Architecture and Design of OASIS NoC with Short-Path Link (SPL)

Takahiro Uesaka

2011/1/26

Contents

1 Introduction

Continuous scaling of CMOS technology makes it possible to integrate a large number of heterogeneous devices that need to communicate efficiently on a single chip. Large-scale integration of these diverse blocks calls for truly scalable communication architectures. While the legacy bus-based communication architecture is the standard solution for on-chip communication, its poor scalability, both in terms of performance and power efficiency, makes it an undesirable choice for future SoC [4].

Many communication architectures have been proposed for connecting multiple on-chip resources, such as Network-on-Chip (NoC). NoC has been introduced as a new interconnection architecture that may be able to integrate a large number of cores while maintaining a high communication bandwidth [7] [9]. In addition, NoC is a scalable architecture, which has a big possibility to remove increasing complexity. The basic idea of NoC is that processors are connected via a packet switched communication, which is similar to the way computers are connected to internet. The packet-switched network routes information between network clients (e.g. processors, memories, and custom logic devices).

Packet switching supports asynchronous transfer of information. It also provides extremely high bandwidth by distributing the propagation delay across multiple switches, effectively pipelining the signal transmission. In addition, NoC offers several promising features. First, it transmits packets instead of messages, where dedicated address lines, like those in bus-based communication systems, are no longer necessary since the destination addresses of packets are embedded in the packet. Second, data transfer can be executed in parallel if the network provides more than one data transfer channel between a sender and a receiver. Accordingly, unlike bus-based communication SoC, NoC presents theoretical infinite scalability, facile IP core reusing, and higher level of parallelism[1] [2] [3].

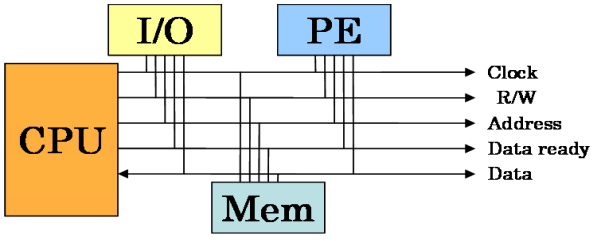

Figure 1. *Bus-based Architecture*

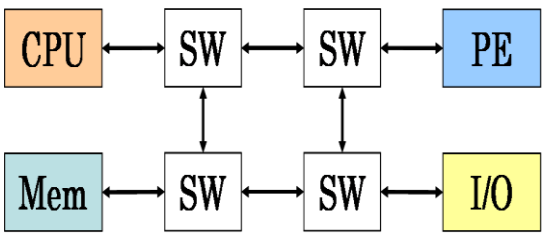

Figure 2.*Network-on-Chip Architecture*

2 Network-on-Chip

In this section, we discuss the background of interconnection architecture and provide a review of some related works in this field.

2.1 Network Topology

The NoC topology is structured to connect between multiple cores and routers. The network interface (NI) converts the data communication into packets having header information. Since NoC topology depends on the layout of cores and routers, it makes a large impact on expansibility and the performance of the system. Topology has various kinds, such as the mesh topology and torus topology, which are the most basic configuration as showed below.

The mesh-based architecture (see Fig.3) proposed by Kumar [4], is the simplest topology arranged into an MxN grid. It consists of the same number of nodes and routers. Routers are connected to each other, and connected to cores using a bidirectional link, and_all routers have 5x5 I/O port, one for each direction, and the number of ports can be reduced depending on the position of the router in the network, like those located in the edges. Each router is thereby connected to four neighboring router and one local core.

Torus-based architecture (see Fig.4) proposed by Dally [5] is constituted like mesh. The only difference is that a router located in the edge is connected to a router located in the edge of the other side directly by a wraparound channel. This architecture comes to have a long transmission distance of the packet in comparison with mesh topology. However the number of the hops tends to become smaller generally.

Furthermore, networks may have irregular topologies (see Fig.5) other than regular ones. The example showed in Fig.5 consists of irregular mesh topology where some cores are connected to two routers, and others are omitted or contain a different number of ports [7]. The customization of the NoC topology [10] with appropriate number and type of routers and the proper number of connections and bandwidth, will lead us to a communication structure with less area and lower average communication latency than the standard NoC topology. Therefore, custom topologies have advantage to basic architecture such as mesh topology and torus topology.

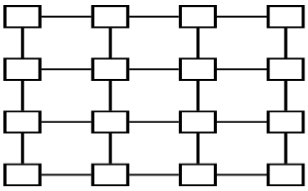

Figure 3. *Mesh topology*

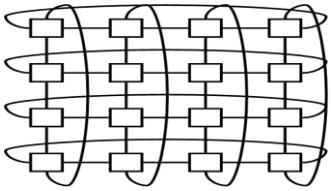

Figure 4. *Torus topology*

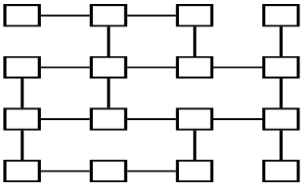

Figure 5. *Custom topology*

2.2 Routing

Routing is the mechanism of determining the path that a packet traverses from the source node to the destination node. Routing can be divided according to the number of data transferring forward from a source node to a destination node. In unicast routing, the packet data has a single destination, while in the case of multicast routing, the packet data has multiple destinations. (see Fig.6)

Also, routing algorithm can be proposed in two types: deterministic and adaptive routing. Deterministic routing algorithm always routes along a deterministic path that is regardless of the current state on the network. Examples of such routing algorithm are: XY routing, *North-first*, *South-first*, *East-first*, and *West-first*. On the other hand, adaptive routing algorithm takes the current network state into account when deciding a path, resulting in a variation of the routing path with time.

According to where routing decisions are taken, it is possible to classify the routing into Source and Distributed routing. In Source Routing, the whole path is decided at the source node. Therefore, the header of the packet has to carry all the routing information, increasing the packet size. In Distributed Routing, The path can be chosen as a function of the network instantaneous traffic condition. Distributed routing can also take into account faulty paths, resulting in fault tolerant algorithms [8].

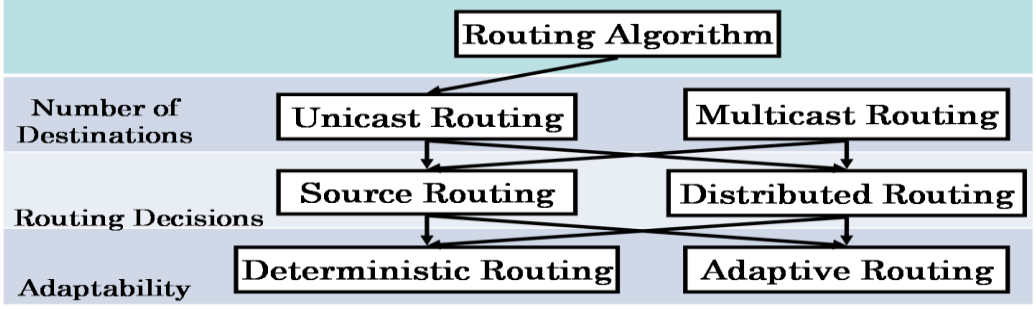

Figure 6. *Routing Algorithm*

2.3 Switching

Switching techniques can be classified based on the network characteristics, as shown in Fig.7. Circuit switching establishes a dedicated communications channel that is a link between source node and destination node connecting them for the duration of the communication session before the messages are being transferred. The link is held until all the data is transmitted. On the other hand, packet switching transmits the packets without reserving the entire path. Many messages are divided into packets at the source node and then transfer into a network. Packets move along a route determined by the routing algorithm and traverse through a series of network nodes and finally arrive at the destination node.

Furthermore, packet switching can be classified as Wormhole Switching, Store & Forward (S&F) Switching, Virtual Cut Through (VCT) Switching.

- ***Wormhore Switching*** : Even if wormhole switching does not forward each flit to a buffer entirely, it can go ahead through the node steadily. Other flits belonging to the same packet simply follow the path taken by the header flit. If the header flit is blocked and then the entire packet is blocked.
- ***Store & Forward (S&F) Switching***: S&F switching transfer a packet only where there is enough space available in the reserving buffer to hole the entire packet. Therefore, there is no need for dividing a packet into flits.
- ***Virtual Cut Through (VCT) Switching*** : VCT switching is the more commonly used switching technique. Unlike S&F, the VCT algorithm divides a packet into flits, which may be further divided into flit. VCT forwards it like Wormhole Switching divided into flit.

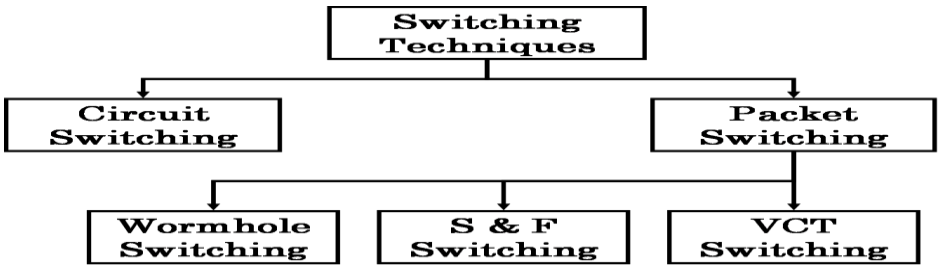

Figure 7. *Switching techniques*

2.4 Flow Control

Flow control depends on how network resources, such as channel bandwidth, buffer capacity, and control state, are allocated to a packet traversing the network. Flow control can be classified as Bufferless Flow Control and Buffered Flow Control. These have a relation of the trade-off with latency in throughput. In addition, there are some methods to prevent overflow by data transfer with a sender node and the receiver node in Buffered Flow Control. For example, Credit Based Flow Control, Handshaking Signals Flow Control, ACK/NACK Flow Control, Stall-Go Flow Control, and T-Error Flow Control, as shown in Fig.8.

- *Credit Based Flow Control* : Sender is keeping count of data transfers. Once the transmitted data packet is either consumed or transmitted at receiver, a credit is sent back. Therefore, sender cannot forward packet data when the credit of receiver is full.
- *Handshaking Signals* : A VALID signal is sent whenever a sender transmits any flits. The receiver acknowledges by asserting a VALID signal after consuming the data flit.
- *ACK/NACK Flow Control* : A copy of a data flit is kept in a buffer until an ACK signal is received. On assertion of ACK, the flit is deleted from the buffer; instead if a NACK signal is asserted then the flit is scheduled for retransmission.
- *Stall-Go Flow Control* : When there is an empty buffer space at receiver, a GO signal is sent to receiver. Upon the unavailable of buffer space, a STALL signal is sent
- *T-Error Flow Control* : It aims at enhancing the performance at the cost of reliability. When congestion is in a state data transmission, it should avoid use.

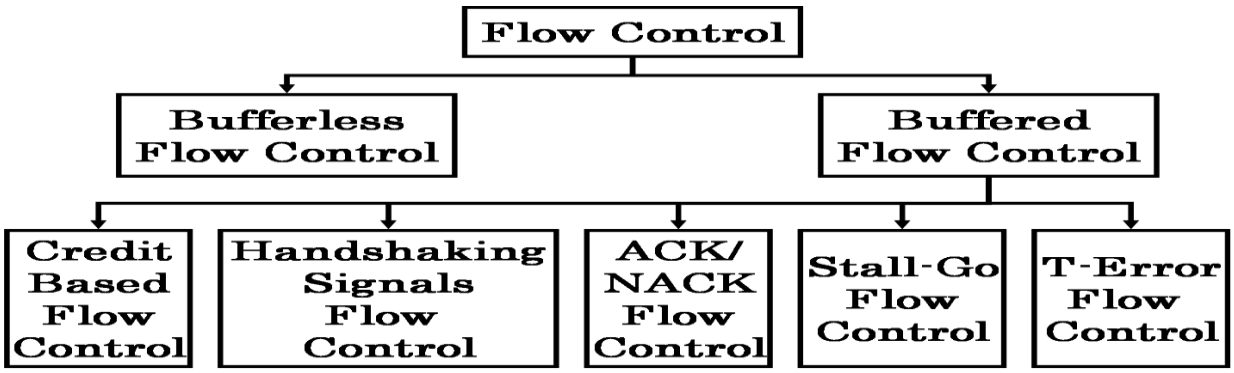

Figure 8. *Flow Control techniques*

3 OASIS NoC Architecture

We previously designed a NoC named OASIS [2] [3] which is a 4x4 mesh topology. It consists of the same number of cores and routers as shown in Fig.9. Each switch can have 5 I/O ports. This number can be reduced depending on the position of the router in the network, like routers located in the edges. The maximum connection pattern in the network of each switch is formed of Local, North, East, South, and West.
OASIS NoC employs static *XY-routing* and *Distributed* routing, which means that all flits contain routing information about the path between source and destination nodes.

Unlike the *Source* routing, in distributed routing each node receives some information about the network from its neighboring nodes and uses this information to determine the manner in which it forwards its traffic. The fact is that it determines a path at each hop to the next port. The next port direction in current node is decided in the input port by comparing the current node address and the destination node address: If X destination address is larger than X address of the current node, next port is EAST, in an opposite situation, next port is WEST. If Y address of destination node is larger than Y address of current node, next port is SOUTH, next port is NORTH in an opposite situation. When current and destination address are equal, next port is SELF.
OASIS NoC flit structure is indicated in Fig.10. Tail (1bit) signals the end of a packet, Next-port (5bit) indicates the output direction, X destination address (3bit), Y destination address (3bit), and Payload (8bit).
OASIS NoC has exchanged additional functionality to the design by implementing Stall-Go flow control and round-robin scheduling. Stall-Go is a very simple realization of an ON/OFF flow control, it requires just two control signals: one going forward and flagging data availability and one going backward and signaling either a condition of

buffers filled (STALL) or of buffers free (GO). Stall-Go can be implemented with distributed buffering along the link. It has some buffers which store multiple flits at each input port. Buffer size is an important NoC parameter that affects area utilization and throughput.

Topology network has some problems like large delay and number of hops though various improvements are done as for OASIS NoC .Therefore we planed some techniques optimizing the network topology.

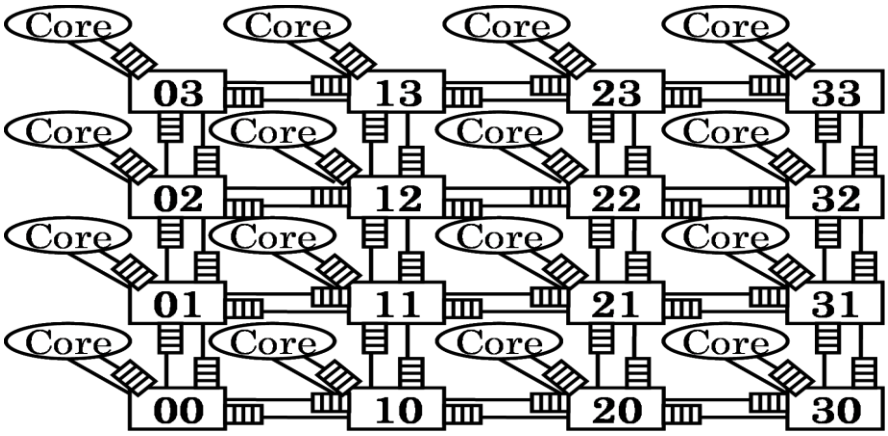

Figure 9. *4x4 Mesh topology*

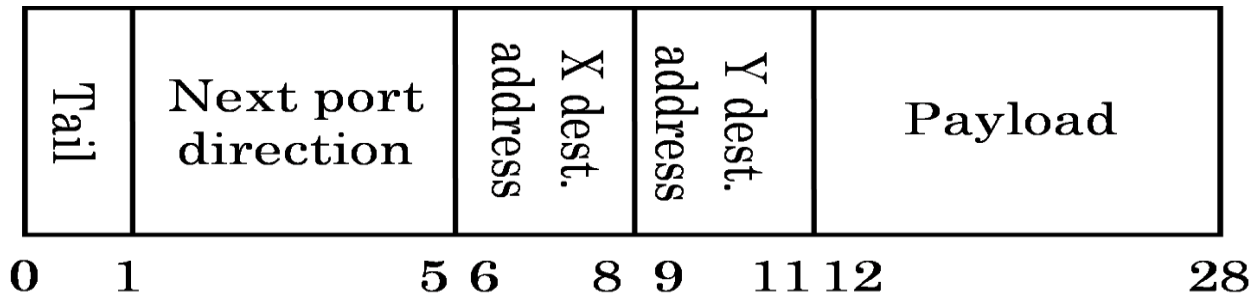

Figure 10. *Flit Structure*

3.1 Network topology

The network topology of OASIS NoC is set 4x4 mesh topology to consist of a router and a node of the same number (see Fig.11). Each route, node and interconnected router consists of the communication channel and then, all routers have 5x5 I/O port other than a router located in the edges. Each router is thereby connected to four neighboring router and one local core. The connection pattern of each router is formed of Local, North, East, South, and West.

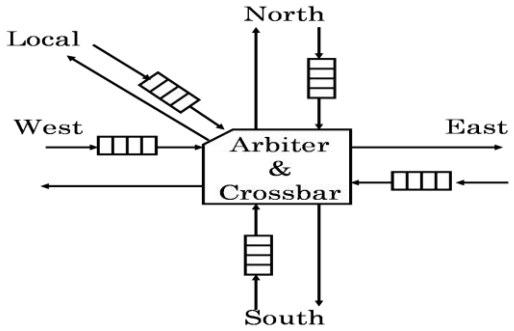

Figure 11. *Router Structure*

3.2 Routing

The routing of OASIS NoC employs XY routing and distributed routing. XY-routing becomes North, East, South, and West. From these four directions, eight 90-degree turns can be formed. The eight turns form two abstract cycles as shown in Fig.12. Unlike source routing, distributed routing does not depend on the path and is determined a path with every hop at next port. The next port direction in next address is decided in input port. To compare next address and destination address, next port in next address is decided. If X of destination address is larger than X of next address, next port is EAST in a contrasting situation, next port is WEST. If Y of destination address is larger than Y of next address, next port is SOUTH in a contrasting situation, next port is NORTH. When next address and destination address is equal, next port is SELF.

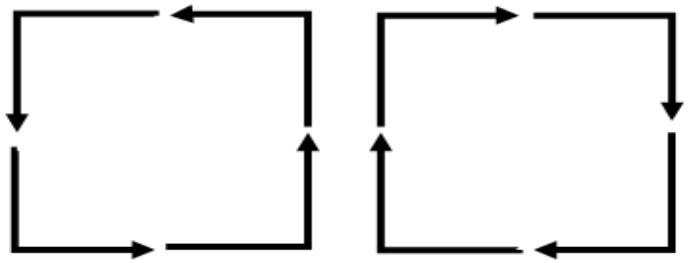

Figure 12 *Turn model*

3.3 Flow Control

Overflow happens and cannot forward precisely data transfer when the capacity of buffer exceeds it. Therefore, OASIS NoC prevents overflow by implementing Stall-Go Flow Control. In addition, FIFO is implemented by a buffer in order to employ Wormhole Switching. Data transfer of Stall-Go Flow Control controls the buffer situation of the receciver node by a signal. Therefore, if FIFO is full, Stall signal is sent to sender node and data transfer is blocked.

The following figures show Avoiding Buffer overflow architecture and State machine of Stall-go Flow Control.

.

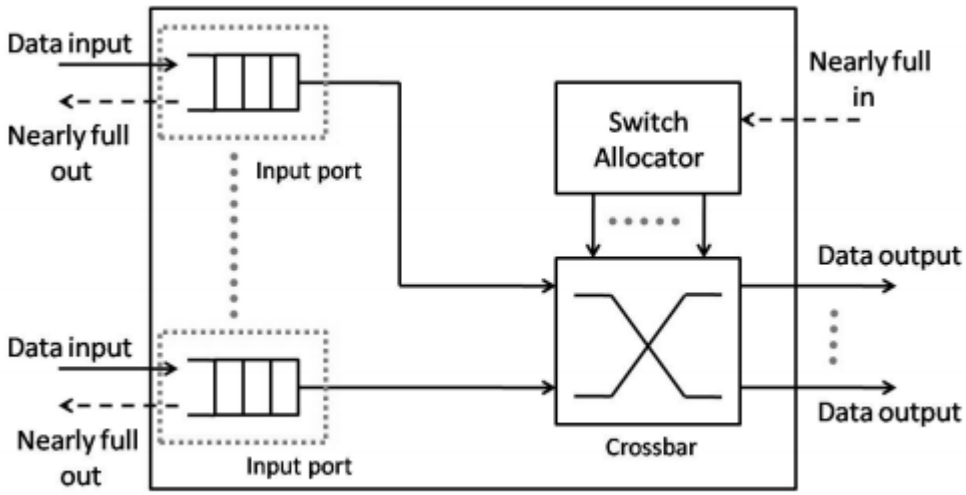

Figure 13. *Avoiding Buffer overflow architecture*

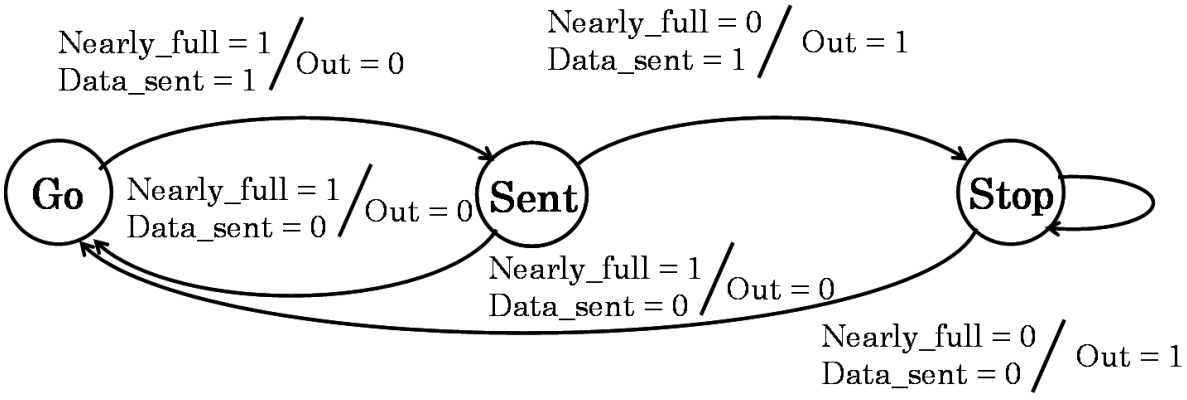

Figure 14. *State machine of Stall-go flow control*

4 Module Structure (Hierarchy)

Each module is described with Verilog HDL.

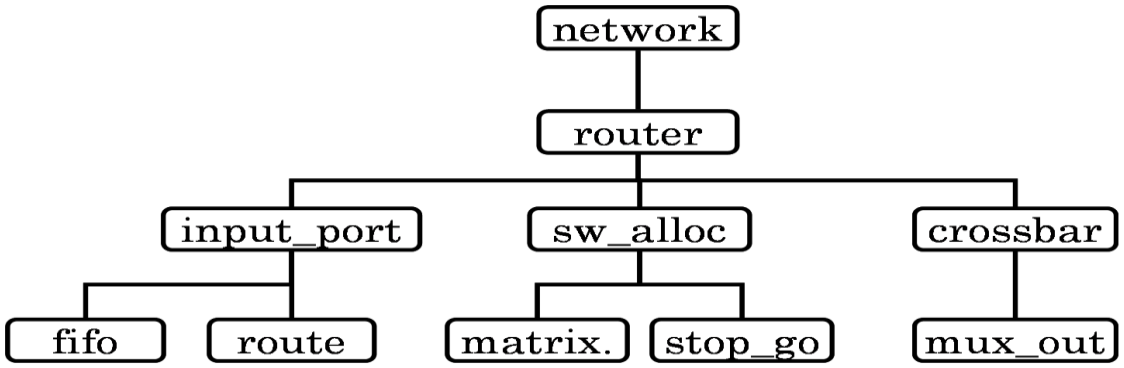

Figure 15. *Module Structure (Hierarchy)*

Module explanation

- **network.v**

  This program connects multi-routers about 2x2 or 3x3.

- **router.v**

  This program structures one router.

- **input_port**

They have buffering and routing mechanisms. Flits are stored to buffers and XY-routing is calculated here.

- **input_port.v**

  This program is input port of router. The input frit has the inside buffer, and if the store is done, and the permission of arbiter descends, is output to crossbar.

- **fifo.v**

  This program is buffers of input port. The stop signal is sent to the router in the transmission origin according to the number of stores in the buffer.

- **route.v**

  This program decides it for the output of the router of the following forwarding site.

  Routing is Static XY routing. X direction is moved, and then Y dirrection is moved.

- **sw_alloc**

  It has scheduling (Round-robin) and flow control mechanism (Stall-Go Flow Control). It calculates which input flits are outputted at first by scheduling. And flow control is used to send flits correctly.

- **sw_alloc.v**

  This program is arbiter in router.

- **stop_go.v**

  It is scheduled to request it for the output as which two or more flits are the same. Moreover, it doesn't transmit full of buffers in the destination.

- **matrix_arb_formultistage.v**

  This program is scheduling management in router
  matrix arbiter format is nxn.

$$\begin{bmatrix} X & p_{12} & p_{13} & p_{14} \\ p_{21} & X & p_{23} & p_{24} \\ p_{31} & p_{32} & X & p_{34} \\ p_{41} & p_{42} & p_{43} & X \end{bmatrix}$$

Each matrix element (i,j) records the binary priority between each pair of inputs. For example, I suppose request i has a higher priority than request j, and then the matrix element (i,j) will become set to 1, and matrix element (j,i) will become set to 0. And if the row is all 1, arbiter send grant signal to input port. Then its priority is set to be the lowest among all requests. So, all requests is uploaded.

$$\begin{pmatrix} X & 1 & 1 & 1 \\ 0 & X & 0 & 0 \\ 0 & 1 & X & 1 \\ 0 & 1 & 0 & X \end{pmatrix} \Rightarrow \begin{pmatrix} X & 0 & 0 & 0 \\ 1 & X & 0 & 0 \\ 1 & 1 & X & 1 \\ 1 & 1 & 0 & X \end{pmatrix}$$

(a) (b)

For example, this means Request 1 has the highest priority, second highest is request 3, third is request 4, and the lowest is request 2. And, Request 1 can be granted a resource, and the priority will be change to the lowest. Request 1 was the lowest priority. So, request 1 was all 0. This result is uploaded.

- **crossbar**

  It forwards output flits to the next port.

- **crossbar.v**

  This program is output port. The flit output from input port is assumed to be an input, and to which direction it outputs it according to the control signal from arbiter is decided.

- **mux_out.v**

  One direction of the output port is controlled.

- **define.v**

  This program is Configuration file. The composition, the size of the system, and

the buffer length etc. of flit can be set.

5 Short Pass Link Architecture

There are the various faults which is large latency with standard mesh topology and large number of hops that OASIS NoC architecture should optimize. Therefore we plan technique optimizing about network topology.

5.1 Short Pass Link (SPL) Feature

Our design architecture (see Fig.16) added the Short Pass Link (SPL) to optimize OASIS NoC performance, such as the large communication latency and number of hops. As discussed in section 1, we modify the previous architecture to a custom mesh topology by adding an SPL like custom link. While adding SPL may cause large area and power consumption, we assume that performance is improved. After simulating the testbench, communication frequencies and distances of the whole network is calculated and then a suitable link is inserted. Therefore, these ways assure the link insertion in a fair way. By repeating simulation, SPL budget can be updated, and changed to insure a better performance. Large SPL budget may cause many obstacles like livelock and deadlock, and then increasing the area and energy consumption like torus topology despite using the mesh topology. Now, we must modify the program directly to be performed manually.

The main modification in OASIS NoC consists in the number of ports and flit structure. Though the connection pattern of each previous switch in the network, had to be formed of Local, North, East, South, and West. The added SPL switch was modified as shown Fig.17 and Fig.18. Other switches do not have Extra-port in order to reduce power consumption, since Extra-port is different from general ports providing a short-cut link to the target node. Next port direction size in the flit

structure is increased to six bit (see Fig.19), representing each possible direction Local, North, East, South, West, and Extra.

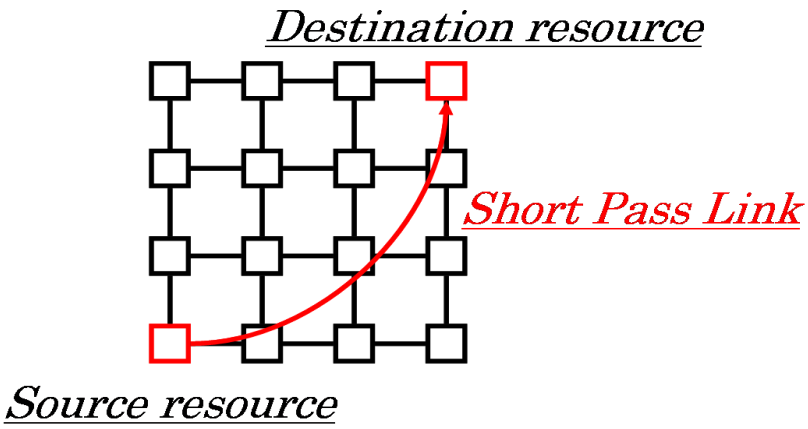

Figure 16. ***Short Pass Link***

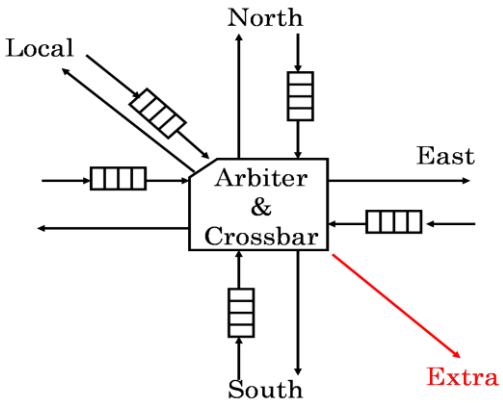

Figure 17. ***Source node of SPL***

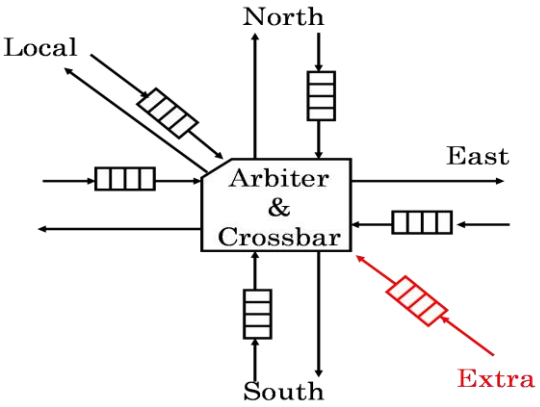

Figure 18. ***Destination node of SPL***

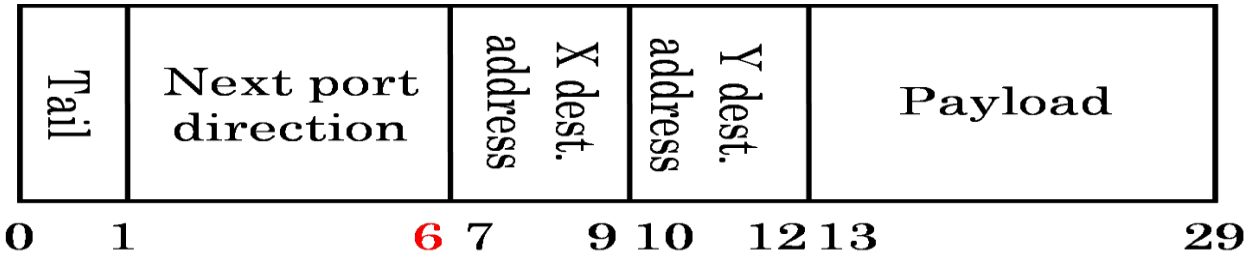

Figure 19. *Flit Structure with adding Extra port direction*

5.2 Algorithm

To facilitate the analysis, we first define a set of variables.

- S: Available SPL resource
- i,j: Between a source node (i) and the destination node (j).
- Fij: Communication frequency
- Dij: Communication distance
- Cij: Total cost

This algorithm first depends on the available SPL resource budget, which must be limited by a certain value, as shown in Fig.20, in order to avoid many disadvantages like large area and power consumption. Next step is to simulate the design with a testbench. This testbench uses a program that includes the hotspot traffic in the network to prove that SPL is an effective technique. Hotspot means that some messages concentrates on a specific node. When communication is concentrated on a specific node, like Hotspot, deciding the buffer size in each node may lead to performance degradation. However, performance is improved using SPL, by reducing the load in some specific nodes. Our algorithm investigates about communication frequencies and distances by simulation to enhance the network performance greatly. The provided communication frequencies and distances are used to decide where we should insert a SPL depending on the simulation results.

The provided communication frequencies are calculated using (1). The communication volume between the processing elements (PE) located at source node *(i)* and the PE located at destination node *(j)* is denoted by *Vij*. We compute the communication frequency between the PEs *i* and *j*, *fij* according to this formula:

$$f_{ij} = \frac{V_{ij}}{\sum_p \sum_{p \neq q} V_{pq}} \qquad (1)$$

On the other hand, the provided communication distances are obtained by (2). For a standard mesh based network, the Manhattan distance dM(i,j) is used to compute d(i,j).

$$d_M(i,j) = |i_x - j_x| + |i_y - j_y| \qquad (2)$$

As a result, this algorithm inserts a SPL in highest path cost and then improves the network performance. If SPL available budget is exhausted, we obtain the final architecture, which inserts SPL in the suitable path, as output.

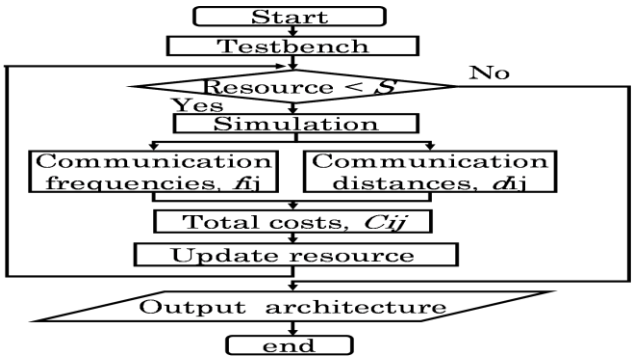

Figure 20. *Short Pass Link (SPL) Insertion Algorithm*

6 Performance Evaluation

OASIS NoC is designed using Verilog HDL, synthesized with Altera CAD tools, and simulated with Model-Sim [11]. It focuses only on transfer, and do not consider real system behavior. We compare area, speed and power between the two topology architecture types of OASIS NoC. Table 1 presents the parameters used for the synthesis for both OASIS NoC design, and Table 2 illustrates the hardware evaluation results obtained comparing the two designs. The results show that the logic utilization is increased by 9.8%. The increased number of ALUTs can be explained by the additional number of port and buffers at each switch. In term of clock speed, OASIS NoC with SPL insertion architecture under-performs the OASIS NoC architecture by 1.7%, and power is 0.2% higher.

We send some flits to concentrate on a specific node in order to estimate the delay caused by this traffic and then compare the communication delay between the topology architecture types of OASIS NoC. Table 3 illustrates for each NoC architecture and network size the performance evaluation results. The node (1, 1) is selected as the destination node and four source nodes are selected to generate the hotspot traffic. From source node (0, 0) to destination node (1, 2), we prepare the data path to pass the hotspot traffic, and then to compare network inserted SPL with normal network. The table indicates the execution cycles between the time of sending all the flits and the time when the destination node receives them all. To compare 3x3 mesh topology networks, the data path from source node (0, 0) to destination node (1, 2) passing the hotspot was improved by 48.3%. Furthermore, the data path from source node (1, 0) to destination node (1, 1) was improved by 1.3%. The reason can be explained by reducing the load on

destination node (1, 1). Destination node (1, 1) has decreased one load by using SPL though five loads.

Parameter	OASIS	OASIS_SPL
Network Size	3x3-mesh	3x3-mesh
Header Size	10 bit	11 bit
Payload Size	8 bit	8 bit
Flit size	28 bit	29 bit
Routing	Static X-Y	
Forwardong	Womehole switching like	
Flow Control	Stall-go	
Scheduling	Round-robin	
Target Device	Altera Stratix Ⅲ	

Table 1: Simulation Parameters

Parameter	3x3 mesh	3x3 custom
Area (ALUTs)	5761	6324
Speed (MHz)	203.54	197.90
Power (mV)	652.94	654.53

Table 2: Hardware Complexity Analysis

Architecture	3x3 mesh	3x3 custom	
Node	Execution cycles		Improvement
(0,0) -> (1,2)	**383**	**198**	***48.3%***
(1,0) -> (1,1)	392	387	***1.3%***
(2,1) -> (1,1)	385	385	0%
(1,2) -> (1,1)	386	386	0%
(0,1) -> (1,1)	386	386	0%

Table 3: Performance Evaluation

7 Conclusion & Future Work

We introduced an optimized technique to insert SPL in mesh topology. The purpose of SPL insertion algorithm was to reduce communication delay and number of hops. Evaluation results show that in term of speed OASIS NoC with SPL under-performs the OASIS NoC architecture, observing a small power overhead with an area utilization penalty. Despite the increasing hardware complexity, OASIS NoC with SPL performance shows an improvement obtained by reducing cycles. For example, the data path inserted SPL was reduced by 48.3%cycles for 3x3 mesh topology network In addition, one load to destination node (1,1) decreased, and the data path from source node (1,0) to destination node (1,1) was improved by 1.3%. Therefore, inserting SPL in the mesh topology is the technique that is effective for hotspot traffic.
In future, we will try to evaluate the system with large benchmark over large network size, and then design a Reconfigurable Network on Chip Architecture. NoC was designed to fulfill the bandwidth requirement between cores for a certain set of running application, but not so optimal for others. Therefore we can use this optimization in another application.

8 Reference

1. Ben Abdallah , and M. Sowa, Basic Network-on-Chip Interconnection for Future Gi- gascale MCSoCs Applications: Communication and Computation Orthogonalization, Proc. of The TJASSST2006 Symposium on Science, Society and Technology, DEC. 4-6, 2006.
2. Kenichi Mori, A. Ben Abdallah, Kenichi Kuroda, "Design and Evaluation of a Complexity Effective Network-on-Chip Architecture on FPGA", The 19th Intelligent System Symposium (FAN 2009), Sep. 2009, pp.318-321.
3. Kenichi Mori, Adam Esch, A. Ben Abdallah, Kenichi Kuroda, "Advanced Design Issue for OASIS Network-on-Chip Architecture", 2010 International Conference on Broadband, Wireless Computing, Communication and Applications
4. Umit Y. Ogras, Radu Marculescu, "It's a Small World After All": NoC Performance Optimization Via Long-Range Link Insertion", IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 14, NO. 7, JULY 2006
5. S. Kumar, A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A Network on Chip Architecture and Design Methodology," Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 105–112, April 2002.
6. W. J. Dally and B. Towles, "Route packet, not wires: On-chip interconnection networks,"

Proceedings of Design Automation Conference, pp. 684–689, 2002
7. Vestias M., Neto H., "Area and Performance Optimization of a Generic Network-on-Chip Architecture", Symposium on Integrated Circuits and Systems Design, SBCCI'06, 68-73, 2006.
8. Mello, A. V; Ost, L. C; Calazans, N. L; Moraes, F. G., "Evaluation of Routing Algorithms in Mesh Based NoCs", TECHNICAL REPORT SERIES Number 040, May, 2004
9. Vestias M., Neto H, "Co-Synthesis of a Configurable SoC Platform based on a Network on Chip Architecture", Asia and South Pacific Design Automation Conference ASP-DAC 2006, jan. 24-27 2006
10. Wen-hsiang Hu , Seung Eun Lee , Nader Bagherzadeh, "DMesh: a Diagonally-Linked Mesh Network-on-Chip Architecture", NoCArc 2008, Lake Como, Italy, November 8th, 2008
11. Rickard Holsmark, " Modeling and Prototyping of a Network on Chip",Master Thesis, Electronics, Sweden Jonkoping University, 2002.
12. journal of Engineering, Computing and Architecture Volume 3, Issue 1, 2009

9 Appendix

The benchmark code with hotspot traffic

```
`timescale 1ns/1ns

`ifndef VCS
  `include "defines.v"
`endif

`define X_WIDTH    3
`define Y_WIDTH    3
`define N_flits    100

module test;

    wire [(`WIDTH-1):0] w_00,w_01,w_02;
    wire [(`WIDTH-1):0] w_10,w_11,w_12;
    wire [(`WIDTH-1):0] w_20,w_21,w_22;

    wire [`X_WIDTH*`Y_WIDTH-1:0] stop_out;
```

```
//register list object value of test input and output
reg [(`WIDTH*`X_WIDTH*`Y_WIDTH)-1:0] data_in;
reg [`X_WIDTH*`Y_WIDTH-1:0]    stop_in;
reg            clk;
reg            reset;

//******* define of input data ******//
reg [(`WIDTH-1):0]     data_in_00,data_in_01,data_in_02;
reg [(`WIDTH-1):0]     data_in_10,data_in_11,data_in_12;
reg [(`WIDTH-1):0]     data_in_20,data_in_21,data_in_22;

//******* payload of input data ******//

reg [(`DATA_WIDTH-1):0] payload_1;
reg [(`DATA_WIDTH-1):0] payload_2;
reg [(`DATA_WIDTH-1):0] payload_3;
reg [(`DATA_WIDTH-1):0] payload_4;
reg [(`DATA_WIDTH-1):0] payload_5;

//******* define of distination address ******//
reg [5:0]              dest_00;//dist_(y,x) = 6'b000000
reg [5:0]              dest_01;//dist_(y,x) = 6'b000001
reg [5:0]              dest_02;//dist_(y,x) = 6'b000010
reg [5:0]              dest_10;//dist_(y,x) = 6'b001000
reg [5:0]              dest_11;//dist_(y,x) = 6'b001001
reg [5:0]              dest_12;//dist_(y,x) = 6'b001010
reg [5:0]              dest_20;//dist_(y,x) = 6'b010000
reg [5:0]              dest_21;//dist_(y,x) = 6'b010001
reg [5:0]              dest_22;//dist_(y,x) = 6'b010010

//******* define of next port direction at only Master router ******//
reg [4:0]              next_SELF;
reg [4:0]              next_NORTH;
reg [4:0]              next_EAST;
reg [4:0]              next_SOUTH;
reg [4:0]              next_WEST;

//******* tail information: 0=not tail, 1=tail ******//
```

```
reg                     tail;
reg                     not_tail;

//loop cycle number
integer         i,j,k;

//send 10 times counter
integer   n,m,l;
integer   o,r;

// send flits number
integer flits;

//counter

        reg [(`DATA_WIDTH-1):0] p1;//payload for counter
        reg [(`DATA_WIDTH-1):0] p2;
        reg [(`DATA_WIDTH-1):0] p3;
        reg [(`DATA_WIDTH-1):0] p4;
        reg [(`DATA_WIDTH-1):0] p5;

        reg [31:0] total1;
        reg [31:0] total2;
        reg [31:0] total3;
        reg [31:0] total4;
        reg [31:0] total5;

        integer t;

        integer counter1 [0:`N_flits-1];
        integer counter2 [0:`N_flits-1];
        integer counter3 [0:`N_flits-1];
        integer counter4 [0:`N_flits-1];
        integer counter5 [0:`N_flits-1];

        integer FS_00;//old flits which is sent from (0,0), and arrive next.
        integer FS_01;//old flits which is sent from (1,0), and arrive next.
        integer FS_02;//old flits which is sent from (2,0), and arrive next.
```

```
        integer FS_10;//old flits which is sent from (0,1), and arrive next.
        integer FS_11;//old flits which is sent from (1,1), and arrive next.
        integer FS_12;//old flits which is sent from (2,1), and arrive next.
        integer FS_20;//old flits which is sent from (0,2), and arrive next.
        integer FS_21;//old flits which is sent from (1,2), and arrive next.
        integer FS_22;//old flits which is sent from (2,2), and arrive next.

        integer LS_00;//new flits which is sent from (0,0), and arrive later.
        integer LS_01;//new flits which is sent from (1,0), and arrive later.
        integer LS_02;//new flits which is sent from (2,0), and arrive later.
        integer LS_10;//new flits which is sent from (0,0), and arrive later.
        integer LS_11;//new flits which is sent from (1,0), and arrive later.
        integer LS_12;//new flits which is sent from (2,0), and arrive later.
        integer LS_20;//new flits which is sent from (0,0), and arrive later.
        integer LS_21;//new flits which is sent from (1,0), and arrive later.
        integer LS_22;//new flits which is sent from (2,0), and arrive later.

        integer send_1;//now sending # flits
        integer send_2;//now sending # flits
        integer send_3;//now sending # flits
        integer send_4;//now sending # flits
        integer send_5;//now sending # flits

    //top_module
    network network(
                                .clk(clk),
                                .reset(reset),
                                .data_in({data_in_22,data_in_21,data_in_20,
                                          data_in_12,data_in_11,data_in_10,
                                          data_in_02,data_in_01,data_in_00}),
                                .data_out({w_22,w_21,w_20,
                                           w_12,w_11,w_10,
                                           w_02,w_01,w_00}),
                                .stop_in(stop_in),
                                .stop_out(stop_out));

    //clock generation
    always #50 clk = ~clk;
```

```
initial begin
    #0
    clk = 0;
    data_in = {(`WIDTH*`X_WIDTH*`Y_WIDTH){1'b0}};
    stop_in = 1'b0;
    reset = 1'b0;

    #100
    reset = 1'b1;

    #100
    reset = 1'b0;

    #100
    //*** dist_(x,y) indicates distination address for each router ***//
    payload_1 = 16'h0001; //for 16 bit
    payload_2 = 16'h1001; //for 16 bit
    payload_3 = 16'h2001; //for 16 bit
    payload_4 = 16'h3001; //for 16 bit
    payload_5 = 16'h4001; //for 16 bit

    //*** dist_(y,x) indicates distination address for each router ***//
    dest_00 = 6'b000000;
    dest_01 = 6'b000001;
    dest_02 = 6'b000010;
    dest_10 = 6'b001000;
    dest_11 = 6'b001001;
    dest_12 = 6'b001010;
    dest_20 = 6'b010000;
    dest_21 = 6'b010001;
    dest_22 = 6'b010010;

    //******* define of next port direction at only Master router ******//
    next_SELF = `SELF;
    next_NORTH = `NORTH;
    next_EAST = `EAST;
    next_SOUTH = `SOUTH;
```

```
next_WEST = `WEST;

//tail information
tail = 1'b1;
not_tail = 1'b0;


    n = 0;
    m = 0;
    l = 0;
    o = 0;
    r = 0;


p1 = 16'h0001; //for 16 bit
p2 = 16'h1001; //for 16 bit
p3 = 16'h2001; //for 16 bit
p4 = 16'h3001; //for 16 bit
p5 = 16'h4001; //for 16 bit

    FS_00 = 0;
    FS_01 = 0;
    FS_02 = 0;
    FS_10 = 0;
    FS_11 = 0;
    FS_12 = 0;
    FS_20 = 0;
    FS_21 = 0;
    FS_22 = 0;

    LS_00 = 0;
    LS_01 = 0;
    LS_02 = 0;
    LS_10 = 0;
    LS_11 = 0;
    LS_12 = 0;
    LS_20 = 0;
    LS_21 = 0;
```

```
        LS_22 = 0;

        send_1 = 0;
        send_2 = 0;
        send_3 = 0;
        send_4 = 0;
        send_5 = 0;

        flits = `N_flits-1;

        total1 = 0;
        total2 = 0;
        total3 = 0;
        total4 = 0;
        total5 = 0;

        t = 0;

  if(!reset)begin
      for(i=0;i<`N_flits;i=i+1)begin
          counter1[i] = 1;
          counter2[i] = 1;
          counter3[i] = 1;
          counter4[i] = 1;
          counter5[i] = 1;

      end

  //set each router payload at each cycle
for(k=0;k<flits*`X_WIDTH*`Y_WIDTH;k=k+1)begin
      #100
     // (0,0) -> (1,2)
     if(stop_out[0] != 1)begin
         if(n> (flits-20) )begin    data_in_00 = 0;
         end else begin
                data_in_00 = {payload_1,dest_21,next_EAST,tail};
                payload_1 = payload_1 + 1;
                LS_00 <= LS_00 + 1;
```

```
    end
     n = n + 1;
end else begin
     data_in_00 = 0;
end//if n end

// (1,0) -> (1,1)
if(stop_out[1] != 1)begin
    if(m>flits)begin    data_in_01 = 0;
    end else begin
            data_in_01 = {payload_2,dest_11,next_NORTH,tail};
            payload_2 = payload_2 + 1;
            LS_01 <= LS_01 + 1;
    end
    m = m + 1;
end else begin
     data_in_01 = 0;
end//if m end

// (2,1) -> (1,1)
if(stop_out[5] != 1)begin
    if(l>flits)begin    data_in_12 = 0;
    end else begin
            data_in_12 = {payload_3,dest_11,next_WEST,tail};
            payload_3 = payload_3 + 1;
            LS_12 <= LS_12 + 1;
    end
     l = l + 1;
end else begin
     data_in_12 = 0;
end//if l end

// (1,2) -> (1,1)
if(stop_out[7] != 1)begin
    if(o>flits)begin    data_in_21 = 0;
    end else begin
            data_in_21 = {payload_4,dest_11,next_SOUTH,tail};
            payload_4 = payload_4 + 1;
```

```
            LS_21 <= LS_21 + 1;
        end
         o = o + 1;
    end else begin
          data_in_21 = 0;
    end//if o end

    // (0,1) -> (1,1)
    if(stop_out[3] != 1)begin
        if(r>flits)begin    data_in_10 = 0;
        end else begin
            data_in_10 = {payload_5,dest_11,next_EAST,tail};
            payload_5 = payload_5 + 1;
            LS_10 <= LS_10 + 1;
        end
         r = r + 1;
    end else begin
          data_in_10 = 0;
    end//if r end

     //count stop
     //(0,0) -> (1,2)
     if(w_21[(`WIDTH-1):12] != p1)begin
            for(send_1=FS_00; send_1<LS_00; send_1=send_1+1)
               counter1[send_1] = counter1[send_1] + 1;
     end else begin
         for(send_1=FS_00+1; send_1<LS_00; send_1=send_1+1)
            counter1[send_1] = counter1[send_1] + 1;
                FS_00 = FS_00+1;
            p1 = p1+1;
     end

     //(1,0) -> (1,1)
     if(w_11[(`WIDTH-1):12] != p2)begin
            for(send_2=FS_01; send_2<LS_01; send_2=send_2+1)
               counter2[send_2] = counter2[send_2] + 1;
     end else begin
         for(send_2=FS_01+1; send_2<LS_01; send_2=send_2+1)
```

```
            counter2[send_2] = counter2[send_2] + 1;
                FS_01 = FS_01+1;
            p2 = p2+1;
        end
        //(2,1) -> (1,1)
        if(w_11[(`WIDTH-1):12] != p3)begin
              for(send_3=FS_12; send_3<LS_12; send_3=send_3+1)
                counter3[send_3] = counter3[send_3] + 1;
        end else begin
            for(send_3=FS_12+1; send_3<LS_12; send_3=send_3+1)
              counter3[send_3] = counter3[send_3] + 1;
                FS_12 = FS_12+1;
            p3 = p3+1;
        end
        //(1,2) -> (1,1)
        if(w_11[(`WIDTH-1):12] != p4)begin
              for(send_4=FS_21; send_4<LS_21; send_4=send_4+1)
                counter4[send_4] = counter4[send_4] + 1;
        end else begin
            for(send_4=FS_21+1; send_4<LS_21; send_4=send_4+1)
              counter4[send_4] = counter4[send_4] + 1;
                FS_21 = FS_21+1;
            p4 = p4+1;
        end
        //(0,1) -> (1,1)
        if(w_11[(`WIDTH-1):12] != p5)begin
              for(send_5=FS_10; send_5<LS_10; send_5=send_5+1)
                counter5[send_5] = counter5[send_5] + 1;
        end else begin
            for(send_5=FS_10+1; send_5<LS_10; send_5=send_5+1)
              counter5[send_5] = counter5[send_5] + 1;
                FS_10 = FS_10+1;
            p5 = p5+1;
        end

    end// loop k end

     #100
```

```
        data_in_00 = 0;
        data_in_01 = 0;
        data_in_12 = 0;
        data_in_21 = 0;
        data_in_10 = 0;

      for(t=0;t<`N_flits;t=t+1)begin
            #100

            total1 = counter1[t]+total1;
            total2 = counter2[t]+total2;
            total3 = counter3[t]+total3;
            total4 = counter4[t]+total4;
            total5 = counter5[t]+total5;

             end
    end//reset
    $finish;
    end // initial begin
endmodule //         test
```