit and the tail flit. The definitions are shown in figure 2.6.

The most significant part is the \( VC.ID \) field, which is used to select the lane that the flit will occupy in the next hop router. The \( Flit.type \) field is used to distinguish different types of the flits. In the head, body or tail flit, it is “0000”, “0100” or “1000”, respectively. The \( X_nY_n \) field is used to indicate the destination address. The \( Data \) field is used to carry the data.

The whole design is a generic design. Some parameters can be configured in the global package to generate the corresponding implementation, making the design very flexible and general. These parameters are:

- Flit width of the transmission data
- Number and length of the VCs per PC
- Number and length of the VCs in the admission part and the sink part

Implementations of the three parts are shown in figures 2.7, 2.12 and 2.13. Values of the parameters used in the figures are:

- Flit width of the transmission data = 32
- Number of the VCs per PC = 6; length = 2
Figure 2.6: Definitions of the unicast flits

- Number of the VCs in the admission part = 4; length = 6
- Number of the VCs in the sink part = 4; length = 6

Figure 2.7 shows the main part of the design. It is composed of the following modules: four Input channels, Crossbar, Allocator next, Allocator sink, Scheduler next, Scheduler sink, Arbiter, Forward lane status and Sink lane status. The data path and the control path are separated by the dashed line. They use the different clocks. The frequency of control path clock is twice as high as the frequency of data path clock.

Each Input channel corresponds to one physical input channel. If the router works in a 2D mesh, there are 4 input channel modules used for 4 directions. The Crossbar size is 8 × 8, from 4 input channels and 4 admission lanes to 4 outputs to other routers and 4 sink lanes. The lane allocation, the flit scheduling and the lane status modules are different from the canonical model. They are divided into 2 parts respectively: next and sink. The next
Figure 2.7: The implementation of the main part of a unicast router
ones are used for the flits that go to another router, and the sink ones are used for the flits that go to the sink part. This separation can make the router handle the flits that go forward or to sink in a more flexible and efficient way.

### 2.2.2 Data path

The data path is discussed in the following paragraphs. Some modules are in the *Input channel*, others are not.

First, the data path part of the *Input channel* is introduced.

The data path part of each input channel includes a *Flit type process*, a *Demultiplexer*, *v* VCs and a *Multiplexer*. The control path part includes a *FSM array* which consists of *v* FSMs, a *Routing selector*, a *Routing logic* and a *Credit out*.

In the data path part of the *Input channel*, when the flit arrives at the router, it first comes to the *Flit type process* before going to the *Demultiplexer*. This module is used for generating a 3-bit control signal to the FSM according to the flit type. Then this signal is sent out together with the flit to the lane which is specified by the *VC_ID* in the flit. The 3-bit control signal for the different flit types are different. For the head flit it is “000”, but for the body or the tail flit it is “001”. This module makes the flit type independent of the signal used by the FSM, so it is easier to make the same flit enter different states in the FSM under different situations. It also makes adding multicasting functions easier. The *VC_ID* in the flit directly controls the *Demultiplexer* to select the target lane. There is no need to transmit the *VC_ID* signal into the lane, because it is useless after the flit enters the lane. So the width of the lane is: \[ \text{Bit}_{\text{transmission\_data}} - \log(v) + \text{Bit}_{\text{flit\_type}}. \]

According to the global parameters specified in the beginning of the section, the width of lane is 32 bits. Because a lane works in the way of first in first out, a FIFO is used to implement it. After a flit is handled by the *Flit type process*, a write signal is sent to the FIFO to store the flit. When received a read signal, the FIFO outputs one flit and then the FSM works according to the 3-bit control signal in the flit. After processed by the control path, the flit is output through the *Multiplexer* at the end of the data path part of the *Input channel* which is controlled by the *Arbiter*. This is the first level of the arbitration mentioned in the canonical model.

After discussing the data path in the *Input channel*, the other data path module, the *Crossbar* is detailed.

The *Crossbar* is at the end of the data path. It is controlled by the *Arbiter*. This is the second level of the arbitration as mentioned in the canonical model. The *Crossbar* is implemented in multiplexers. There are two kinds of crossbars used in the thesis. Figure 2.8 shows the crossbar
CHAPTER 2. UNICAST

Figure 2.8: The implementation of the crossbar in the decoupled admission
Figure 2.9: The implementation of the crossbar in the coupled admission
used in the decoupled admission model. There are $p + v_{\text{sink}}$ multiplexers. Each one is a $p + v_{\text{admission}}$ to 1 multiplexer with $\log(p + v_{\text{admission}})$ bits selection signal and 1-bit enable signal. Allocation results for the next hop are combined with the corresponding flits at the outputs connected to other routers. These allocation results are sent out by the Arbiter. This scheme can save many resources compared to combining them in the outputs of the FIFOs at the Input channels. The number of multiplexers used in combining them in the outputs of the FIFOs is $p \times v$, but here is $p$. The crossbar used for the coupled admission model is shown in figure 2.9. It consists of $p \times (p + 1 \text{ to } 1)$ multiplexers and $v_{\text{sink}} \times (p \text{ to } 1)$ multiplexers.

### 2.2.3 Control path

The control path is discussed in the following paragraphs. Some modules are in the Input channel, others are not.

First, the control path part of the Input channel is introduced.

As mentioned in the canonical model, the head flit goes through the routing, the lane allocation and the arbitration stages, and the body flit or the tail flit goes through the flit scheduling and the arbitration stages. All of these operations are controlled by the FSM. Each lane has an FSM. Figure 2.10 shows the flow of the FSM. “S0” is the initial state. When there are flits in the FIFO, the FSM sends out a read signal to the FIFO and enters “S8”. The reason to use “S8” is that when the FIFO receives a read signal, it needs one data path clock cycle to output the flit. So after one control path clock cycle, the FSM will enter “S1” automatically. If the flit that goes out of the FIFO is a head flit, the FSM will send out a routing request to the Routing selector and enter “S2”. When one lane wins the selector, the Routing logic will calculate the output direction of that flit and send the result together with a grant signal back to the corresponding FSM. The FSM saves the routing result in registers. If the packet does not arrive at the destination, the FSM will send a request to the Allocator next, otherwise it will send a request to the Allocator sink. At the same time the FSM enters “S3”. After successful processing, the lane allocator sends back a grant signal to the FSM. Then the FSM requests the Arbiter and enters “S7”. If the arbitration succeeds, the Arbiter will send a feedback signal to the FSM, otherwise the flit will be waiting at the current state “S7” until the arbitration succeeds. When the body flit or the tail flit comes out of the FIFO, the FSM enters “S4”. If the packet does not arrive at the destination, the FSM will send a request to the Scheduler next. Otherwise, the FSM will send a request to the Scheduler sink. If scheduling succeeds, a grant will be sent back to the FSM. Then the FSM requests in the Arbiter the same as the head flit and
Figure 2.10: The FSM of the unicast router
enters "S7". After the FSM receives the feedback from the Arbiter, the flit will be transmitted out of the router. At the same time, the FSM will check if there is any flit in the FIFO. If so, the FSM will send out immediately a read signal to the FIFO and enter "S8". Otherwise, the FSM will enter the initial state "S0" and wait for the next flit.

Since one Input channel corresponds to one input physical channel and at most one new flit will arrive at each Input channel every data path clock cycle, one Routing logic is enough for each input channel. The Routing selector is used to guarantee that only one lane does routing operation each control path clock cycle. When receiving a request, the Routing selector stores it in the waiting list. Then the selector selects out one lane from the list according to the algorithm, and sends out its destination information with the lane number to the Routing logic. There are many algorithms to implement the selection operation. In order to give every lane the equal chance to get routing, Round-Robin is used here. It has strong fairness. A Round-Robin algorithm works on the principle that a request that was just served should have the lowest priority on the next round of selection. So if a lane is selected out to get routing, it will have the lowest priority in the next round. All the other lanes will be checked before that lane can be checked again.

![Figure 2.11: Nodes numbers in the mesh](image)

The routing algorithm is implemented in the Routing logic. XY routing is used. This is the simplest and the most universally used routing algorithm. A flit from node \(X_a Y_a\) to node \(X_b Y_b\) will first go along the X direction in the mesh. When it reaches the value of \(X_b\), it goes along the Y direction and reaches \(Y_b\) at last. No flit can go from the Y direction to the X direction. Figure 2.11 shows the nodes numbers in the mesh. When the Routing logic
receives the data from the FSM, it will calculate the output physical channel. Then the routing result and a grant are sent back to the corresponding FSM. The routing result is organized in a 6-bit vector “bbbbb”. The leftmost bit is the most significant bit. The 6th bit is the result of the “OR” operations of the 5th bit to the 2nd bit. This means that if there is a high in the 5th, 4th, 3rd, or 2nd bit, the 6th bit will be high, otherwise it will be low. The 5th, 4th, 3rd and 2nd bits represent the four directions of up, right, down, left in the mesh, respectively. The 1st bit represents the sink direction. When a flit arrives at its destination, this bit is set to high, otherwise it is low. There is another way to represent the routing result, an encoding scheme. For example “000” represents up, “001” represents right, “010” represents down, “011” represents left and “100” represents sink. The reason of using the former scheme rather than the latter is that if a flit goes to sink and up directions together, the former can represent it as “110001”, but the latter cannot handle it. Since this kind of situations is very common in multicasting, the former is adopted to make the design more flexible and more extensible.

Another module is the Credit out. It is used to generate the credit plus signal to the Forward lane status in the neighboring router connected to this input channel. A credit plus signal consists of a VC.ID with an enable. It will be generated, when a flit wins the arbitration and goes out of the router. At most one flit can be output per data path clock cycle from each input channel, so one credit plus signal for each input channel is enough.

After discussing the control path in the Input channel, other modules are detailed in the next paragraphs.

The allocator modules are used to find out an available lane on the output path for the head flits. The Allocator next is used for the flits that go to the next router. The Allocator sink is used for the flits that go to the sink. When an allocation request arrives, it is immediately stored in a register. Then the allocators check the lanes statuses in the corresponding lane status modules, where the buffer usage status of the forward and sink lanes are stored. The Allocator next checks the Forward lane status and the Allocator sink checks the Sink lane status. If there is a FIFO that is not full and is not in using status, it will be associated to this lane by the head flit. Then the allocators save allocation results (VC.IDs) in an encoding style. For example the VC.ID for the 1st lane is “00” and for the 3rd lane is “11”. The allocators will also set a using mark in that lane to forbid the other packets to use it. In the mean time they send out an add signal to the Forward lane status or Sink lane status to increase the number of used buffers. Then the allocation finishes and a feedback is sent back to the FSM. The allocation algorithm is very important and will affect the performance. Round-Robin is used. There are two places that need to use Round-Robin: each output to other router
CHAPTER 2. UNICAST

and the lanes in that direction. The first Round-Robin makes the lanes that want to go to the same direction be Round-Robin selected. The second one makes the lanes in the next router be Round-Robin used. Another important function of the allocators is the tail flits checking. If a flit goes out through the Crossbar, the allocators will check its flit type field to see if it is a tail flit. If so, the allocators will clear the using status of the lane which the tail flit is heading for, so other packets can use it.

The scheduler modules are responsible for the body and the tail flits. The Scheduler next is used for the flits that go to next router, and the Scheduler sink is used for the flits that go to the sink direction. They check the buffer usage status in the corresponding lane status modules. If there is an available buffer in the lane associated by the head flit, the scheduling succeeds. Then the schedulers send out a grant signal to the FSM and an adding signal to the lane status modules to increase the status of used buffers. Because scheduling only functions as checking the buffer status of the associated lane, there is no need to use a scheduling algorithm.

The Forward lane status and Sink lane status are stored in the independent modules. They are connected to the allocators and schedulers. They update the statuses at every control path clock cycle. The status is recorded in an encoding scheme, e.g. "11" means three buffers are used. When the lane status modules receive adding signals from allocators or schedulers, they will increase the number of used buffers by 1 in the corresponding lanes and then refresh the new lane status to the allocators and the schedulers. It is impossible to receive adding signals for the same lane from the allocators and the schedulers. Because only one flit can be processed in each lane, it must be a head flit, a body flit or a tail flit, so it needs to be either allocated or scheduled. Another important signal in this module is the credit plus signal. If a flit goes out of a lane in the forward router, the router will send out a credit plus signal to the Forward lane status in this router. Then the buffer usage status will be decreased by 1 according to the lane number. In contrast to the Forward lane status, in the sink lane there is a clear signal instead of the credit plus signal. This clear signal comes from the sink part. When the Sink lane status receives a clear signal, it clears the number of used buffers to zero. This is because in the sink part the output unit is a packet not a flit. When the whole packet arrives, it is sent out entirely and then the sink lane will be empty.

The arbitration is divided into two levels as mentioned before. The decisions are made together. When the arbitration request comes, it is stored in the register. Because at most one flit can transmit from the each input channel through the crossbar to the output in each data path clock cycle, a selection algorithm is needed. For each output, a Round-Robin is used.
CHAPTER 2. UNICAST

This gives every lane the equal chance to output the flit. When the Arbiter selects out one lane to the output, the flit will pass the Multiplexer from each input channel and the crossbar, so the control signals should be sent to them. At the same time, the grants for these lanes will be sent back to the FSMs. The Arbiter also sends out the allocation results of the granted lanes to the crossbar. Priority is used in the arbitration to achieve better performance. There exist two kinds of priorities: the flits that go to sink which arrive at the destination, and the flits that go to the next router. The former have higher priority than the latter. This makes the router sink more quickly.

2.2.4 Admission and sink

The Admission channel and the Sink channel are discussed respectively.

![Diagram](image)

Figure 2.12: The implementation of the admission part

The first module is the Admission channel. It is almost equal to the Input channel. It is shown in figure 2.12. In this module there are \( v_{\text{admission}} \) virtual channels. The length of them is \( Width_{\text{packet}}/Width_{\text{flit}} \), and the width of them is the width of a flit. When a packet goes to the Admission channel, it is decomposed into flits. Then the Admission channel checks which lane has available buffers using Round-Robin algorithm, and puts the flits into that lane one by one. After the whole packet is put into the lane, a grant signal is sent to the outer IP block and a new packet can come. After that, the packet passes the router through the same four stages: Routing, Lane allocation, Flit scheduling and Arbitration.

The Sink channel does the operations opposite to the Admission channel. It composes the flits to the packets. When the head or the body flit goes to
the Sink channel, it is saved in the lane associated by the head flit and waits for the tail flit. When the corresponding tail flit arrives, the Sink channel composes the separate flits to a whole packet and sets a output request to the outer IP block. After completely sending out and getting a receive grant signal from the outer IP block, the Sink channel sends a clear signal to the Sink lane status. Figure 2.13 shows its implementation.

There are clocks used in the design as mentioned in the beginning of the implementation. So when signals go out of one module to another module with different clock, the synchronization is needed. For example, the signals that go out of the Arbiter to control the Crossbar must be held during the data transmission.

The above sections are the whole implementation and working procedure of the unicast wormhole router. It is not a pipelined design. However, by using the enough VCs, more than 4, it can achieve the same performance as the pipelined one.

### 2.3 Validation

After the design and implementation of the unicast wormhole router, some tests are done to validate it.

Four formulations are used to process test results and draw curves:

- **Network load** (fraction of link capacity):

\[
\frac{\text{Sum}_{\text{active_link}}}{\text{Number}_{\text{link}} \times \text{Cycle}_{\text{simulation}}}
\]
• Average packet latency:
\[
\frac{\text{Sum}_{\text{latency}}}{\text{Number}_{\text{packet\_received}}}
\]

• Packet injection rate:
\[
\frac{\text{Number}_{\text{flit\_injected}}}{\text{Number}_{\text{node}} \times \text{Cycle}_{\text{simulation}}}
\]

• Throughput:
\[
\frac{\text{Number}_{\text{flit\_received}}}{\text{Number}_{\text{node}} \times \text{Cycle}_{\text{simulation}}}
\]

For testing, packet source queuing buffers is put at the input of each admission channel, as shown in figure 2.14. Packets from the outside will be sent to this queuing buffers at a constant rate $\alpha$ and then wait for processing. The router takes packets out of the queuing buffers at a rate $\beta$. The $\alpha$ is determined by the source. The $\beta$ is determined by the processing rate of the router and is related to the design and implementation. They are calculated based on the data path clock. The length of buffers should be large enough, so that the buffers will not overflow when the $\alpha$ is higher than the $\beta$. By testing at different values of the $\alpha$, different packet latencies and throughputs can be got. The curve can be drawn.

![Diagram of packet queuing](image)

Figure 2.14: Queuing buffers

The way of calculating packet latency used here is shown in figure 2.15. The latency begins to be recorded at the end of the source queuing buffers, when a packet is just put into the admission channel. It stops being recorded when the packet is out of the network.

Two kinds of test patterns for sending packets are used here, the uniform traffic and locality traffic. The uniform traffic entails that each router sends out packets to all the other nodes randomly. So every node will receive the equal number of packets sent out by this node. The locality traffic means that more packets will be sent to the closer nodes, and fewer packets will be sent to the farther nodes. So every node will receive more packets sent out by the closer nodes and fewer packets sent out by the farther nodes.
Figure 2.15: Packet latency without source queuing time

Figure 2.16: Power spectrum of any one sequence of numbers

In the tests, the total number of the packets to be sent is 16000, thus 1000 for each node. The uniform traffic is generated by a uniform function, and each router uses a different seed to generate a series of numbers. They are random. This is proved by the power spectrum figure 2.16. In the frequency domain, each sequence of numbers is a white noise in reality. Different se-
CHAPTER 2. UNICAST

Figure 2.17: Correlation of any two sequences of numbers

quences are not correlated. This is proved by the correlation figure 2.17. The axis X is the total number of the numbers. The axis Y is the normalized relation.

When one of the routers finishes sending, the test is over. The packet injection rate $\alpha$ is presented in the format of $1/n$, from 1 to 1/25, 1/30 and 1/40. We began to record the test data at the 3000th data path clock cycle. This will avoid the effect of warm-up.

The first test is under the decoupled admission model, ideal sink model and uniform traffic. The result of the 4 VCs and the 8 VCs schemes are shown in figure 2.18.

Test conditions are:

- Mesh size: 4 x 4
- Number of the VCs per PC: 4; 8
- Number of the VCs per PC in the admission: 4
- Number of the VCs per PC in the sink: 16; 32
- Length of the VCs: 2
• Number of the flits per packet: 4

![Graphs showing network performance metrics](image)

Figure 2.18: Test results of the ideal sink model

From figure 2.18, the “Average packet latency” goes exponentially with the “Network load”. The “Throughput” goes linearly with the “Injection rate” before the network is saturated. After the network is saturated, the throughput is stable. The 8 VCs router can reach higher network load and higher throughput than the 4 VCs one, but it will cost more since there are more VCs.

In meshes, the minimum average latency of a canonical model is:

\[
Minimum \ average \ latency = \frac{2}{3} \cdot N + Length_{packet} = 6.67
\]

N is the number of the rows in meshes, equal to the number of the columns.

But in figure 2.18, the minimum average latency is about 30 data path clock cycles. The reason for this difference is that this design is not a single cycle design. Differences between this design and the single cycle model are shown in table 2.1. The unit is the data path clock cycle. The “Pass” means a flit goes through the router from its input to the input of the next router. The “Admission” means a flit goes through the router from its admission input to the input of the next router. The “Sink” means a flit goes through the router from its input to the output of the sink.
### Table 2.1: Minimum average latency

<table>
<thead>
<tr>
<th>Number of Cycles</th>
<th>Single cycle model</th>
<th>This design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head flit</td>
<td>1 1</td>
<td>5 5 6</td>
</tr>
<tr>
<td>Body or tail flit</td>
<td>1 1</td>
<td>3 3 3</td>
</tr>
</tbody>
</table>

In this design, the formulation should be:

\[
\text{Minimum average latency} = \text{Cycle}_{\text{head admission}} + \frac{2}{3} \times (N - 1) \times \text{Cycle}_{\text{head sink}} + \text{Cycle}_{\text{body sink}} \times \text{Length}_{\text{body and tail}} = 28.33 \tag{2.2}
\]

After discussion, the performance is consistent with the canonical one, so the design and implementation are correct.

The next test is done in p-sink model. In a p-sink model, the number of sink channels is equal to the number of PCs 4. Other test conditions do not change. Results are shown in figure 2.19.

![Figure 2.19: Test results of the p-sink model](image)

In contrast to the ideal sink model, here the 4 VCs one can reach higher
network load and higher throughput than the 8 VCs. This is because there are only 4 sink channels, when 8 VCs are used, the network is easier to be blocked. So after the network is saturated the 8 VCs has worse performance than the 4 VCs.

All the tests above are under the uniform traffic, so a test under the locality traffic is taken. The decoupled admission, the p-sink and 4 VCs are used. Results are shown in figure 2.20. The average packet latency is a little lower than the uniform one. This is consistent with the canonical model.

![Figure 2.20: Test results of the p-sink under the locality traffic](image)

Next, the coupled admission model is tested. In this model each admission VC will only be connected to one direction.

First it is tested with 16 admission VCs and the p-sink model. Each 4 admission VCs are connected to one direction. Because the router model is not a single cycle model, if only 4 admission VCs are used, there will be a lot of spare time in the links. The results are shown in figure 2.21. As can be observed, the performance is almost the same as the decoupled one. This is discussed by Zhonghai Lu in his paper [2].

The next test is under the locality traffic. 4 VCs are used. Results are shown in figure 2.22. The average packet latency is a little lower than the uniform one, and the performance is almost the same as the decoupled one under the locality traffic.

This design is a synthesizable VHDL design. The synthesis tool is Synopsys Design Compiler, and the version is 2004.06 SP2, 32-bit. The tech-
CHAPTER 2. UNICAST

Figure 2.21: Test results of the p-sink with the coupled admission

Figure 2.22: Test results of the p-sink with the coupled admission under the locality traffic

ology library is UMC18 (180nm). In this library, a NAND gate has area 12.197µm². The mapping effort is medium. Since the FIFOs from Altera library are used, they are not counted in the synthesis result. The results of the unicast router with 4 VCs are shown in table 2.2. The area of the
CHAPTER 2. UNICAST

A coupled admission model is smaller than the decoupled model, and it can reach a higher frequency. When optimized for timing in the control path, the frequency can reach higher by trading off the area. This shown in table 2.3. The area about each module is listed in the appendices.

<table>
<thead>
<tr>
<th></th>
<th>Decoupled</th>
<th>Coupled</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossbar</td>
<td>2533 gates</td>
<td>1478 gates</td>
<td>41.7%</td>
</tr>
<tr>
<td>Arbiter</td>
<td>10277 gates</td>
<td>8682 gates</td>
<td>15.5%</td>
</tr>
<tr>
<td>Other logic</td>
<td>41718 gates</td>
<td>41172 gates</td>
<td>1.31%</td>
</tr>
<tr>
<td>Whole design</td>
<td>54528 gates</td>
<td>51332 gates</td>
<td>5.86%</td>
</tr>
<tr>
<td>Max data path</td>
<td>398 MHz</td>
<td>400 MHz</td>
<td>-0.5%</td>
</tr>
<tr>
<td>Max control path</td>
<td>43 MHz</td>
<td>53.4 MHz</td>
<td>-24.2%</td>
</tr>
<tr>
<td>Whole design</td>
<td>21.5 MHz</td>
<td>26.7 MHz</td>
<td>-24.2%</td>
</tr>
</tbody>
</table>

Table 2.2: Synthesis results of the unicast wormhole router

<table>
<thead>
<tr>
<th></th>
<th>Decoupled</th>
<th>Coupled</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossbar</td>
<td>2533 gates</td>
<td>1478 gates</td>
<td>41.7%</td>
</tr>
<tr>
<td>Arbiter</td>
<td>13087 gates</td>
<td>11074 gates</td>
<td>15.4%</td>
</tr>
<tr>
<td>Other logic</td>
<td>45242 gates</td>
<td>46587 gates</td>
<td>-2.97%</td>
</tr>
<tr>
<td>Whole design</td>
<td>60862 gates</td>
<td>59139 gates</td>
<td>2.83%</td>
</tr>
<tr>
<td>Max data path</td>
<td>389 MHz</td>
<td>392 MHz</td>
<td>-0.77%</td>
</tr>
<tr>
<td>Max control path</td>
<td>152 MHz</td>
<td>154 MHz</td>
<td>-1.32%</td>
</tr>
<tr>
<td>Whole design</td>
<td>76 MHz</td>
<td>77 MHz</td>
<td>-1.32%</td>
</tr>
</tbody>
</table>

Table 2.3: Synthesis results of the unicast wormhole router when optimized for timing in the control path

The power of unicast wormhole router are tested by in a related project [4]. They are shown in figures B.1 and B.2 in the appendices.
Chapter 3

Multicast

3.1 Multicast scheme

In one-to-all multicast communication, one sender can send out multicast packets to a group of receivers. The packets are addressed to a group of receivers rather than to a single receiver, so each receiver in that group will receive one copy.

![Diagram of multicast tree]

Figure 3.1: Tree-based multicasting

Since the software approach is not efficient enough, hardware support of multicast communication is proposed. Two approaches to implement it are path-based multicasting and tree-based multicasting. In tree-based multicasting, a multicast packet is delivered along a common path as far as possible. Then it is replicated, and each copy of it is forwarded on a different channel bound for a unique set of destination nodes. The path followed by each copy may further branch in this manner until the packet is delivered to every destination node [9]. It is shown in figure 3.1. The grey one is the sender,
the black ones are receivers and the dashed line is the transmission path. To support deadlock-free multicast or broadcast wormhole routing, the tree-based communication pattern does not perform well if packets are not short, because the entire tree is blocked if any of its branches is blocked. A solution is to prohibit branching at intermediate nodes, leading to a multicast path pattern. In path-based multicasting, the head of each packet has multiple destination addresses. The sender arranges these destinations as an ordered list, depending on their intended order of traversal. As soon as the packet is injected into the network, it is routed based on the address list in the leading header flit from the first destination to the last one [9]. Figure 3.2 shows this.

Figure 3.2: Path-based multicasting

In path-based multicasting, each packet must contain all the destination addresses. If many packets are sent to the same destinations, duplicate addresses will be sent. So a new scheme is proposed to solve this waste of the network resources.

Figure 3.3: Multicast packet format

First, the packet format of multicasting shown in figure 3.3 is introduced. Packet fields are explained as follows:
CHAPTER 3. MULTICAST

- **PacketType:** it indicates the type of a packet. It has seven options: unicast, multicast setup, multicast setup add, multicast setup response, multicast data, multicast release and multicast release response.

- **Dadr:** the destination address.

- **MultiID:** the multicast group identity number, which is unique for each multicast group. It is established by one router.

- **Sadr:** the source address. It is the address of the node that initiates the multicast setup. This node is called the group master.

- **GroupType:** the type of the multicast group. This is the key for QoS multicasting. It is used to inform the routers whether the multicast group will reserve a lane or not. *Not reserve lane* means the lanes used by one multicast group can be used by the unicast packets, but can not be used by other multicast groups. They are mixed in the lanes during transmission. *Reserve lane* means the lanes reserved by one multicast group can not be used by the unicast packets or other multicast groups. One group is allowed to reserve only one lane at each input of a router since the number of lanes is limited. Different groups must use the different VCs.

- **MemAdrs:** the addresses of the multicast members. The order of the addresses listed specifies the multicasting path. The first address in the list will be reached first, and then the second and so on. Once reaching a member node with Dadr, the next member address in the list will replace the Dadr field.

The GroupType and MemAdrs fields are only needed in a multicast setup packet.

Second, the multicasting procedure which consists of three phases is introduced:

1. Group establishment: First the router which wants to set up a group (group master) checks if it has a free record to set up. This means that if the number of the groups set up by this router reaches the maximum number, it can not set up any more. If it can, the router sends out a setup packet. The setup packet contains all the multicast destination addresses in order. The setup packet is divided into the setup head, the setup body and the setup tail flits when injected into the network. Then flits are forwarded to the Dadr according to the routing algorithm. When the setup packet reaches a node, the router
records the multicast information and reserves resources according to
the group type. This record will be used later to transmit multicast
data. If it is a group member, the Dadr field will be replaced by the
next group member address. If the setup packet reaches the last group
member, a setup response packet will be sent back to the group master
to acknowledge the success. If the setup fails in a node, for example,
due to lane unavailability, a response packet will be sent back to the
master from the current node to inform the failure. When a setup
failure response packet arrives at the group master, a release packet is
sent out automatically to release the record and reserved resources of
that group.

The group members can be increased by sending out a setup add packet.
It is transmitted in the same way as a multicast data packet before
reaching the last group member, but it will not be delivered to the
nodes. It works like a setup packet after reaching the last group mem-
ber.

2. Multicast communication: After a successful setup, the master can
send multicast data packets. A packet carrying MultiID is divided
into a head flit, some body flits and a tail flit to transmit. It will be
admitted in the same VC and transmitted along the same path and
the same VCs as the setup packet passed before, but it can sink in a
different VC. When a data packet reaches a destination group member
node, it is replicated to sink and also forwarded to the next member. In
this way, all the members will receive the packet. The group members
can also send multicast data packets, but only to the members in the
downstream since a multicasting path is simplex, thus only simplex
communication is allowed.

3. Group release: A group can only be released by its master by sending
a release packet to its members. When the release packet reaches a
node, the multicast record in the router and the reserved lane will be
freed after all on-going group transactions complete. When reaching
the last member, a release acknowledgment is sent back to the master
to indicate the success.

The response packet for multicast group setup or release is handled as a
unicast packet. By our scheme, a multicast group can be established and
released dynamically.
3.2 Deadlock avoidance

Deadlock avoidance is very important in wormhole router.

The first problem is that the XY routing is deadlock free in meshes, but in multicast there will be flits that go from Y to X in the group members nodes, and this will introduce the deadlock. In figure 1.3, on page 4, assume that: A and C are multicast packets; B and D are unicast or multicast packets. Clearly, they form a cyclic dependency A → D → C → B → A. Consequently, none of them can make progress. The packets that cause deadlock are: setup, setup_response (when fails), release_response (when fails) and release (when the master receives a setup failure response and then sends out the release); in the *Not reserve lane* group type multicast data can also cause deadlock.

A solution is proposed to handle this problem. By limiting the number of groups that every input of the node belongs to, to be fewer than the number of the VCs, the deadlock will be avoided. Such as in a 4 VCs network, every input of a node can only belong to fewer than or equal to 3 groups. This guarantees that there is at least one VC available to break any possible dependency cycle due to exhaustion of lane resources.

![Figure 3.4: A cycle in a group](image)

The second problem is the following. Since the group is path based, not tree based, and the multicast records in the router can only store one member address of the downstream, if there are cycles like those in figure 3.4 in the path passed by a group, the router can not forward the multicast data packet in the correct path. Two ways to solve it: the first one is to make the multicast records store two members addresses of the downstream, and give the correct next member address to the multicast data packet when the packet arrives; the second one is to forbid this kind of cycle by arranging the
member addresses in the group setup packets in a proper order.

\[ \text{Figure 3.5: Deadlock in sinking} \]

The third kind of the deadlock occurs in the sink part, shown in figure 3.5. The solid line means occupying, and the dashed line means wanting to forward. This appears when the router has groups in opposite directions. They occupy all the VCs in the sink and all the VCs at the input channels from each direction. Each of them want to forward and sink, but the forward VCs are occupied by the previous fits and the sink VCs are occupied by the fits in the opposite direction. There are two ways to avoid this, the first one is to make the VC long enough to contain a whole packet; the second one is to make the number of the sink VCs bigger than the number of the multicast groups. E.g. in a 3 multicast groups network, 4 sink VCs are enough. In the ideal sink model, the number of sink channels is already more than the number of multicast groups, avoiding this kind of deadlock.

### 3.3 Implementation

#### 3.3.1 Overview

In this section the multicast router is implemented. By adding some functions, the unicast wormhole router becomes a router supporting both unicast and multicast. This will not change the unicast router much and make the design configurable. It is shown in figure 3.6.

Figure 3.7 shows the definition of all flit types used in multicast router including the unicast ones.

The “Uni.head”, the “Uni.body” and the “Uni.tail” fits are used to compose the unicast data packets. The “Setup.head”, the “Setup.body” and the “Setup.tail” fits are used to compose the multicast setup packets or the multicast setup add packets. The multicast data packets used in the Not reserve
CHAPTER 3. MULTICAST

Figure 3.6: A multicast wormhole router model

lane consists of the “Multi_head”, the “Multi_body” and the “Uni_tail”. The multicast data packets used in the Reserve lane consists of the “Multi_head”, the “Multi_body” and the “Multi_tail”. The “Setup_response” flit, the “Release” flit, and the “Release_response” flit corresponds to the multicast setup response packet, the multicast release packet and the multicast release response packet, respectively.

The most significant part is the VC_ID field, which is used to select the lane that the flit will occupy in the next hop router. The Flit_type field is used to distinguish different types of the flits. The XnYN field is used to indicate the destination address. The Data field is used to carry the data. The Multi_ID is the multicast group identity number, used to distinguish the different groups set up by each router. The XmYN is the address of the group master. The Multi_ID and the XmYN makes one multicast group unique in the network. The If_reached_tail field is used to distinguish the setup add packets from the setup packets. If it is ‘1’, this is a setup packet. Otherwise it is a setup add packet. When a setup add packet reaches the last member of the group, this value is changed to ‘1’ and it becomes a setup packet. The Group_type field represent the different types of groups: Unicast – “000”, Not reserve lane – “110” and Reserve lane – “101”. Bits from the X1Y2 to the X14Y14 are the group members addresses. The last group member is followed by a XmYN to indicate the end of this field. The If_succeed means that if the setup succeeds. ‘0’ means failed, and ‘1’ means successful.

The implementation is shown in figure 3.8. The system architecture is equal to that of the unicast router, except the modules with a circle. They are new modules or changed modules. the Multi record is a new module. The Flit type process, the FSM array, the Allocator next and the Arbiter are
Figure 3.8: The implementation of the main part of a multicast router
changed modules. They are discussed in the following paragraphs.

### 3.3.2 Data path

![Data Path Diagram](image)

Figure 3.9: A multicast record in the flit type process module

The module changed in data path is the *Flit type process*. It is extended to support the multicast flits. The first change is that it can store the information about the multicast groups passing the router. The format of the record in it is shown in figure 3.9. The number of the records determines the number of the groups that each input in the router belongs to. The information is recorded from the setup packets, and used by the other kinds of the multicast packets. The *Valid for multidata* indicates that if the group members can send the multicast data packets. The *Valid for next VC.ID and routing direction*, the *Routing direction* and the *Valid* function the same as the ones in figure 3.12. The *Group member* indicates if a router is a member of the group stored in this record. The *Tail* indicates if the node is the last member in the group. The *VC.ID* is the virtual channel ID that the flit occupies at this input channel of the router. The *Multi.ID*, the *Xm.Ym*, the *Group.type*, the *Next VC.ID* and the *Next Xn.Yn* are the same as mentioned before.

The second change of this module is that it can process all the types of the flits in figure 3.7. They are handled in the flow shown in figure 3.10 and figure 3.11.

In figure 3.10, when a unicast data head flit arrives, the *Flit type process* will send out the 3-bit control signal “000” to the FSM. This signal will make the FSM enter the routing stage and then goes to the lane allocation stage. This is the same as mentioned in the unicast one.

When a unicast data body, a multicast data body, a unicast data tail or a multicast tail flit used in *Reserve lane* type arrives, the *Flit type process* will send out “001” to make the FSM go to the flit scheduling stage.

When a multicast data head flit arrives, it will not work in the normal way like the unicast data head flit. First it will check the multicast records to
Figure 3.10: Flow of the hit type process module 1

CHAPTER 3. MULTICAST

42
Figure 3.1.1: Flow of the fit type process module 2
CHAPTER 3. MULTICAST

find out the corresponding record according to the group the Multi_ID and
the $X_mY_m$ fields in the flit. Then it will check if the node is the last member
of the group. If so, it will send out \textit{101}, and then the FSM will do the
allocation for sink. Otherwise, it will check the group type in that record.
If it is a Reserve lane type and arrives at the group member, the flit type
process module will send out \textit{010}, and then the FSM will do the scheduling
for next and allocation for sink. If the flit is a Reserve lane type but does
not arrive at the group member, the flit type process module will send out
\textit{001} to make the FSM do the scheduling. If it is not a Reserve lane type
and arrives at the group member, it will replace the $X_nY_n$ field in the flit
with the next $X_nY_n$ field of the record. The module will send out \textit{011} for
the FSM to let it do the routing, the allocation for sink and the allocation for
next to the lane specified by the Next VC_ID field in the record. If it is not
a Reserve lane type and does not arrive at the group member, the module
will send \textit{000} to do the routing and the allocation for next to the specific
lane.

If a setup head flit arrives at the router, the module will check the
\textit{If Reached Tail} field of the flit to distinguish whether it is a setup head
flit or a setup add head flit. If the value is \textit{0}, it is a setup add head flit.
Then the Flit type process will check the records by the Multi_ID and the
$X_mY_m$ fields to find out the corresponding record. If the setup add head
flit reaches the last group member, it will be saved in the buffers inside the
module and made a mark \textit{Reach tail} which will be used later when all the
setup body flits and the setup tail flit arrive. If the setup add head flit does
not reach the last group member and the group is the Reserve lane type,
the module will send \textit{001} to do scheduling and mark \textit{Go directly}. If the
setup add head flit does not reach the last group member and the group is
the Not reserve lane type, it will go the same way as the multicast data head
flit but without the need to sink. If the value of \textit{If Reached Tail} is \textit{1}, the
flit is a setup head flit. The module will check if there is an empty multicast
record. If so and the flit also arrives at the group member, then the flit will
be saved in the buffers, be marked with \textit{Store} and wait for the setup body
and the setup tail flits. If there is an empty record but the flit does not
arrive at the group member, the module will copy the multicast information
to that empty record, send out \textit{000} to the FSM to do routing and mark
\textit{Go directly}. If there is no empty record, the module will send a setup
fail response back to the group master and mark \textit{Delete}. The response flit
will go to the routing stage according to the \textit{000} sent out by the Flit type
process.

In figure 3.11, when the setup body flit or the setup tail flit arrives at
the router, it will operate according to the mark made by the setup head
CHAPTER 3. MULTICAST

45

flit. If the mark is the “Store” or the “Reach tail”, the flit will be saved in the buffers. When all the flits of a setup packet arrive and the mark is the “Store”, the module will copy the multicast information to the record. If the setup packet reaches the last group member, the module will make a tail mark in the record, and send back a setup succeed response back to the group master together with “000” to the FSM to make it enter the routing stage. Because a feedback is one flit but a setup packet is at least three flits, when the Flit type process sends out a feedback flit, a credit plus signal will be sent to the previous router to make up for this difference. If the setup packet does not reach the last member of the group, the module will replace the $X_n Y_n$ field in the flit with the Next $X_n Y_n$ in the record, and then send all the flits of the packet out of the buffers one by one. When the setup head flit is sent out, “000” will be sent to the FSM to do the routing. When the body or the tail flit is sent out, “001” will be sent out to do scheduling. If all the flits of a setup packet arrive and the mark is the “Reach tail”, the setup add packet will turn into a setup packet by setting the If_reached_tail field in the head flit. The module will clear the Tail bit and write the address of the next group member to the Next $X_n Y_n$ field in the multicast record. At the same time it will copy the address of the next group member $X_2 Y_2$ field to the $X_n Y_n$ field in the flit. After this, the module will send out the whole packet in the buffers in the way mentioned before. If the mark is the “Delete”, all the body and the tail flits will be deleted when arriving, and a credit plus signal will be sent to the previous router to make up for this difference. If the mark is the “Go directly”, “001” will be sent to the FSM to make the body or tail flit do the scheduling.

If a setup response flit arrives at a router, the Flit type process will check if it arrives at the group master. If so, the module will find out the record according to the Multi_ID and the $X_m Y_m$, and then clear the Setup sign bit in it which is set when setting up the group. Then if it is a setup succeed response, the flit is deleted and a credit plus signal will be sent back to the previous router. If the response is a setup fail response, the Release sign bit will be set in that record and a release flit will be sent out to release the group. If the group type is the Reserve lane, “100” will be sent to the FSM. Then the flit will do the routing, copying the Next VC_ID from the record as its allocation result and the scheduling. If the group type is Not reserve lane, the FSM will receive “000” and the flit will do the routing and the allocation for next to a specific lane. If the flit does not arrive at the group master, the module will send “000” to the FSM to make the flit do the routing.

If a release flit arrives at a router, the Flit type process will check the records to find out the corresponding record specified by the Multi_ID and
the \(X_m Y_m\), and then clear it. If it reaches the last group member, a release response flit will be sent back to the group master and “000” will be sent to the FSM for the routing. If it does not reach the last group member, the module will process it as a multicast data head flit but without sink.

If a release response flit arrives at the group master, the module will clear the \textit{Release sign} bit in the corresponding record specified by the \multi and the \(X_m Y_m\). Otherwise, the module will send out “000” to the FSM to do the routing.

### 3.3.3 Control path

![Diagram](image)

Figure 3.12: A multicast record in the multicast record module

The \multi module is used to store the information about the multicast groups set up by this node. The number of the records determines the number of the groups that can be set up by each router. One record is used when one new group is set up by the router. If there is no free record, the router can not set up any more. The information is recorded from the setup packets, and used by the other kinds of multicast packets. The format is shown in figure 3.12. The \textit{Valid for next VC.ID and routing direction} are the valid bit for the \textit{Next VC.ID} and the \textit{Routing direction}. It is set to ‘1’ when the \textit{Next VC.ID} and the \textit{Routing direction} are recorded. The \textit{Routing direction} is used to record the routing result of this group. It is stored in an encoding format, e.g. “00” means left. The \textit{Admi VC.ID} is the ID of the lane used by this group in the admission part. The \textit{Valid} means if the record is valid. The \textit{Next VC.ID} is the ID of the lane used by this group at the input channel of the next hop router. The \multi and the \textit{Group.type} are copied from the multicast setup packet to distinguish the group. The \textit{Next X_nY_n} is the address of the next group member. The \textit{Setup sign}, the \textit{release sign} and the \textit{add sign} represent three states of a multicast group: setting up, releasing and adding members, respectively. At most one of them can be ‘1’. The \textit{Setup sign} is ‘1’ if a setup packet is sent
out and the setup response has not come back. The \textit{Release sign} is ‘1’ if a release packet is sent out and the release response has not come back. The \textit{Add sign} is ‘1’ if a setup add packet is sent out and the setup response has not come back.

Figure 3.13: The FSM of the multicast router
CHAPTER 3. MULTICAST

The FSM array is the next changed module. It is extended as shown in figure 3.13. The “000” and “001” are the unicast part, but a counter is added. When a flit wants to go to both the next router and the sink direction, the counter is ‘1’. If the flit only goes to the next router or the sink, the counter is ‘0’. If the counter is ‘1’, the requests are sent to both the next and the sink modules of the allocators or the schedulers. Only when all the two requests are handled, for the next and for the sink, the FSM will go to the arbitration stage. When the flit has been transmitted to all the outputs that it requests, a grant is sent out from the Arbiter and the FSM can handle the next flit. When “100” arrives, the FSM will first go to “S9” to do the routing. When successful, it will send out a copy request to the Allocator next module where the allocation results are saved and then it will go to “S10”. When receiving the request, the Allocator next module will copy the allocation result from the lane specified by the Next VC_ID field in the multicast record to the allocation result of the current lane. When this succeeds, a grant will be sent back to the FSM and then the FSM will go to the next state “S4”. If “010” arrives, the FSM will go to “S5”, the counter is set to ‘1’ and the requests to the Allocator sink and the Scheduler next are sent. If “101” arrives, the FSM will go to “S5”, the counter is set to ‘0’ and the request to Allocator sink is sent. Then when all the conditions are met, the FSM goes to “S7”. If “011” arrives, the FSM goes to “S6”, the counter is set to ‘1’ and the request to do the routing and the allocation for sink are sent out. When all the conditions are met, the FSM goes to “S3”.

In the Allocator next module, some changes are made. The first is about the priority. The multicast packets have the highest priority. The unicast packets that go to the sink have the second highest priority. The lowest one belongs to the unicast packets. This scheme can guarantee the physical bandwidth of multicasting. The second change is to support the VC_ID copy operation. This operation copies the allocation result of one lane to another. The third one is that the ability of processing the tail flits is extended. Some multicast flits need to be tread as the tail flits to release the resources. They are shown in table 3.1. A resource-release flit is treated as a tail flit to clear the using status of the lane which it is heading for.

<table>
<thead>
<tr>
<th>Flit type</th>
<th>Releasing resources</th>
<th>Not releasing resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserve lane</td>
<td>3, 7, 8, 9</td>
<td>1, 2, 4, 5, 6, 10, 11</td>
</tr>
<tr>
<td>Not reserve lane</td>
<td>3, 6, 7, 8, 9</td>
<td>1, 2, 4, 5, 10, 11</td>
</tr>
</tbody>
</table>

Table 3.1: The flits releasing the resources or not

In the Arbiter module, the first change is the priority scheme. It is the
same as the one used in the *Allocator next* module. It is used to guarantee the physical bandwidth of multicasting. The second is that when the multicast setup tail flit succeeds in the arbitration, the arbitration to that direction will keep stopping until the setup is handled by the *Flit type process* module of the next hop router. At that time, the next hop router will send out a restart signal to the *Arbiter* in this router to start it again. A requirement of this mechanism is that the length of the multicast setup packets must be smaller than or equal to the length of the VC. Otherwise, the setup packets can not be completely sent to the next hop router.

### 3.3.4 Admission and sink

Not every type of flits will go through the admission or the sink part. Some of them are generated and deleted within the network. The ones that go through the admission and the sink parts are shown in Table 3.2. So a little change is done to the admission part to support segmenting the multicast packets into the multicast flits. A little change is also done in the sink part to support composing the multicast flits into the multicast packets.

<table>
<thead>
<tr>
<th>Flit type</th>
<th>Admission</th>
<th>Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserve lane</td>
<td>1,2,3,4,5,6,8,10,11</td>
<td>1,2,3,10,11</td>
</tr>
</tbody>
</table>

Table 3.2: The flits going through the admission or the sink

### 3.4 Experimental results

After the design and implementation of the multicast wormhole router, some tests are done to check its performance.

The way of calculating packet latency is different from the unicast one. It is shown in figure 3.14. The latency begins to be recorded before the source queuing buffers, and it stops being recorded when the packets are sent out from the network. There are no essential differences between the both methods.

The first test compares the pure multicast with the pure unicast to show the merits of multicasting. It is done in a linear array. The conditions are:

- Linear array size: 7
- Number of the VCs per PC: 4
Figure 3.14: Packet latency with source queuing time

- Number of the VCs per PC in the admission: 4
- Number of the VCs per PC in the sink: 8
- Length of the VCs: 2
- Number of the flits per packet: 4
- Number of the groups set up by each group master: 4

Figure 3.15: Test in a linear array

In figure 3.15, the group masters are the leftmost and rightmost nodes. Each master sets up 4 groups along one direction. For example the master in the leftmost sets up 4 groups and all the others are the group members. When testing the pure unicast performance, only the leftmost and the rightmost nodes send the unicast packets under the uniform traffic, and all nodes receive. When testing the pure multicast performance, only the group masters send the multicast packets, and all the group members receive.
CHAPTER 3. MULTICAST

The total number of the unicast packets being sent is 6000, and 1000 for multicasting. Since one unicast packet is received by 1 router and one multicast packet is received by 6 routers, the total numbers of the received packets is the same. When one of the routers finishes sending, the test is over. We began to record the test data at the 1000th data path clock cycle. This will avoid the effect of warm-up. The source sending rate varies from 1 to 1/25, 1/30 and 1/40. The result is shown in figure 3.16. The latency of the multicast is a little higher than the unicast, but the throughput is 6 times higher than the unicast. Because the average distances of unicasting and multicasting are same, so the high latency of the multicast is due to the algorithm: only when the multicast flit have allocated or scheduled to all the directions that it requests, it can enter the arbitration stage. The higher throughput is due to that one multicast packet will be sent to six routers not to just one.

![Graphs showing average packet latency and throughput](image)

Figure 3.16: Test results of the pure unicast and the pure multicast in a linear array

Next tests are taken in a mesh to show if the multicast affects the unicast performance. The test conditions are:

- Mesh size: 4 x 4
- Number of the VCs per PC: 4
- Number of the VCs per PC in the admission: 4
CHAPTER 3. MULTICAST

- Number of the VCs per PC in the sink: 4
- Length of the VCs: 2
- Number of the flits per packet: 4
- Number of the groups set up by each group master: 2

The group masters are at the top left corner and the bottom right corner. Each master sets up 2 groups in the path shown in figure 3.17, for example the master at the top left corner sets up 2 groups along the path and all the other nodes in the path are group members. When testing the pure unicast performance, all the nodes in the mesh send the unicast packets under the uniform traffic. When testing the mixed unicast and multicast, the group masters send the unicast packets and the multicast packets in a ratio, and the other nodes send the unicast packets. All multicast packets go along the path and the ratio decides the percentage of the multicast traffic in the network. In each group master, a multicast packet is sent out in every \( x \) packets. The percentage of the multicast traffic is presented in the following formula.

\[
\text{Percentage of multicast} = \frac{\text{Multicast traffic}}{\text{Total traffic}} = \frac{2 \times 6}{16 \times (x - 1) + 14 + 12}
\]

\( 2 \times 6 \) means that there are 2 group masters, and the traffic that one multicast packet introduced is equal to 6 unicast packets traffic. \( 16 \times (x - 1) + 14 + 12 \) means that there are \( 16 \times (x - 1) + 14 \) unicast packets traffic and 12 multicast packets traffic, when every node sends \( x \) packets. A setup delay is tested.
during the warm-up stage. It is the time from sending out the setup packet to receiving the setup response packet.

The total number of the unicast packets to be sent is 16000, and 1000 for multicasting. When one router finishes sending, the test is over. We began to record at 1000th data path clock cycle. The source sending rate varies from 1 to 1/25, 1/30 and 1/40. In each group master, 1 multicast packet is sent in every 8 packets. This will result in 8.7% multicast traffic. The result is shown in figure 3.18. From the figures, the conclusions can be got that the unicast performance is not affected much by the multicast when mixed sending them, and the multicast will bring high throughput for the group members. The high latency of the multicast is due to both the algorithm mentioned in the previous test and the long average distance. The average distance of unicasting is 2.67 (2N/3, N is the number of the rows in meshes, equal to the number of the columns), but of multicasting is 3.5 (Average of 1 to 6). The setup time is not smooth, because the setup packets are sent out and received during the network warm-up stage. If it tested in an empty network, it will become a constant.

![Figure 3.18: Test results of 8.7% multicast traffic in a mesh](image)

The next test is under 16.2% multicast traffic. This traffic resulted from the fact that 1 multicast packet is sent out in every 4 packets. The result is shown in figure 3.19. From the figures, the same conclusions can be got as mentioned in 8.7% traffic.

The next test is under 28.5% multicast traffic. This traffic resulted from the fact that 1 multicast packet is sent out in every 2 packets. The result is
CHAPTER 3. MULTICAST

Figure 3.19: Test results of 16.2% multicast traffic in a mesh

shown in figure 3.20. From the figures, the latency of the multicast and the unicast in the mixed traffic goes high near 0.35 network load. So under 28.5% multicast traffic, the latency of the unicast will be affected by the multicast but the throughput will be still good.

Figure 3.20: Test results of 28.5% multicast traffic in a mesh

The next tests are taken in a mesh under 16.2% multicast traffic to com-
pare the performance of not using and using lane reservation. Test conditions are:

- Mesh size: 4 x 4
- Number of the VCs per PC: 6
- Number of the VCs per PC in sink: 24
- Length of the VCs: 2
- Number of the flits per packet: 4
- Number of the groups set up by each group master: 2

Figure 3.21: Test results of 16.2% multicast traffic without lane reservation in a mesh

From figure 3.21 and 3.22, the latency will improve 4.6 data path clock cycles in average by using lane reservation.

The multicast design is a synthesizable design. It can be elaborated, but when performing optimization in Synopsys Design Compiler, it results in out of 4GB memory, which is the boundary of 32-bit version, even in XG mode.
CHAPTER 3. MULTICAST

Figure 3.22: Test results of 16.2% multicast traffic with lane reservation in a mesh
Chapter 4

Summary

In this project, we have first implemented a unicast wormhole router. It is validated in a mesh network. This is a generic RTL design where the flit data width, the number and the depth of the VCs in a PC, and the number and the depth of the VCs in the admission and the sink are configurable. With constraint on the timing of the control path, the synthesized design using Synopsys design compiler with the medium mapping effort has a gate count of 60K, the maximum control frequency is 152 MHz, the maximum data frequency is 389 MHz (non-constraint path). Since the control frequency must run two times as fast as the data frequency, the data operating frequency is 76 MHz. The VHDL code for the synthesizable design is about 8000 lines.

In order to support multicast, we have developed a multicast protocol. By this protocol, a multicasting procedure decomposed the multicast data communication from the group establishment. Specifically, it consists of the group setup, the data communication and the group release phases. During the establishment, a multicast group can select to reserve the VCs along the multicast path or not. In order to enhance the performance for multicasting, the multicast packets have priority on link arbitration against the unicast packets. This multicast protocol has been implemented in the unicast wormhole router. The routing algorithm used is dimension order XY routing, which routes packets first along X axis, followed by Y axis. This routing algorithm is deadlock free on meshes. Since the multicast path is built beforehand, it allows the turn from Y to X. Consequently it may lead to deadlock. To avoid deadlock, we use virtual channels in a way that the number of the groups passing through a input channel of a router is not allowed to be greater than the number of the VCs. In such a case, there is always at least one VC available to break the cycle dependency. The resulting wormhole router supports both the unicast and the multicast communication. We have tested it in a linear array and a mesh network.
CHAPTER 4. SUMMARY

4.1 Conclusion

From experimental results, we can draw conclusions as follows:

- Multicasting can be efficiently implemented for the wormhole-switched NoC. Multicasting results in higher throughput while it has minimum impact on the unicast performance. In a mixed unicast and multicast traffic scenario where the multicast takes up less than 20% of the total network traffic, the impact of the multicast on the unicast is negligible if the network is not saturated.

- Multicasting with different QoS can be accomplished. With lane reservation, the performance of the multicast traffic is improved. since it reserves the resources, it has a negative impact on the unicast traffic.

- The overhead for incorporating multicast is high due to the increased complexity in the control path, which is the speed bottleneck in a wormhole unicast router.

4.2 Future work

From this study, we find that the speed bottleneck of a wormhole router is the control path. It can be about 10 times slower than the data path. The overhead for implementing the multicast occurs mostly in the control path as the flit types and the arbitration become more complicated. To make a practical design where high speed is the design goal, the control path must be optimized. The improvements could be done from the following three angles:

- Simplify the algorithm used in the modules.

- Change the coding style. Different coding styles will result in significantly different synthesis results.

- Modify the architecture of the design to make it work more efficiently.

Another direction is to use the multicast to emulate the bus behavior. As there exists a large body of bus applications, making a smooth transition from a bus architecture to a network architecture will be essential. With the QoS multicasting technique, a simplex communication path can be established. A reverse direction communication path may be set up in the same way. As a result, a bidirectional bus can be virtually established. Then they can be used to emulate a bus protocol.
Appendix A

Code structure

The structure of the VHDL files is shown in figure A.1. The files in the dashed-line box should be compiled first, and all the others should be compiled following their hierarchical structure.

The “altera_mf.vhd” and the “altera_mf_components.vhd” are used for the “fifo.vhd”. They are generated from Altera Quartus II 4.1. The “DW-packages.vhd” is a library from Synopsys. The “global_parameters.vhd” defines the global parameters.
Figure A.1: The structure of VHDL files
Appendix B

Power

Figure B.1: Power of the unicast wormhole router under the uniform traffic [4]
Figure B.2: Power of the unicast wormhole router under the locality traffic [4]
Appendix C

Synthesis area results

C.1 Unicast router with decoupled admission

****************************************************
Report: constraint
Design: router
Version: V-2004.06-SP2
Date: Thu Mar 24 19:23:02 2005
****************************************************

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>max.transition</td>
<td>0.00</td>
</tr>
<tr>
<td>max.fanout</td>
<td>0.00</td>
</tr>
<tr>
<td>max.capacitance</td>
<td>0.00</td>
</tr>
</tbody>
</table>

****************************************************
Report: cell
Design: router
Version: V-2004.06-SP2
Date: Thu Mar 24 19:23:03 2005
****************************************************

Attributes:
- b - black box (unknown)
- h - hierarchical
- n - noncombinational
- r - removable
- u - contains unmapped logic

<table>
<thead>
<tr>
<th>Cell</th>
<th>Reference</th>
<th>Library</th>
<th>Area</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>U271</td>
<td>BUFD1</td>
<td>umcl18u250t2.bc</td>
<td>12.197000</td>
<td></td>
</tr>
<tr>
<td>U272</td>
<td>INV1</td>
<td>umcl18u250t2.bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U273</td>
<td>TIEH</td>
<td>umcl18u250t2.bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U274</td>
<td>NOR2M1D1</td>
<td>umcl18u250t2.bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U275</td>
<td>NOR2M1D1</td>
<td>umcl18u250t2.bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U276</td>
<td>NOR2M1D1</td>
<td>umcl18u250t2.bc</td>
<td>16.261999</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C. SYNTHESIS AREA RESULTS

<table>
<thead>
<tr>
<th>U277</th>
<th>NOR2MID1</th>
<th>umcl18u250t2.bc</th>
<th>16.261999</th>
</tr>
</thead>
<tbody>
<tr>
<td>U278</td>
<td>NOR2MID1</td>
<td>umcl18u250t2.bc</td>
<td>16.261999</td>
</tr>
<tr>
<td>U279</td>
<td>NOR2MID1</td>
<td>umcl18u250t2.bc</td>
<td>16.261999</td>
</tr>
<tr>
<td>U280</td>
<td>NOR2MID1</td>
<td>umcl18u250t2.bc</td>
<td>16.261999</td>
</tr>
<tr>
<td>U281</td>
<td>NOR2MID1</td>
<td>umcl18u250t2.bc</td>
<td>16.261999</td>
</tr>
<tr>
<td>U282</td>
<td>INV1</td>
<td>umcl18u250t2.bc</td>
<td>8.131000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>adm.channel</th>
<th>adm.channel</th>
<th>103677.132812</th>
</tr>
</thead>
<tbody>
<tr>
<td>allo.next1</td>
<td>allo.next</td>
<td>126968.664062</td>
</tr>
<tr>
<td>allo.sink1</td>
<td>allo.sink</td>
<td>30345.644531</td>
</tr>
<tr>
<td>arbitration1</td>
<td>arbitration</td>
<td>125354.867188</td>
</tr>
<tr>
<td>clk.data.arb.reg</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>counter.reg</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>counter.reg2</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>counter.reg3</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>counter.reg4</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>counter.reg5</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>counter.reg6</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>counter.reg7</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>counter.reg8</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>cross.bar</td>
<td>cross.bar</td>
<td>30898.876953</td>
</tr>
<tr>
<td>forward.lane</td>
<td>forward.lane</td>
<td>22193.978516</td>
</tr>
<tr>
<td>input.channels.0</td>
<td>input.channel.0</td>
<td>21043.568359</td>
</tr>
<tr>
<td>input.channels.1</td>
<td>input.channel.1</td>
<td>21043.568359</td>
</tr>
<tr>
<td>input.channels.2</td>
<td>input.channel.2</td>
<td>21043.568359</td>
</tr>
<tr>
<td>input.channels.3</td>
<td>input.channel.3</td>
<td>21043.568359</td>
</tr>
<tr>
<td>next.used.clr1_reg2</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>next.used.clr1_reg3</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>next.used.clr1_reg4</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>sch.next1</td>
<td>sch.next</td>
<td>40229.328125</td>
</tr>
<tr>
<td>sch.sink1</td>
<td>sch.sink</td>
<td>8452.351562</td>
</tr>
<tr>
<td>sink.channel1</td>
<td>sink.channel</td>
<td>87589.515625</td>
</tr>
<tr>
<td>sink_lane</td>
<td>sink_lane</td>
<td>3642.781738</td>
</tr>
<tr>
<td>sink_used.clr_ctrl2</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>sink_used.clr_ctrl3</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>sink_used.clr_ctrl4</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>sink_used.clr_ctrl5</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
</tbody>
</table>

Total 43 cells 665076.375000
C.2 Unicast router with coupled admission

******************************************************
Report : constraint
Design : router
Version: V-2004.06-SP2
Date : Thu Mar 24 19:06:22 2005
******************************************************

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_transition</td>
<td>0.00 (MET)</td>
</tr>
<tr>
<td>max_fanout</td>
<td>0.00 (MET)</td>
</tr>
<tr>
<td>max_capacitance</td>
<td>0.00 (MET)</td>
</tr>
</tbody>
</table>

******************************************************
Report : cell
Design : router
Version: V-2004.06-SP2
Date : Thu Mar 24 19:06:23 2005
******************************************************

Attributes:
- b – black box (unknown)
- h – hierarchical
- n – noncombinational
- r – removable
- u – contains unmapped logic

<table>
<thead>
<tr>
<th>Cell</th>
<th>Reference</th>
<th>Library</th>
<th>Area</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>U272</td>
<td>BUFD1</td>
<td>umc18u250t2_bc</td>
<td>12.197000</td>
<td></td>
</tr>
<tr>
<td>U272</td>
<td>INVD1</td>
<td>umc18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U273</td>
<td>TIEHI</td>
<td>umc18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U274</td>
<td>TIELO</td>
<td>umc18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U275</td>
<td>NOR2MID1</td>
<td>umc18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U276</td>
<td>NOR2MID1</td>
<td>umc18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U277</td>
<td>NOR2MID1</td>
<td>umc18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U278</td>
<td>NOR2MID1</td>
<td>umc18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U279</td>
<td>NOR2MID1</td>
<td>umc18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U280</td>
<td>NOR2MID1</td>
<td>umc18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U281</td>
<td>NOR2MID1</td>
<td>umc18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U282</td>
<td>NOR2MID1</td>
<td>umc18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U283</td>
<td>INVD1</td>
<td>umc18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>admิ.channel</td>
<td>admỉ.channel</td>
<td>umc18u250t2.bc</td>
<td>97005.468750</td>
<td>b, h, n</td>
</tr>
<tr>
<td>allo.next1</td>
<td>allo.next1</td>
<td>umc18u250t2.bc</td>
<td>126968.664062</td>
<td>h, n</td>
</tr>
<tr>
<td>allo.sink1</td>
<td>allo.sink1</td>
<td>umc18u250t2.bc</td>
<td>30345.644531</td>
<td>h, n</td>
</tr>
<tr>
<td>arbitration</td>
<td>arbitration</td>
<td>umc18u250t2.bc</td>
<td>105900.937500</td>
<td>h, n</td>
</tr>
<tr>
<td>clk.data.arb.reg</td>
<td>DFFRNQ1</td>
<td>umc18u250t2.bc</td>
<td>81.311996</td>
<td>n</td>
</tr>
<tr>
<td>counter_reg</td>
<td>DFFRPQ1</td>
<td>umc18u250t2.bc</td>
<td>81.311996</td>
<td>n</td>
</tr>
<tr>
<td>counter_reg2</td>
<td>DFFRPQ1</td>
<td>umc18u250t2.bc</td>
<td>81.311996</td>
<td>n</td>
</tr>
<tr>
<td>counter_reg3</td>
<td>DFFRPQ1</td>
<td>umc18u250t2.bc</td>
<td>81.311996</td>
<td>n</td>
</tr>
<tr>
<td>counter_reg4</td>
<td>DFFRPQ1</td>
<td>umc18u250t2.bc</td>
<td>81.311996</td>
<td>n</td>
</tr>
<tr>
<td>Component</td>
<td>Type</td>
<td>Package</td>
<td>Area (µm²)</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
<td>--------------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>counter_reg5</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>counter_reg6</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>counter_reg7</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>counter_reg8</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>cross_bar1</td>
<td>cross bar</td>
<td>18035.246094</td>
<td></td>
<td>h</td>
</tr>
<tr>
<td>forward_lane1</td>
<td>forward_lane</td>
<td>22193.978516</td>
<td></td>
<td>h, n</td>
</tr>
<tr>
<td>input_channels.0</td>
<td>input_channel.0</td>
<td>21043.568359</td>
<td></td>
<td>b, h, n</td>
</tr>
<tr>
<td>input_channels.1</td>
<td>input_channel.1</td>
<td>21043.568359</td>
<td></td>
<td>b, h, n</td>
</tr>
<tr>
<td>input_channels.2</td>
<td>input_channel.2</td>
<td>21043.568359</td>
<td></td>
<td>b, h, n</td>
</tr>
<tr>
<td>input_channels.3</td>
<td>input_channel.3</td>
<td>21043.568359</td>
<td></td>
<td>b, h, n</td>
</tr>
<tr>
<td>next_used_clr1.reg2</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>next_used_clr1.reg3</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>next_used_clr1.reg4</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>next_used_clr1.reg5</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>sch_next1</td>
<td>sch_next</td>
<td>40229.328126</td>
<td></td>
<td>h, n</td>
</tr>
<tr>
<td>sch_sink1</td>
<td>sch_sink</td>
<td>8452.351562</td>
<td></td>
<td>h, n</td>
</tr>
<tr>
<td>sink_channel1</td>
<td>sink_channel</td>
<td>87589.515625</td>
<td></td>
<td>h, n</td>
</tr>
<tr>
<td>sink_lane1</td>
<td>sink_lane</td>
<td>3642.781738</td>
<td></td>
<td>h, n</td>
</tr>
<tr>
<td>sink_used_clr_ctrl2</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>sink_used_clr_ctrl3</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>sink_used_clr_ctrl4</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
<tr>
<td>sink_used_clr_ctrl5</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td></td>
</tr>
</tbody>
</table>

Total 44 cells 626095.312500
C.3 Unicast router with decoupled admission optimized for timing

******************************************************************************
Report : cell
Design : router
Version: V-2004.06-SP2
Date : Sun May 1 12:42:26 2005
******************************************************************************

Attributes:
  b – black box (unknown)
  h – hierarchical
  n – noncombinational
  r – removable
  u – contains unmapped logic

<table>
<thead>
<tr>
<th>Cell</th>
<th>Reference</th>
<th>Library</th>
<th>Area</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>U271</td>
<td>BUF1</td>
<td>ucl18u250t2_bc</td>
<td>12.197000</td>
<td>b, h, n</td>
</tr>
<tr>
<td>U272</td>
<td>INVD1</td>
<td>ucl18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U273</td>
<td>TIENI</td>
<td>ucl18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U274</td>
<td>NORGNI1</td>
<td>ucl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U275</td>
<td>NORGNI1</td>
<td>ucl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U276</td>
<td>NORGNI1</td>
<td>ucl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U277</td>
<td>NORGNI1</td>
<td>ucl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U278</td>
<td>NORGNI1</td>
<td>ucl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U279</td>
<td>NORGNI1</td>
<td>ucl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U280</td>
<td>NORGNI1</td>
<td>ucl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U281</td>
<td>NORGNI1</td>
<td>ucl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U282</td>
<td>INV1</td>
<td>ucl18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
</tbody>
</table>

admi.channell, allo.next1, allo.sink1, arbitration1, clk.data.arb_reg, counter_reg, counter_reg2, counter_reg3, counter_reg4, counter_reg5, counter_reg6, counter_reg7, counter_reg8, cross.bar, forward.lane, input.channels.0, input.channels.1, input.channels.2

30898.876953 22193.978516 20143.568359 20143.568359 20143.568359
<table>
<thead>
<tr>
<th>input_channels.3</th>
<th>input_channel.3</th>
<th>21043.568359</th>
</tr>
</thead>
<tbody>
<tr>
<td>next_used_clr1.reg2</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>next_used_clr1.reg3</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>next_used_clr1.reg4</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>next_used_clr1.reg &lt;3&gt;</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>sch.next1</td>
<td>sch.next</td>
<td>40229.328125</td>
</tr>
<tr>
<td>sch.sink1</td>
<td>sch.sink</td>
<td>8452.351562</td>
</tr>
<tr>
<td>sink.channel1</td>
<td>sink.channel</td>
<td>87589.515625</td>
</tr>
<tr>
<td>sink.lane1</td>
<td>sink.lane</td>
<td>3642.781738</td>
</tr>
<tr>
<td>sink_used_clr_ctrl.reg2</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>sink_used_clr_ctrl.reg3</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>sink_used_clr_ctrl.reg4</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
<tr>
<td>sink_used_clr_ctrl.reg &lt;3&gt;</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2.bc</td>
</tr>
</tbody>
</table>

Total 43 cells 742333.937500
C.4 Unicast router with coupled admission optimized for timing

******************************************************************************
Report : cell
Design : router
Version : V-2004.06-SP2
Date : Sun May 1 22:04:25 2005
******************************************************************************
Attributes:
  b — black box (unknown)
  h — hierarchical
  n — noncombinational
  r — removable
  u — contains unmapped logic

<table>
<thead>
<tr>
<th>Cell</th>
<th>Reference</th>
<th>Library</th>
<th>Area</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>U271</td>
<td>BUFDI</td>
<td>umcl18u250t2_bc</td>
<td>12.197000</td>
<td>b, h, n</td>
</tr>
<tr>
<td>U272</td>
<td>INVDI</td>
<td>umcl18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U273</td>
<td>TIEHI</td>
<td>umcl18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U274</td>
<td>TIELO</td>
<td>umcl18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>U275</td>
<td>NOR2MID1</td>
<td>umcl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U276</td>
<td>NOR2MID1</td>
<td>umcl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U277</td>
<td>NOR2MID1</td>
<td>umcl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U278</td>
<td>NOR2MID1</td>
<td>umcl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U279</td>
<td>NOR2MID1</td>
<td>umcl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U280</td>
<td>NOR2MID1</td>
<td>umcl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U281</td>
<td>NOR2MID1</td>
<td>umcl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U282</td>
<td>NOR2MID1</td>
<td>umcl18u250t2_bc</td>
<td>16.261999</td>
<td></td>
</tr>
<tr>
<td>U283</td>
<td>INVDI</td>
<td>umcl18u250t2_bc</td>
<td>8.131000</td>
<td></td>
</tr>
<tr>
<td>admi.channel1</td>
<td>admi.channel</td>
<td></td>
<td>97005.468750</td>
<td>b, h, n</td>
</tr>
<tr>
<td>allo_next1</td>
<td>allo_next</td>
<td></td>
<td>185330.093750</td>
<td>h, n</td>
</tr>
<tr>
<td>allo_sink1</td>
<td>allo_sink</td>
<td></td>
<td>38033.531250</td>
<td>h, n</td>
</tr>
<tr>
<td>arbitration1</td>
<td>arbitration</td>
<td></td>
<td>135075.171875</td>
<td>h, n</td>
</tr>
<tr>
<td>clk_data_arb_reg</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td>h, n</td>
</tr>
<tr>
<td>counter_reg2</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td>h, n</td>
</tr>
<tr>
<td>counter_reg3</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td>h, n</td>
</tr>
<tr>
<td>counter_reg4</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td>h, n</td>
</tr>
<tr>
<td>counter_reg5</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td>h, n</td>
</tr>
<tr>
<td>counter_reg6</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td>h, n</td>
</tr>
<tr>
<td>counter_reg7</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td>h, n</td>
</tr>
<tr>
<td>counter_reg8</td>
<td>DFFRPQ1</td>
<td>umcl18u250t2_bc</td>
<td>81.311996 n</td>
<td>h, n</td>
</tr>
<tr>
<td>cross_bar1</td>
<td>cross_bar</td>
<td></td>
<td>18035.246094</td>
<td>h</td>
</tr>
<tr>
<td>forward_lane1</td>
<td>forward_lane</td>
<td></td>
<td>22193.978516</td>
<td>h, n</td>
</tr>
<tr>
<td>input_channels.0</td>
<td>input_channel.0</td>
<td></td>
<td>21043.568359</td>
<td>h, n</td>
</tr>
<tr>
<td>input_channels.1</td>
<td>input_channel.1</td>
<td></td>
<td>21043.568359</td>
<td>h, n</td>
</tr>
<tr>
<td>input_channels.2</td>
<td>input_channel.2</td>
<td></td>
<td>21043.568359</td>
<td>h, n</td>
</tr>
</tbody>
</table>
### APPENDIX C. SYNTHESIS AREA RESULTS

<table>
<thead>
<tr>
<th>input_channels_3</th>
<th>input_channel_3</th>
<th>21043.568359</th>
</tr>
</thead>
<tbody>
<tr>
<td>next_used_clr1_reg2</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2_bc 81.311996 n</td>
</tr>
<tr>
<td>next_used_clr1_reg3</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2_bc 81.311996 n</td>
</tr>
<tr>
<td>next_used_clr1_reg4</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2_bc 81.311996 n</td>
</tr>
<tr>
<td>sch_next1</td>
<td>sch_next</td>
<td>40229.328125</td>
</tr>
<tr>
<td>sch_sink1</td>
<td>sch_sink</td>
<td>8452.351562</td>
</tr>
<tr>
<td>sink_channel1</td>
<td>sink_channel</td>
<td>87589.515625</td>
</tr>
<tr>
<td>sink_lane1</td>
<td>sink_lane</td>
<td>3642.781738</td>
</tr>
<tr>
<td>sink_used_clr_ctrl_reg2</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2_bc 81.311996 n</td>
</tr>
<tr>
<td>sink_used_clr_ctrl_reg3</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2_bc 81.311996 n</td>
</tr>
<tr>
<td>sink_used_clr_ctrl_reg4</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2_bc 81.311996 n</td>
</tr>
<tr>
<td>sink_used_clr_ctrl_reg</td>
<td>DFFRFQ1</td>
<td>umcl18u250t2_bc 81.311996 n</td>
</tr>
</tbody>
</table>

**Total 44 cells** 721318.812500
Bibliography


