國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

自時花托連結網路利用一五編碼方式

Self-Timed Torus Interconnection Network with

one-of-five encoding

研 究 生：黃曼珍

指導教授：陳昌居 教授

中 華 民 國 九 十 七 年 六 月

自時花托連結網路利用一五編碼方式

Self-Timed Torus Interconnection Network with one-of-five encoding

研 究 生：黃曼珍　　Student：Man-Chen Huang

指導教授：陳昌居　　Advisor：Chang-Jiu Chen

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

In

Computer Science

June 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

Self-Timed Torus Interconnection Network with one-of-five encoding

Student : Man-Chen Huang **Advisor : Prof. Chang-Jiu Chen**

Department of Computer Science and Information Engineering

National Chial-Tung University

Abstract

Because of the slow signal propagation caused by the high wire loads and resistance, the interconnection of SoC cannot be satisfied by the use of a shared bus. A common method of substitution is using unidirectional, point-to-point connections and multiplexers. The Networks on Chip (NoC), due to their characteristics such as scalability, flexibility, high bandwidth, have been proposed as a suitable approach to meet communication requirements in SoC. However, the global clock of the network may cause lots of problems such as clock skew and the design of global clock architecture.

Hence, Torus topology was selected to suffice the scalable demands in SoC. We implemented the network with self-timed, cut-through routing approach with one-of-five data encoding. In the experiment, we verify the results of transitions in the interconnection network are correct and our system should operate at 63.9MHz. Today, the multi-core systems are produced in SoC; we can use our network to communicate between them and resolve some clock problems efficiently.

Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Motivation

Because the performance of a large SoC composed by several cores depends strongly on its interconnection structure, the traditional global shared bus architecture may not satisfy the new requirements of interconnection network of modern large SoC.

A popular method to replace it is to use unidirectional, point-to-point connections and multiplexers. The Networks on Chip (NoC) have been proposed as a suitable approach to meet communication requirements in SoC due to their characteristics such as scalability, flexibility and high bandwidth. To continue increasing the number and variety of macrocells embedded on a single chip, the modern SoC requires interconnects with greater flexibility system networks.

Most of the current NoCs uses the mesh topology. However, there are no feedback links in the last nodes in every row and column. It may cause some performance degradation for unidirectional connection. Thus the torus topology was selected in our design. Any node in the network can transmit data to any other nodes in the network with a simple unidirectional network.

The global interconnection spans the entire chip which may lead to clock skew problems. Clock management of the torus network is also problematic. Furthermore, interconnect delays and crosstalk are becoming worse because the high wire loads and resistance levels. Using self-timed techniques for torus network on a chip eliminates these problems.

In this thesis, we implemented a self-timed, asynchronous interconnection network with torus topology using a delay-insensitive one-of-five data encoding combined with a return-to-zero signaling protocol for NoCs.

1.2 Asynchronous circuit design

Synchronous circuit design is the major method in digital circuit design. Unfortunately, the synchronous digital circuit designs increasingly lead to bring many problems including clock skew, power consumption of clock, EMI, and the global clock tree distribution, etc. In order to resolve those problems, the asynchronous circuit design may be the answer to address three problems.

1.2.1 Advantages and challenges of asynchronous circuits

In a word, asynchronous circuit is a clock-less digital circuit. Instead of the global clock, the handshaking protocols are used. Compared with the synchronous circuit design, the asynchronous circuit design has some advantages over the synchronous ones :

(1) Average-case performance : the asynchronous circuit does not need to use the global clock to synchronize the action; thus, every component can work at its own speed.

(2) Lower system power requirements : the asynchronous circuit has no clock signal; therefore, there’s no extra power consumption. It can almost consume zero power in the idle state, because almost all components are in idle state.

(3) Modularity : the asynchronous circuit is easier to implement modular design. Every component can be connected via the same communication protocol.

(4) No clock skew : the asynchronous circuit is clock-less and thus it can avoid the clock skew problem.

(5) Lower Electro-Magnetic Interference (Noise) : without the clock distribution, the asynchronous design has low EMI (Electromagnetic Influence).

However, asynchronous circuit still has some problems over synchronous circuit. First, without the clock control, asynchronous circuit needs more control signals, resulting in increasing in area. Second, there are few CAD tools to support asynchronous circuit design and test, and it makes the asynchronous circuit harder and longer to design.

1.2.2 The signaling protocols

There are two types of signaling protocols in the asynchronous circuits, the two-phase and four-phase signaling. The two-phase handshake is shown in Figure 1.1. The falling and rising edge are signals of request and acknowledgement. After an action is completed, the control signal does not need to be reset to zero. When the sender is ready to send the data, the sender changes the request signal (0→1 or 1→0). Then the receiver changes its acknowledged signal (0→1 or 1→0) and receives the data at the same time; thus the handshake is completed.

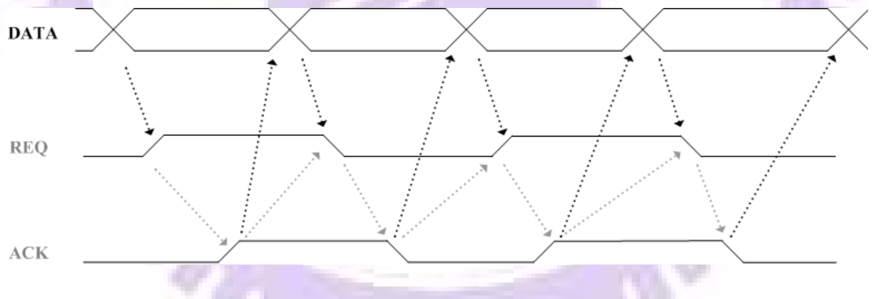


Figure 1.1 The Two-phase handshake protocol

Another type of handshake protocol is four-phase handshaking protocol. It is different from the two-phase protocol; the active signal must be a rising edge. It means that the handshake signal must be reset after an action. The four-phase handshake protocol is shown in Figure 1.2. When the sender is ready to send the data, the sender pulls up the request signal, called Req. Then, the receiver will pull up the acknowledge signal, called Ack, and receives the data at the same time. At the end, all the control signals will be returned to zero, the sender resets the Req signal, and the receiver resets the Ack signal. The handshake is completed.

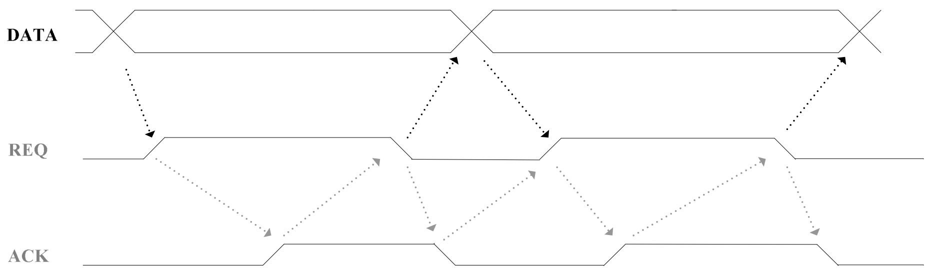


Figure 1.2 The Four-phase handshake protocol

1.2.3 The data encoding protocols

There are several types of data encoding in the asynchronous circuit designs, for examples the bundled data (single rail) [1], dual rail (1-of-2), and 1-of-n (e.g. 1-of-4) [13]. The bundled data protocol encodes data signals the same with synchronous circuits. It separates the control signals (REQ and ACK) from the data signals. Except the bundled data protocol, the DI encoding protocols encode the REQ signal with the data signals. The dual-rail protocol uses two wires to represent one bit of information, and the 1-of-4 protocol uses four wires to represent two bits of information. The 1-of-4 protocol only pulls up one wire signal among the four wires. It means that the 1-of-4 protocol only has half transitions (half power) than the dual rail ones. The 1-of-4 data encoding is shown in Table 1.

Table 1. The 1-of-4 data encoding

Signal	Data
0001	00
0010	01
0100	10
1000	11

1.3 The interconnection network

In a word, the interconnection network is a programmable system, and its components include buffers, channels, switches, and controls that work together to deliver data. It transports data between terminals (processors) as shown in Figure 1.3.

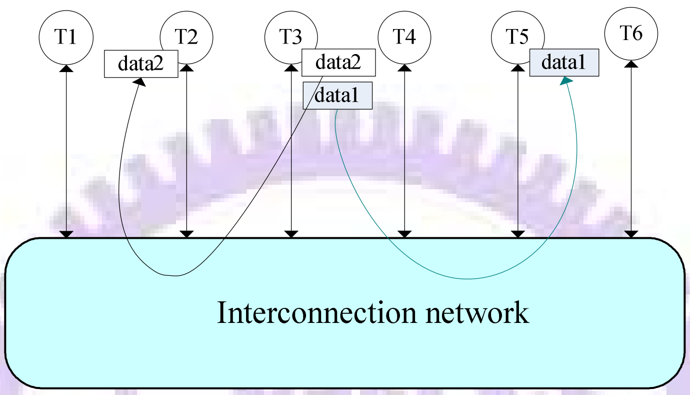


Figure 1.3 The interconnection network

The most common interconnection network applications include computer systems, communication switches, and control systems. During the late 1980s, most of these applications were served by a very simple interconnection networks, such as shared bus, ring, star, stretch ring [9], mesh and torus, etc. Today, the most popular method uses point-to-point interconnection network to replace a traditional shared bus and so on. This trend is due to the performance scaling. The demand for interconnection performance is increasing with processor performance and network bandwidth.

The next several sections give a brief introduction of those interconnection networks including their advantages and disadvantages.

1.3.1 Shared bus

As shown in Figure 1.4, all processors or components are connected to the shared bus. The shared bus only broadcasts data to all processors or components. Thus, any sender transmits data with it, and the data are transmitted to every processor or component. Such networks can tolerate any faults of some processors or components.

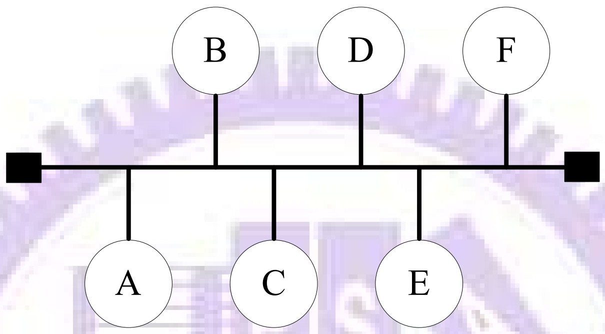


Figure 1.4 Shared bus network

The shared bus architecture has some advantages and disadvantages.

Advantages :

(1) Easy to implement and extend.

(2) Suitable for small networks not requiring very high speed.

(3) Cheaper than other topologies.

(4) Cost effective than others.

Disadvantages :

(1) Limited wire length and number of processors or components.

(2) Maintenance costs may be higher in the long run.

(3) Performance degrades as additional processors are attached or under heavy traffic.

(4) It works best with limited number of nodes.

(5) It may be slower than the other topologies.

(6) If one processor or component is down then the entire network will go down.

1.3.2 Ring

All processors or components are connected to a ring as shown in Figure 1.5. The data can be transmitted along the ring. In addition, the data can only go through them one by one.

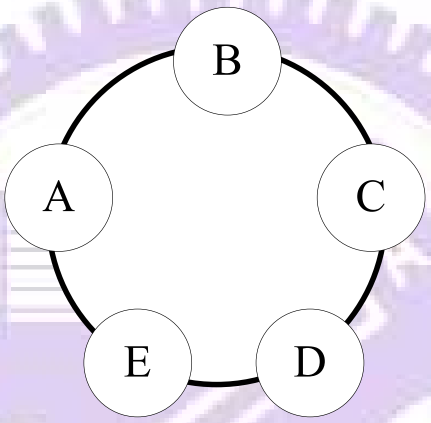


Figure 1.5 Ring network

The followings are the advantages and disadvantages of the ring network.

Advantages :

(1) Very orderly network where every processor or component can access the data or have the opportunity to transmit.

(2) Does not require network controller to manage the connectivity between the processors or components.

Disadvantages :

(1) One malfunctioning processor or bad channel causes problems for the entire network.

(2) To Move, add and change of the processors or components can affect the

network.

1.3.3 Star

The star network is shown in Figure 1.6 which uses the central switched hub to control. A fully connected network built around a switched hub approach. The data from any processor or system is transmitted through the central switched controller to its destination.

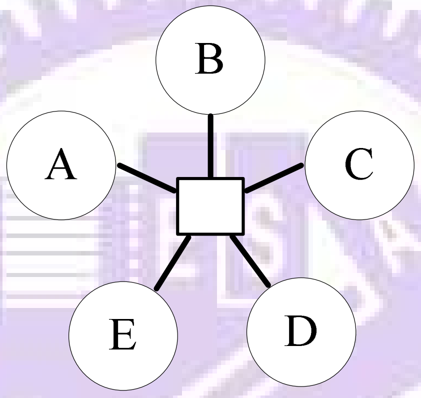


Figure 1.6 Star network

Its advantages and disadvantages are shown below.

Advantages :

(1) Passing of data packet through unnecessary node will be avoided.

(2) Each processor or component is isolated by the link that connects it to the central switched hub. One processor in the network does not affect the others in the network.

(3) The topology is easy to understand and establish.

Disadvantages :

(1) The topology is highly dependent on the functioning of the central switched hub.

(2) The malfunctioning central switched hub causes the entire network fail.

(3) It needs to increase the cost of a central switched hub.

1.3.4 Mesh

The mesh network is shown in Figure 1.7. It is also known as d-dimensional mesh network where d is 2. The 2-dimensional is a simple interconnection network. There are links between any node and its adjacent nodes. The number of its adjacent nodes is limited, usually is 2d.

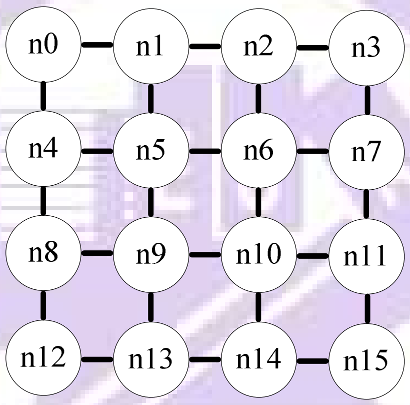


Figure 1.7 Mesh network

The mesh network's advantages and disadvantages are shown as below.

Advantages :

(1) When one path is break off in a node, there are still other transmission paths that can be used.

(2) One node in the network does not affect the others in the network.

Disadvantages :

(1) The cost of this mesh network is expensive.

(2) If the mesh network is unidirectional, the first and the last nodes in every row and column have no direct links. There are no feedback links to transmit data.

1.3.5 Torus

The torus network is very similar to the 2-dimensional mesh network. The different is there are links between the first nodes and the last nodes in every row and column. Every node in the network can send data to others in the network. The torus network will be explained in Chapter 2.

1.4 Improving interconnect performance

After introducing some popular interconnection networks, we can understand that the network selection is very important. To select a suitable, scalable, and high performance network becomes a significant lesson. Furthermore, the global clock design is more difficult in large interconnection network, thus the global asynchronous local synchronous is proposed [12].

Some asynchronous network techniques are discussed in [3][11], using the asynchronous bus network to communication. The throughput can be increased with asynchronous design, but the area is larger oppositely. There are some techniques to increase the performance of the interconnections including :

(1) Repeaters and pipeline latches : A higher throughput can be obtained by using a pipeline latch instead of the amplifier (buffer) /inverter to amplify the signal and spread the link delay over multiple pipeline stages. Splitting the long wire into the narrow pipeline stages improves the throughput with minimal latency.

(2) Datapath width : Unfortunately, adding latches into a wide single-rail datapath may cause problems because of the high load on the latch-enable signal. For a delay-insensitive (DI) encoded datapath, it offers the possibility of operating at a higher frequency by breaking a wide datapath

into a group of narrower datapath.

(3) Delay-Insensitive (DI) interconnects ：Delay-insensitive design methods only obtain the average case delay rather than the worst case. The crosstalk between wires will cause the redundant delays with single rail design, and the single rail design needs some timing analysis. A delay-insensitive design style avoids the need for this timing analysis, its designs that operate correctly whatever the delay in the interconnecting wires.

1.5 Organization of this thesis

The overview of the torus routing chip (TRC) and the system design of TRC will be illustrated in the next chapter. The full designs of the self-timed torus network with one-of-five data encoding will be illustrated in chapter 3. In addition, the experiment results will be represented in chapter 4. Finally, a brief conclusion and future work are discussed in chapter 5.

Chapter 2. Related work

2.1 The torus topology

In recent years, the demands of System on Chip (SoC) increasingly, many cores or processors may be composed on a chip. Thus, the interconnection network architecture is more important. In fact, traditional and hierarchical buses may introduce many new problems on large SoCs. A common alternative is using unidirectional, point-to-point connections and multiplexers. The performance and throughput can be easily improved via using these interconnection networks. Therefore, we must choose a suitable, parallel computing interconnection network to satisfy the requirements of SoC.

There are many topologies of interconnection network, such as shared bus, ring, mesh, torus etc. Among these topologies, the shared bus and ring topology may not be satisfied for the scalable SoC.

Several Networks on Chip (NoC) have been reported in the literature [8]. In our work, we select the torus network to implement. Except above advantages, it also reduces the network diameter; thus it can keep the latency small.

Torus topology is similar to mesh topology; the only difference is that nodes at the edges are connected to the nodes at the opposite edges. Due to the symmetry, torus imposes the same degree for all routers; this makes them homogeneous and the routing algorithm easily to implement.

Thus, the torus topology is a suitable network to suffice the scalable SoC. The next section presents a design using torus topology with self-timed approach.

2.2 Introduction to TRC

A self-timed chip, the torus routing chip (TRC) [15] is proposed in 1986. It uses bundled data encoding to perform cut-through routing [10] in k-ary n-cube multiprocessor interconnection networks. In addition, it uses a new method to avoid deadlock called virtual channels [16]. It provides deadlock-free packet communications in k-ary 2-cube (2-dimensional) network. The TRC can be used to construct concurrent computers with up to 2^{16} nodes in each dimension. It may be a little difficult to distribute a global clock over the network. In order to avoid this problem, the TRC is entirely self-timed.

The TRC uses cut-through routing [10] to replace store-and-forward routing. The method can reduce the latency of communications that traverse more than one channel. It transmits each byte of the packet to the next node when it arrives, instead of reading an entire packet before starting transmission to the next node. Moreover, the communications do not use the memory bandwidth of intermediate nodes. A packet does not interact with the processor or memory of intermediate nodes along its route. In the TRC network, packets are routed and transmitted strictly until they reach their destination.

In order to avoid deadlock problem, the TRC uses virtual channels to execute deadlock-free routing in torus networks. It splits each physical channel into two virtual channels and makes routing dependent on the virtual channel. The details of attestation and method are described in [16].

The next section describes the system design of the TRC.

2.3 System design with TRC

The torus routing chip (TRC) can be used to construct k-ary n-cube interconnection networks. Each TRC routes packets in two dimension, and can be used to construct networks of dimension greater than two. A block diagram of a 4-ary 2-cube TRC is shown in Figure 2.1. Each node consists of a processor, its local memory, and a TRC. Every TRC in the torus is connected to its processor by a processor input channel and output channel. Connections on the edges of the torus wrap around to the opposite edge.

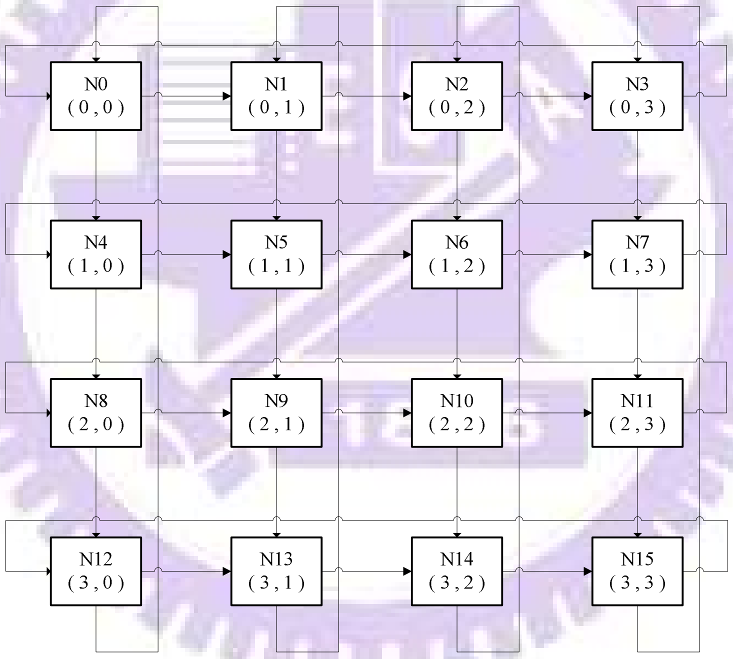


Figure 2.1 A torus system

The TRC uses cut-through routing to transmit a flit (Def. 1). A flit in the TRC is a byte which 8 bits must be transmitted in parallel here. The X and Y channels both consist of 8 data wires and 4 control wire, as known as the bundled data encoding protocol. The 4 control wires are used to separate request/acknowledge signal pairs for each of two virtual channels. The processor channels are also 8 bits wide, but have only two control wires each.

Definition 1. A flow control digit or *flit* is the smallest unit of information that a queue or channel can accept or refuse. Generally a *packet* consists of many flits. Each packet carries its own routing information.

The packet format in TRC begins with two address bytes. The data field of the packet follows the addresses. This field may contain any number of non-zero data bytes. In the end of a packet, the packet is terminated by a zero tail byte. The two address bytes contain the relative X and Y addresses of the destination node. The relative address is a count of the number of channels that must be traversed to reach the destination.

The TRC network's routing rule is, first handling the X dimension, then in the Y dimension. Packets are routed in the dimension of decrementing the relative address at each step. First, when the relative X address is decremented to zero, it means that the packet has reached the correct X-dimension destination. The X address is then stripped from the packet, and the packet begins routing in the Y dimension. When the Y address is decremented to zero, the packet has reached the destination node. The Y address is then stripped from the packet, and the data and tail bytes are transmitted to the node. The transition of the packet is completed. The following algorithm shows the processing of flits arriving at the x-input channel.

// x-input in TRC

```
wait for new flit on x_input
read relative_x_address from x_input
```

```
if relative_x_address is zero then
    drop it
    wait for relative_y_address on x_input
    if relative_y_address is zero then
        drop it
        wait for and read flit_data from x_input
        send flit_data to own processor

        while ( flit_data ≠ tail ) do
            wait for and read flit_data from x_input
            send flit_data to own processor
    else // relative_y_address is not zero
        decrement relative_y_address
        send relative_y_address along y-dimension

        pass all flit_data in the packet arriving on x_input
        along the y-dimension until the tail has been sent
else // relative_x_address is not zero
    decrement relative_x_address
    send relative_x_address along x-dimension

    wait for relative _y_address (may be zero) on the x_input and
    pass it along the x-dimension

    pass all flit_data in the packet arriving at the x_input
    along the x-dimension until the tail has been sent
```

The design of the TRC began in August 1985. A new version TRC in December 1985 can operate at 20 MHz, the input to output delay is 50ns, and throughput is 64 Mbits/s in each dimension.

The full design of the TRC uses self-time, bundled data encoding protocol. It also means that the design must have many timing analyses and constraints. It may cause redundant interconnect delays and reduce the throughput. Moreover, the crosstalk problem is significant with high wire load in bundled data encoding. These problems can be solved efficiently with only need a few timing constraint with the delay-insensitive data encoding.

Thus, we proposed and implemented a self-timed torus interconnection network with one-of-five encoding. The implementation of our design will be described in the next chapter.

Chapter 3. The design of the self-timed torus network with one-of-five encoding

In chapter 1, we have discussed how to improve the interconnect performance. Obviously, the data-path and interconnect method will affect the throughput. Thus, we will redesign a 4-ary self-timed torus interconnection network with one-of-five data encoding. The transmission method of flits in the packet is the same as the torus routing chip (TRC), cut-through routing. However, the data encoding is different from that of original TRC.

The next two sections will illustrate the system and all logic designs of the torus interconnection network with one-of-five encoding.

3.1. The System Design

The torus interconnection network was represented in chapter 2. The overview of system design including the processors (dotted lines) linking in each node of the torus network is shown in Figure 3.1.

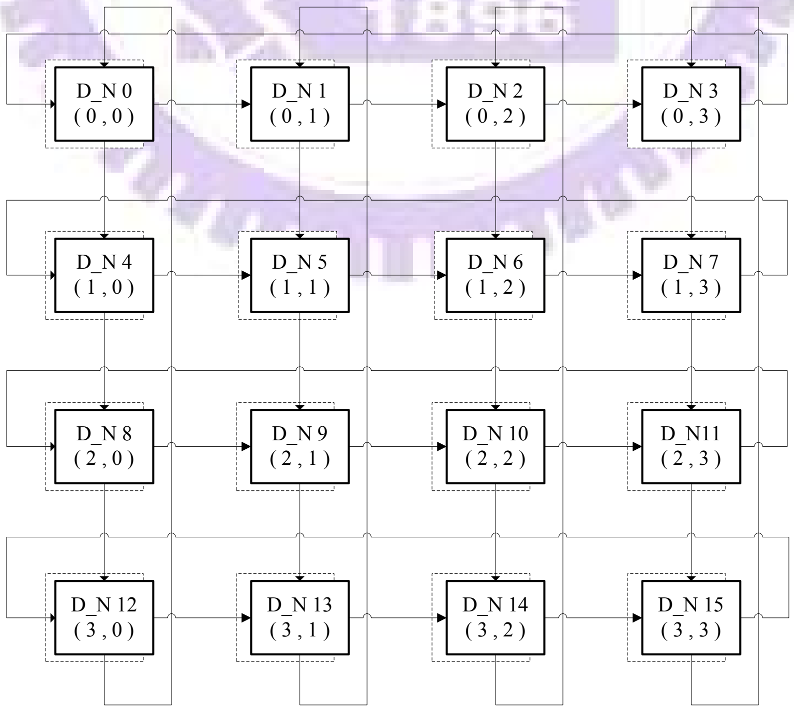


Figure 3.1 The overview of the torus network

An example of a packet transmission flow is shown in Figure 3.2. If the packet0 should be sent from P0 to P14, the transition flow is shown as the gray line. First, the packet0 from P0 is sent to the linking node D_N0, and then D_N0 extracts the relative address and send the packet to the correct output dimension. Thus, the packet flow is D_N0→ D_N1→ D_N2→ D_N6→ D_N10→ D_N14. When it arrives its destination, the node D-N14, it sends the data of packet0 to the interconnect processor P14. The transmission flow is completed.

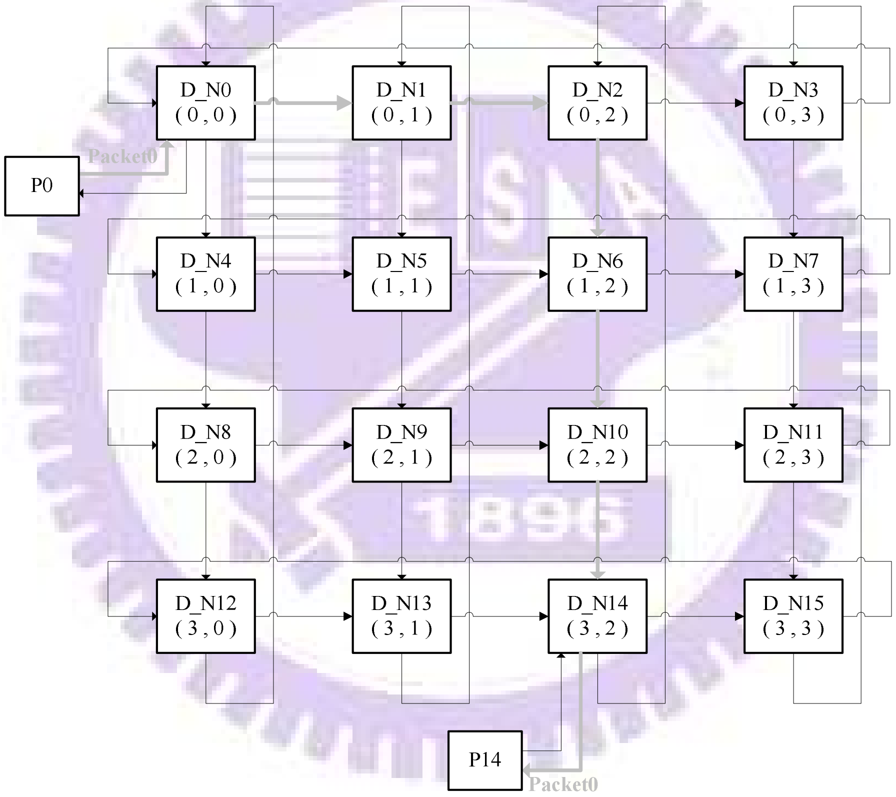


Figure 3.2 Example of the packet flow

In the thesis, we implement a 4-ary 2-cube, a 4 by 4 symmetrical mesh. To match up the one-of-five data encoding, a flit in our design is 5-bits.

The packet format is shown in Figure 3.3. A packet begins with two 5-bits address. The location of the destination node is composed of a 5-bit relative X address and a 5-bit relative Y address. The data of the packet follows after the address. Then, the packet is terminated by a tail EOP (end-of-packet).

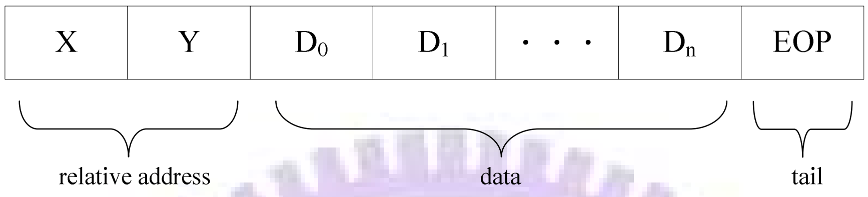


Figure 3.3 Packet format

According to the packet format, our system follows the routing flow represented below :

Step 1 : extracts x and y addresses

Step 2 : constructs the correct path connection according to the addresses

Step 3 : transmit the data through the selected path until the EOP flit is detected

Our design uses a delay-insensitive data encoding combined with a four-phase (return-to-zero) signaling protocol. The connection consists of one acknowledgement wire and five forward-going wires. Table 2 shows this one-of-five data encoding where only one wire in a group is allowed to signal data at any one time (known as one-hot encoding). During normal data transmission, one of the wires (w3, w2, w1, w0) transmission two valid data bits per cycle. The end-of-packet (EOP) signal separates consecutive data transfer. This data encoding requires only a simple five-input OR gate to detect valid data at the receiver.

Table 2. One-of-five data encoding

EOP signal	w3	w2	w1	w0	Information
0	0	0	0	0	Idle state
0	0	0	0	1	Two-bit data value 00
0	0	0	1	0	Two-bit data value 01
0	0	1	0	0	Two-bit data value 10
0	1	0	0	0	Two-bit data value 11
1	0	0	0	0	End of packet

3.2. The Logic Designs

The proposed architectural overview is shown in Figure 3.1. The design of each node will be introduced here. The inner block diagram of a node is shown in Figure 3.4. It consists of three input router blocks, a central switch with arbiters, and three output pipeline latches. There are also two sub-routers in X and P router modules separately. In addition, the pipeline latches are used to replace buffers in the output channels.

The router is responsible for selecting the correct route. When a packet header arrives, the router selects the correct output channel, adjusts the header by decrementing and sometimes stripping the flit, and then passes all flits to the switch until the EOP is detected.

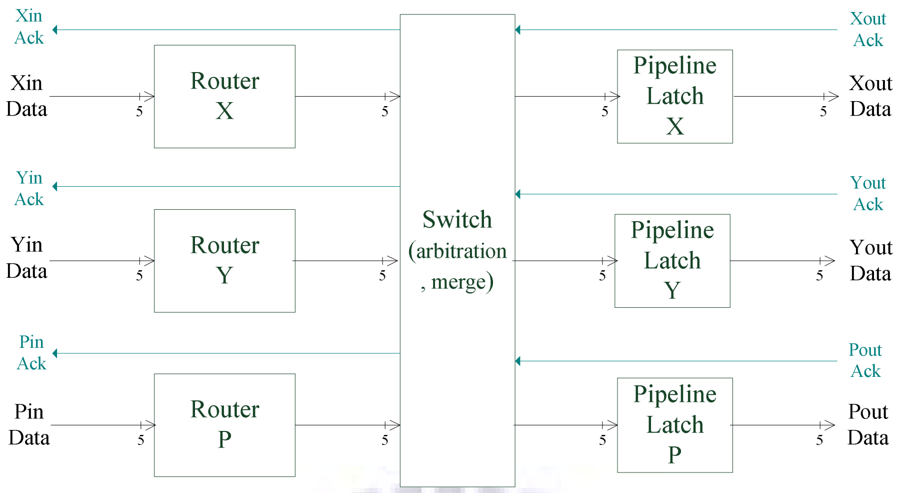


Figure 3.4 The block diagram of D_N

The detailed sub-modules of D_N are shown in Figure 3.5. The data from X-dimension can be transmitted to any output channel; however, the addresses of data from the processor must be detected to prevent the data sent back to itself. Thus, both the module Router X and Router P uses two sub-routers to select the correct output channels. When the data arrive, the router will determine the next state, select the appropriate output channel, and manipulate the current flit.

The switch performs switching and arbitration with arbiters. An arbiter (tree) uses two or three input interlock (mutual-exclusion) element in each output channel; hence only one input channel can use the output channel. If inputs are received simultaneously, the interlock will decide which can be granted first.

After arbitration, some stage pipeline latches are used to replace buffers to store the output data in every output channel. Some logic designs are similar to the DI one-of-four data encoding modules proposed in [6], such as pipeline latch, merge and select.

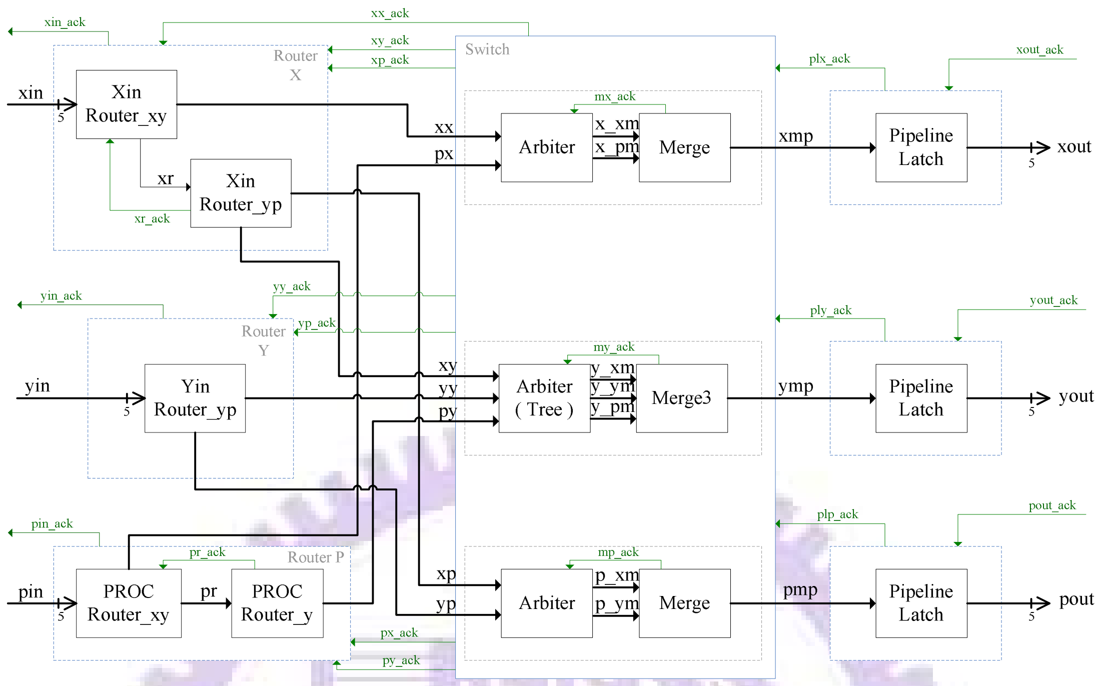


Figure 3.5 The detailed sub-modules of D_N

The finite state machine (FSM) of D_N is shown in Figure 3.6. There are three parallel input dimensions, X, Y and P. If different packets to the node are sent to all three input channels, the data can be processed respectively. First, each router extracts the X or Y addresses and decides whether decrement, strip or pass. Then, it selects the correct output channel to send data. If it wins the arbiter to use the output channel, the data will be transmitted to the selected output pipeline latches. Otherwise, it must wait until it wins the arbitration.

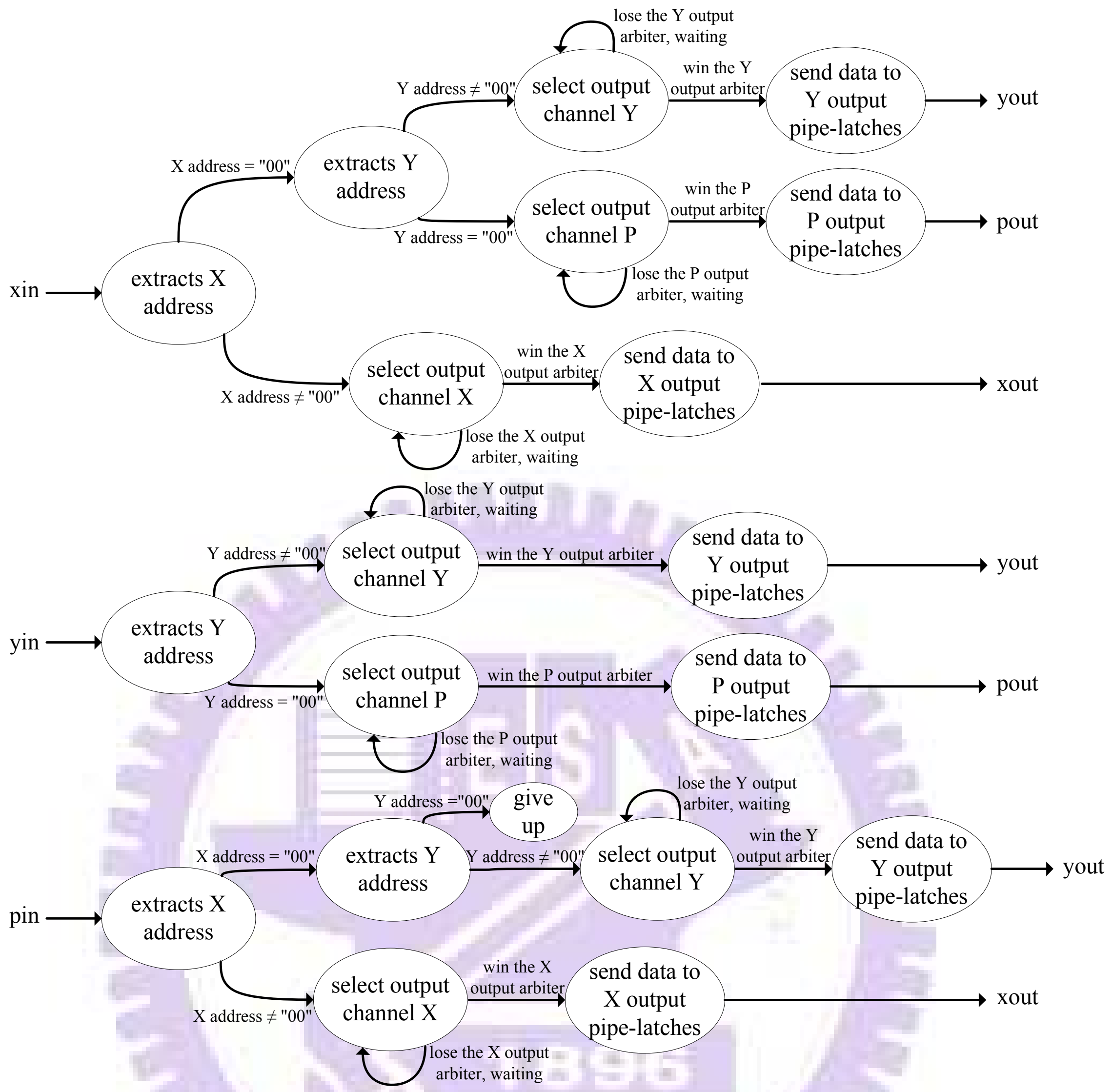


Figure 3.6 The FSM of D_N

3.2.1 The Router

The router module is shown in Figure 3.7. It includes three sub-modules, Set_Dec, Decrement, and Route control. When a flit of packet arrives, the Set_Dec module must decide the current state and the current flit whether to strip, decrement, or pass. According the current state, the decrement may decrease or bypass the current flit. The state and corresponding behavior is shown in Table 3.

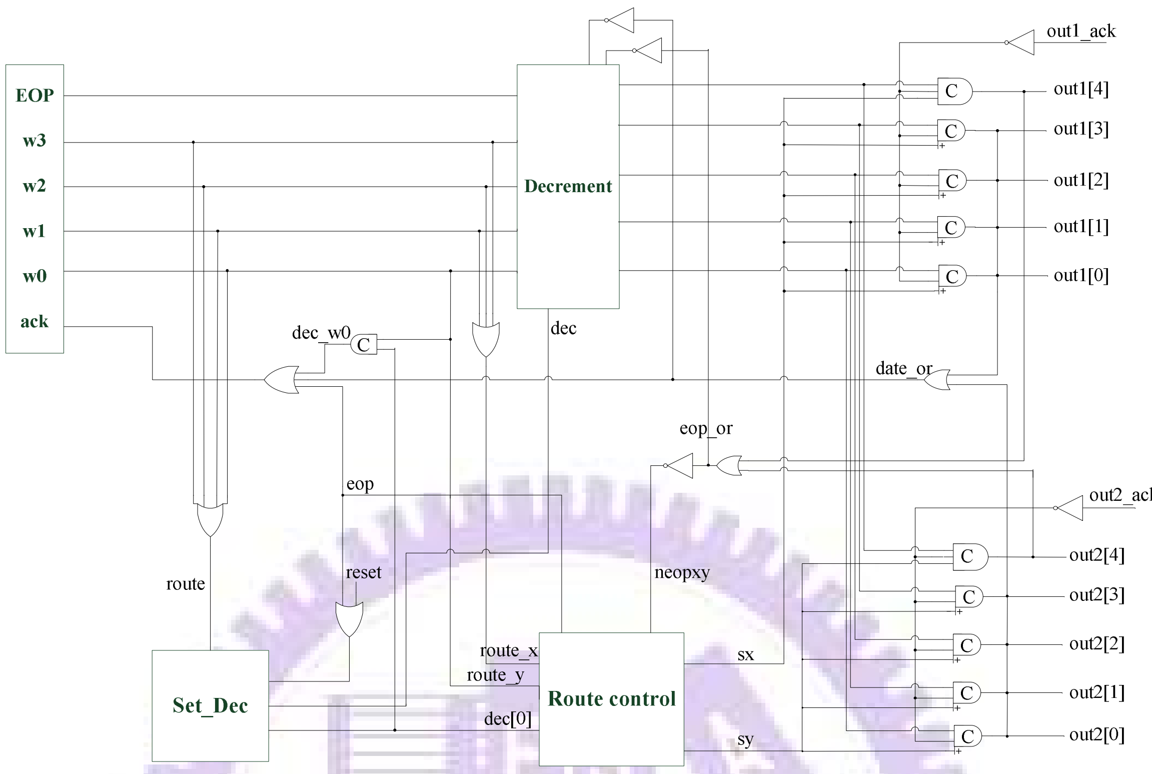


Figure 3.7 A Router

After the operation, the flit is latched in the C-elements and waits the correct output channel from the route control. The route control checks the first flit. If the first flit data values is not ‘00’; it means that the packet has not yet arrived the correct destination in the current dimension. The packet will be transmitted to the same dimension continuously; otherwise, the dimension will be changed.

When the flit is transmitted to the correct dimension by passing through the C-elements, the OR gate detects valid data and sends the *ack* signal to the sender. While the sender receivers the *ack* signal, it will change the flit to the null state and wait the *ack* signal pull down. Then, the sender continuously sends the next flits following the above-mentioned step until the EOP flit.

Table 3. The corresponding behavior of the current state

Current state	Behavior
00	Clear or latch
01	Decrement (shift)
10	Pass
11	× (don't use)

The Set_Dec module is implemented as a finite state machine. The initial state is idle '00'. When the first flit is arrived, the current state is set to '01' and it means that the decrement module decreases the current flit which is represented address value. Then the next state will be latched in the C-elements until the flit returns to zero. When the next flit arrives, the next state will be assigned to the current state, and will set to be '10', the decrement will pass the flit until detect the EOP flit. The Set_Dec will be reset. The truth table of the states and route signal (OR the flit bits *w3* to *w0* represent the data information) is shown in Table 4. In addition, the corresponding circuit is shown in Figure 3.8.

Table 4. The truth table of the route signal, state and next state

route	state (s[1] s[0])	next state (n[1] n[0])
0	00	00
0	01	10
0	10	10
1	00	01
1	01	10
1	10	10

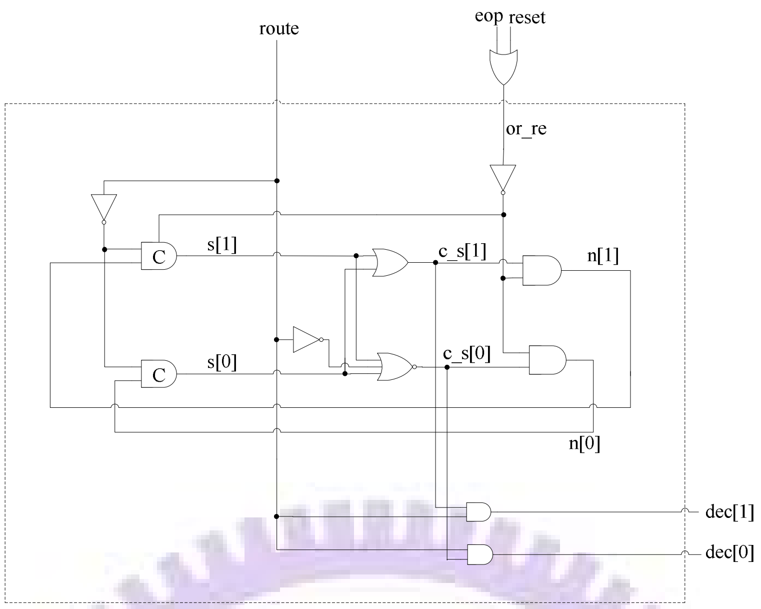


Figure 3.8 Set Dec

After the operation of Set_Dec, the decrement module uses the *dec* signal (2-bits) to process the flit whether decrement or pass as shown in Figure 3.9. Note that the *dec* signals will be reset before the flit return to zero. The asymmetric 3-input C-element is shown in Figure 3.10(a) with its transistor level in (b).

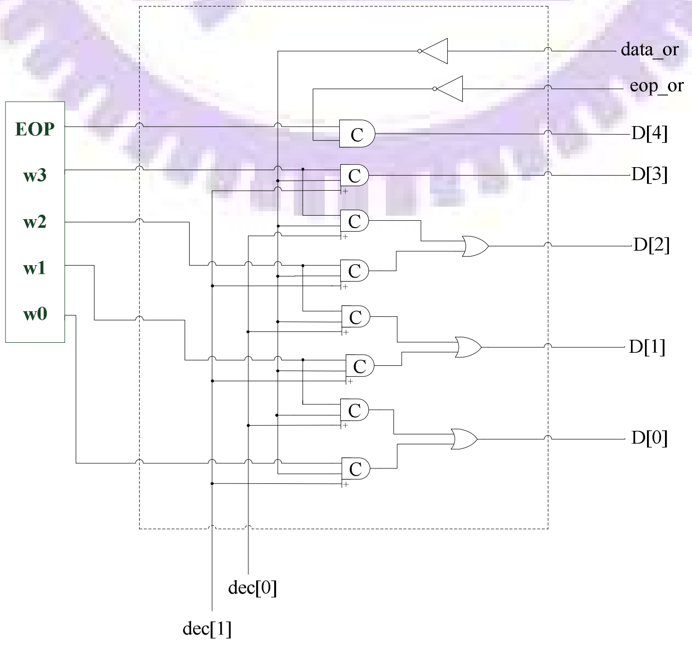
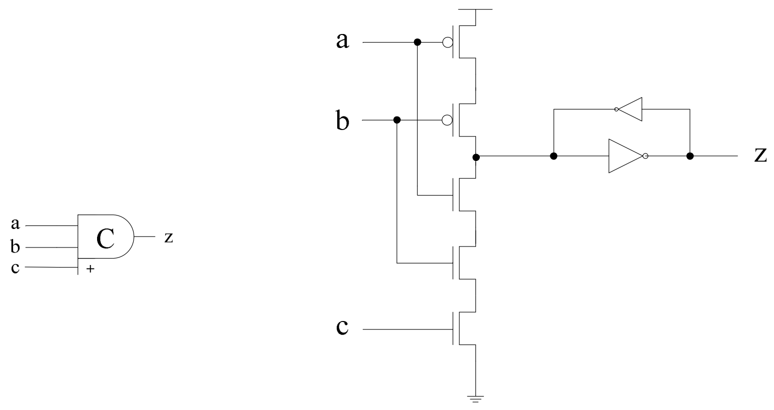


Figure 3.9 Decrement



(a) (b)

Figure 3.10 (a) 3-input asymmetric C-element (b) transistor level

As shown in Figure 3.10, it is a 3-input asymmetric C-element. If we are sure that the c signal is always pulled down before both the *a* and *b* signals are pulled down, then the down signal of input *c* can be ignored. Thus, the asymmetric C-element can save a transistor and increase the circuit speed.

The Route control module shown in Figure 3.11 will use the first flit data information to select the output dimension. The wires from *w3* to *w1* are ORed generate the signal *route_x*. It means that the information of the encoded data is not zero. The *route_y* signal is the *w0* wire; its information of the encoded data is ‘00’. The two signals can be used to decide the correct output dimension. After determining the correct output dimension, the C-elements hold the select control signals *sx* and *sy* until the EOP flit of packet is detected. The C-elements in Figure 3.7 latch the flit and pass it to the correct dimension by the control signal *sx* or *sy*.

While the modules detects the *neopxy* signal from outside, and it means the EOP flit is transmitted to the correct dimension, then the route control module will reset the control signal to wait the next new packet.

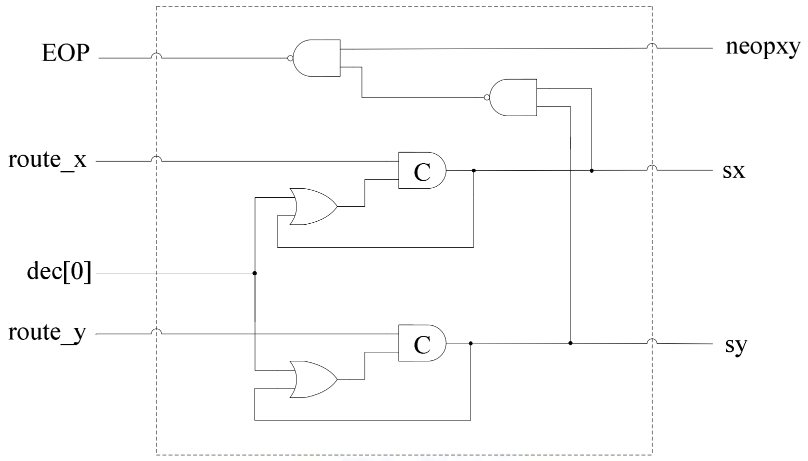


Figure 3.11 Route control

3.2.2 The Arbiter (Tree) and merge

The Arbiter or Arbiter tree modules uses the mutual exclusion circuit to make sure that only one input channel at one time can use the output channel. The mutual exclusion (MUTEX) circuit is shown in Figure 3.12 and the arbitrated Call Block [2][4][5] is shown in Figure 3.13.

The MUTEX circuit can allow only one *ack* signal to be pulled up. If the *R1* and *R2* are pulled up at the same time, the MUTEX circuit will keep the *A1* and *A2* signals low until metastability[1] is resolved.

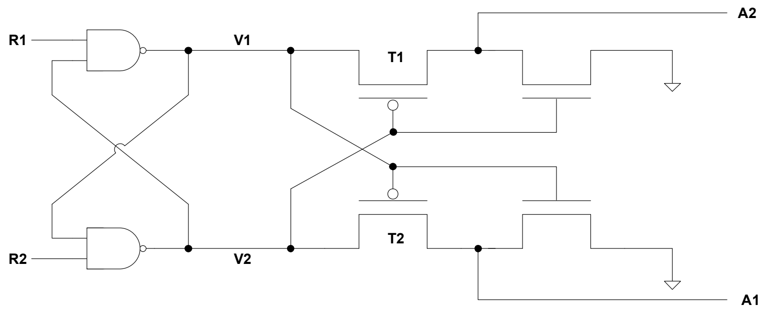


Figure 3.12 Mutual exclusion (MUTEX) circuit

note1. Metastability is the state of a non-equilibrium electronic state to persist for a long period of time. The asynchronous circuit can avoid this situation efficiently.

The arbitrated call block id designed as shown in Figure 3.13 with the MUTEX circuit. If only one request from one of two input channels is produced, the Arbitrated Call Block is granted immediately, and if two are produced simultaneously it will decide which to grant first. The Arbitrated Call Block can deal with two input requests. Thus, the arbiter uses the Arbitrated Call Block to allow only one input channel flit to be transmitted shown in Figure 3.14. It also uses the C-elements in front of the MUTEX to latch the occupied dimension which wins the MUTEX and reset until detects the EOP. After the arbitration, we use OR gate to merge the flits that may come from two input channels to one output channel. We also use the C-elements to latch the data until the receiver is ready.

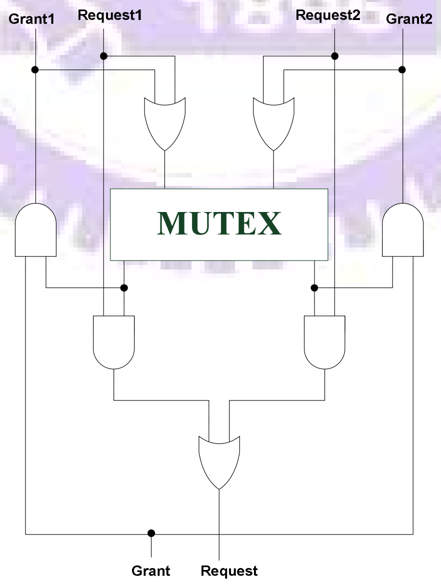


Figure 3.13 Arbitrated Call Block

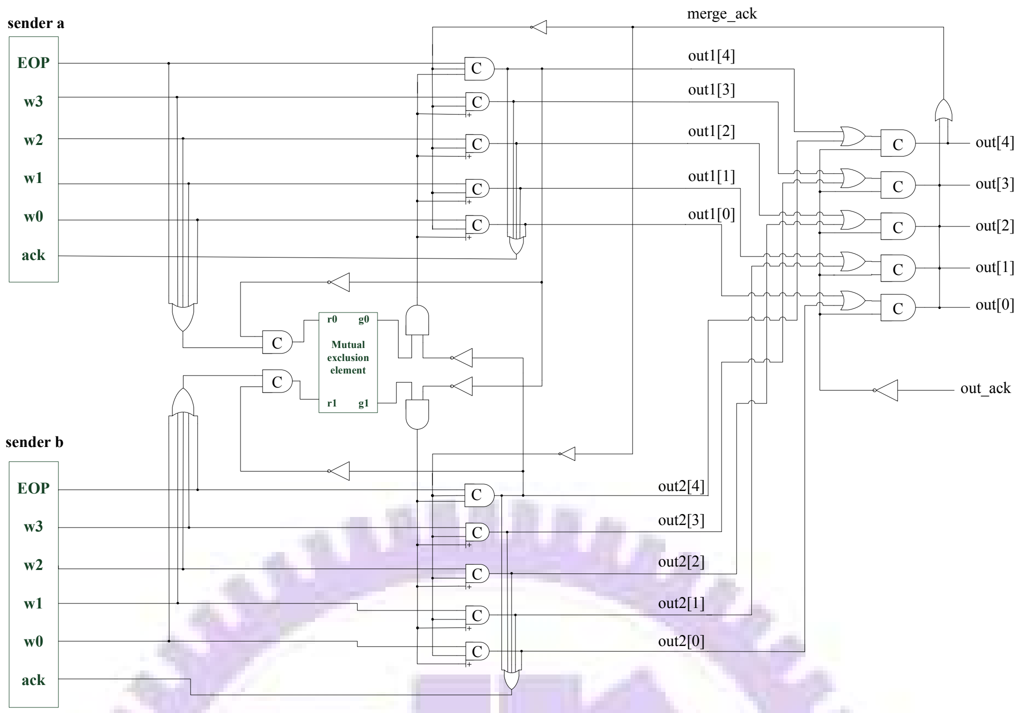


Figure 3.14 An arbiter using MUTEX and merge

Figure 3.15 shows the arbitration tree implemented with some arbitrated call blocks. A method was proposed in [4][5] to solve many initiator requests and allow the bus-bandwidth to be apportioned as required. Our design was shown in Figure 3.5, the data of y-dimension output channel may come from up to two input dimension channels. Thus, the arbiter module in the y-dimension will use the technique with arbiter tree to switch and arbitrate.

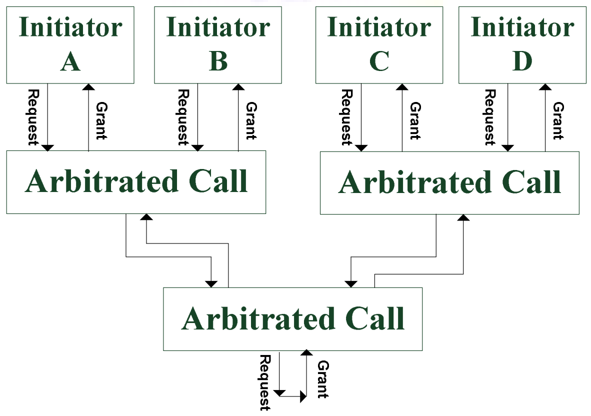


Figure 3.15 Arbitration Tree

3.2.3 The pipeline latch

After arbitration and merge, only one input dimension can use the output channel. Then, there are some stages pipeline latches to buffer the flits behind the merge from the output channel. While the receiver gets the null flit, it will pull down the ack signal. Then, the flit buffered in the pipeline latches waits until it can be transmitted to the output channel. The two stage pipeline latch is shown in Figure 3.16.

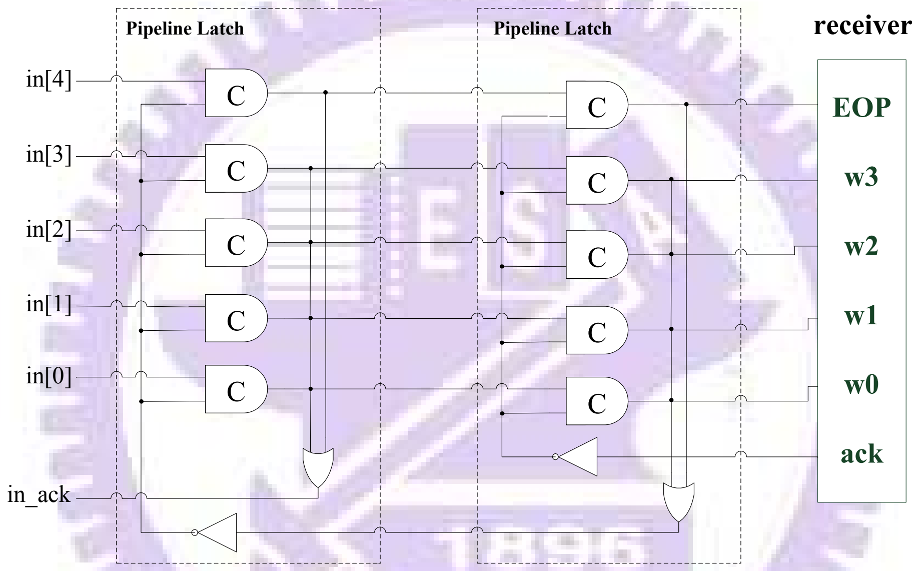


Figure 3.16 Pipeline latch

Chapter 4. Experiment results

In chapter 3, we describe the all logic designs of functional blocks in our system. We use the Verilog HDL to build our system and construct the whole system with above mentioned sub-modules in chapter 3.

We implemented our self-timed torus interconnection network with 1-of-5 encoding in gate-level. The design was synthesized and simulated with the TSMC130 nm library. The simulator used is Modelsim 6.0.

The Modelsim software is used to verify the correctness of our system. In order to verify the correctness, we give the variable length of flits in each packet to every processor at the same time. Each processor sends three or four packets; total count of packets is 100. Figure 4.1 shows the wave form of the whole system simulation. In the experiment, we verify the results of all flits of packets transmitting to the correct destination processor.

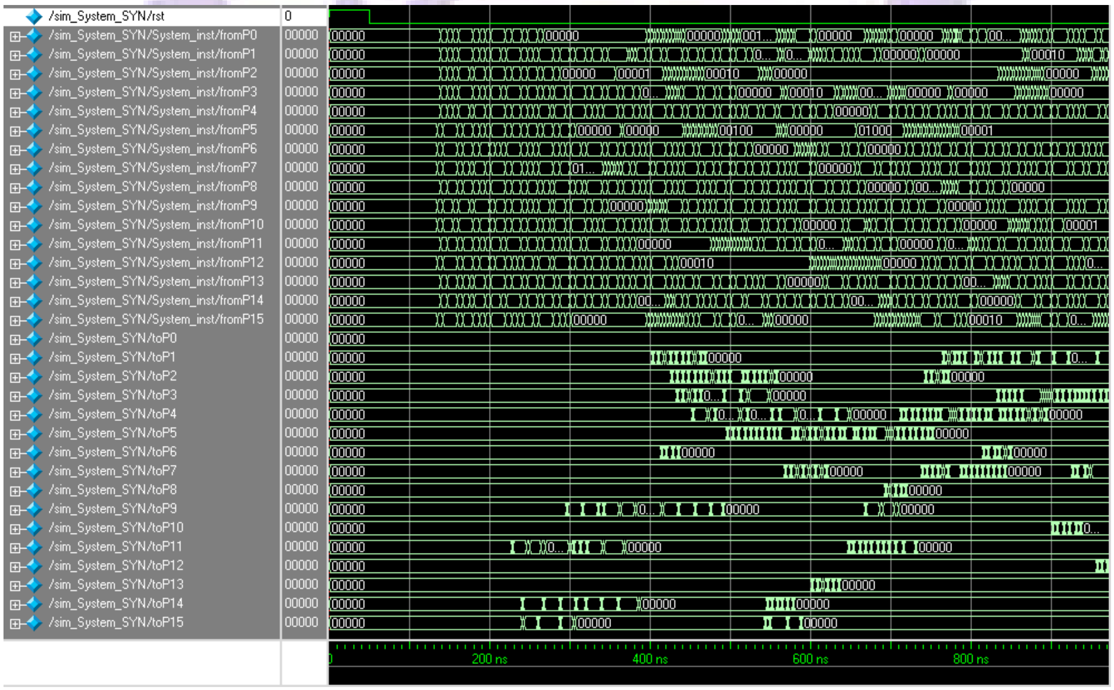

/sim_System_SYN/rst
/sim_System_SYN/System_inst/fromP0
/sim_System_SYN/System_inst/fromP1
/sim_System_SYN/System_inst/fromP2
/sim_System_SYN/System_inst/fromP3
/sim_System_SYN/System_inst/fromP4
/sim_System_SYN/System_inst/fromP5
/sim_System_SYN/System_inst/fromP6
/sim_System_SYN/System_inst/fromP7
/sim_System_SYN/System_inst/fromP8
/sim_System_SYN/System_inst/fromP9
/sim_System_SYN/System_inst/fromP10
/sim_System_SYN/System_inst/fromP11
/sim_System_SYN/System_inst/fromP12
/sim_System_SYN/System_inst/fromP13
/sim_System_SYN/System_inst/fromP14
/sim_System_SYN/System_inst/fromP15
/sim_System_SYN/toP0
/sim_System_SYN/toP1
/sim_System_SYN/toP2
/sim_System_SYN/toP3
/sim_System_SYN/toP4
/sim_System_SYN/toP5
/sim_System_SYN/toP6
/sim_System_SYN/toP7
/sim_System_SYN/toP8
/sim_System_SYN/toP9
/sim_System_SYN/toP10
/sim_System_SYN/toP11
/sim_System_SYN/toP12
/sim_System_SYN/toP13
/sim_System_SYN/toP14
/sim_System_SYN/toP15
200 ns
400 ns
600 ns
800 ns

Figure 4.1 System simulation

4.1. Performance

There are 100 packets in our experiment transmitted from 16 processors in the network. Each processor sends 3 or 4 packets with random length of data and transmits packets to the random destinations. The transition paths of all packets are not fixed which may be the shortest path, one, or the longest path, six. According to the experiment results, the start time of the transition of the first flit sent from random processor to our system is 134.86 ns, and the end time of the transition of the latest flit sent from our system to the random processor is 1700.7 ns. The all transition time of all 100 packets is 1565.84 ns. Thus, as the result of calculate our system should operate at 63.9MHz.

4.2. Area

The area and gate count of the sub-modules and DI-encoding TRC are shown in Table 5. Our torus network only includes the interconnection network area, without the processors. Because the Router module includes some sub-modules and several C-elements, its area is larger. The circuit of Arbiter Tree module uses many gates to insure and determine that the circuit can allow only one input channel to use the output channel. Moreover, our network uses several stages of Pipeline latch to buffer the flits by C-elements, it results in the area of Pipeline latch module is larger.

Table 5. The area and gate count of the modules

module	area (μm^2)	gate count
Router	1275	250
Arbiter/Tree	683/1193	134/234
Merge	290	57
Pipeline latch	7619	1494
Our torus network (without processors)	543048	106480

Chapter 5. Conclusions and future work

In this thesis, we implement an asynchronous, self-timed interconnection network with torus topology. Our system uses the cut-through routing and the one-of-five DI encoding that can improve the interconnect performance efficiently. Moreover, we use the asynchronous circuits to solve the problems that may arise with the global clock. Therefore, the torus topology network has very good scalability, it can suffice the demands of SoC interconnect increasingly.

There are some important reasons using a torus network to implement our interconnect network. First, the torus is easier to implement. If the number of the interconnected processors increases, the torus network can be easily extended. Second, the torus distributes loads to the communication channels more.

Unfortunately, there is a problem in our design, the deadlock. Our torus network uses cut-through routing; it will occupy one channel until the transmission is finished. In order to increase the throughput, the problem is hardly to avoid. However, this problem must be resolved. There are some methods that can be adopted. First, the virtual channels can be implemented with our design. However, the technique has some constraints, such as limited packets size and a little higher cost. Second, we can add the control signals in each row and column to prevent the deadlock. Although the method can solve the deadlock problem, the performance may decrease obviously.

In the experiment, the simulation result of our self-timed interconnection network is proved and the performance is 63.9 MHz. We use a new encoding method, 1-of-5 encoding, to design the asynchronous torus interconnection network. In the future, the technique can be used in the global asynchronous local synchronous (GALS) system to resolve the communication problem between many synchronous modules which each has own clock rate.

Reference

[1] A.M.G Peeters, “*Single-Rail Handshake Circuits*,” doctoral dissertation, Technische Universiteit Eindhoven, Eindhoven, Netherlands, 1996.

[2] Bainbridge, W.J., A. Bardsley, D.M. Clark et al., “*AMULET3i an Asynchronous System-on-Chip*, ” Proc. 6th IEEE Int’l Symp. Asynchronous Circuits and Systems (Async), IEEE CS Press, Los Alamitos, Calif. , 2000, pp.162-175.

[3] Bainbridge, W.J., *Asynchronous System-on-Chip Interconnect*, Springer, Heidelberg, Germany, 2002.

[4] Bainbridge, W.J., Furber, S.B., “*Asynchronous Macrocell Interconnect using MARBLE*,” Proc. Async’98, San Diego, (April 1998) pp.122-132

[5] Bainbridge, W.J., Furber, S.B., “*MARBLE: An asynchronous on-chip macrocell bus*,” Microprocessors and Microsystems, (July 2000)

[6] Bainbridge, W.J., Furber, S.B., “*Delay Insensitive System-on-Chip Interconnect using 1-of-4 Data Encoding*,” Proc. 7th IEEE Int’l Symp. Asynchronous Circuits and Systems (Async), IEEE CS Press, Los Alamitos, Calif., 2001, pp.118-126.

[7] F. Moraes and Al, "*Hermes: an infrastructure for low area overhead packet switch NoC*," Integration VLSI J. vol. 38, no. 1, pp. 69-93, Oct. 2004.

[8] H. Wang, P.S. Peh, and S. Malik, "*A technology aware and energy oriented topology exploration for on-chip networks*," in DATE'05, vol. 2, pp 1238-1243, Mars 2005.

[9] IBM Corporation, CoreConnect Bus Architecture, product brief.

[10] Kermani P, Kleinrlck L (1979), “*Virtual cut-through: a new computer communication switching technique*,” Computer Networks 3: 267-286.

[11] Molina, P.A., “*The Design of a Delay-Insensitive Bus Architecture using Handshake Circuits*,” Ph.D. theses, Imperial College of Science, Technology and

Medicine, University of London, UK, (1997).

[12] S. Moore, G. Taylor, R. Mullins, P. Robinson , “*Point to Point GALS interconnect*s,” Proc. 8th IEEE Int’l Symp. Asynchronous Circuits and Systems (Async), IEEE CS Press, Los Alamitos, Calif., 2002, pp.69-75.

[13] T. Verhoeff, “*Delay-Insensitive Codes – An Overview,*” Distributed Computing, vol. 3, no. 1, 1988, pp. 1-8.

[14] W. Dally and B. Towles, “*Principles and practices of the interconnections networks,*” Morgan Kaufmann, 2005.

[15] W. J. Dally and C. L. Seitz. “*The torus routing chip,*” J. of Distributed Computing, 1(3):187{196, 1986.

[16] W.J. Dally, and C.L. Seitz, “*Deadlock-free message in multiprocessor interconnection networks,*” IEEE Trans. on computers, vol. 36, no. 5, May 1987, pp. 538-546.