System C based NoC Modeling Course Project

Carlo Dumlao

Toronto Metropolitan University

***Abstract*— Two 2D NoC systems (i.e. 4x4 mesh and 4x4 torus) were designed and modeled in a SystemC environment, and their performance was evaluated by generating uniform and neighbouring patterns. During simulation, it was observed that a packet delay is significantly higher when the uniform pattern is applied, in comparison to the neighbouring pattern. For the uniform, the average packet delay of 4x4 mesh and 4x4 torus is 33 and 28 clock cycles, respectively; on the other hand, the neighbouring pattern results in 14 clock cycles for both topologies.**

I. Introduction

The objective of this project is to investigate the architecture, behavior, and performance of 2D NoC systems that consist of routers (or switches) and IP cores. To achieve this, a representation of a single IP core was firstly designed in SystemC by interconnecting the source, sink, and router modules. Moreover, the arbiter of the router was also modified to support the required XY routing algorithm. Several of IP cores were then linked to one another to form two interconnection structures: 4x4 mesh and 4x4 torus. The functionality and performance (i.e. packet delay) of these topologies were then tested by generating various types of communication pattern from the source to the sink, such as uniform and neighbour traffic.

II. Theoretical

The performance of an NoC is often measured by a metric called packet delay which is the number of cycles required to deliver a packet from source to destination. The packet delay is expressed as:

$$Delay_{pkt_i} = N_{hops} + W_{pkt_i} \quad \text{Eq. 1}$$

where N_{hops} is the number of routers traversed by the packet and W_{pkt_i} is the contention delay. The hop-count delay depends on the routing algorithm implemented and interconnection structure of the NoC. On the other hand, W_{pkt_i} is dependent on the link contention that occurs when the packets compete for the same channel and buffer congestion which happens when the router buffers are in full capacity.

In this project, the average packet delay is calculated and printed into the terminal by introducing a new variable to both the source and sink modules:

- Source module: a variable clk_count is added within the module that increments every clock cycle. The source would then send this value into the sink via a packet.
- Sink module: similarly, a clk_count is implemented. Once a packet is received, the packet delay is easily calculated by finding the difference between this clk_count and the time that the packet was sent by the source module.

III. Design

A. Router Architecture

The complete architecture of the router is presented in Figure 1 below. Three components are crucial for the functionality of this module, including a FIFO, arbiter, and crossbar. The FIFO component contains four registers which are used to temporarily store the incoming packets. The arbiter is responsible for configuring the crossbar such that a packet from the FIFO is correctly delivered to the correct link. Meanwhile, the crossbar is similar to a multiplexer which physically connects the FIFO into one of the external data outputs of the router.

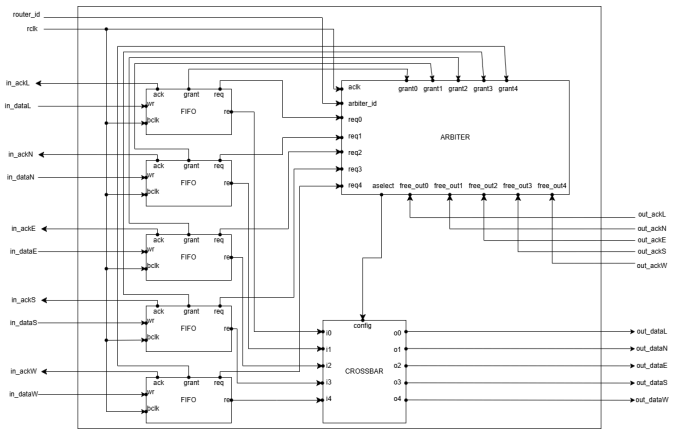

Fig. 1 Architecture of the router (or switch)

B. Source Module

The flow diagram of the source module is shown in Figure 2. Notice that the source module firstly checks whether the router FIFO is full or not. If it is full, the source will remain at idle until the router gives permission to send data again. On the other hand, if it is not full, the source will immediately generate a new packet to be sent to the sink module. The packet has a length of 21 bits and formatted as follows:

Packet = [data, source id, destination, imaginary clock bit, tail/header bit]

The imaginary clock bit flips between 0 and 1 for every packet generated, while the tail bit is asserted to signify the end of the message. In this design, the message has five packets.

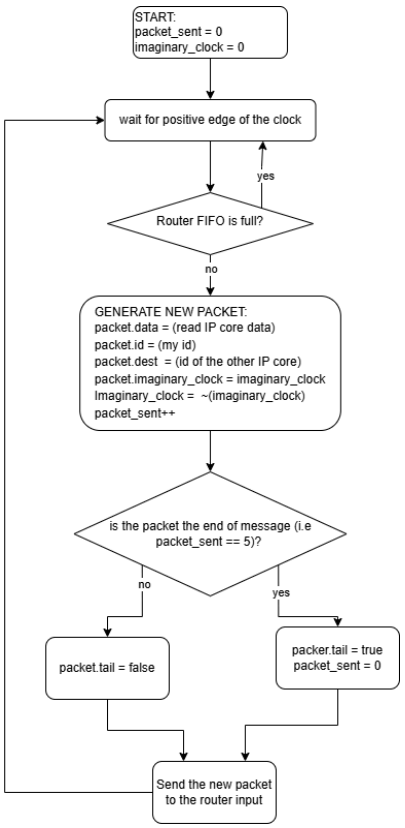

Fig. 2 Flow diagram of the source module

C. 4x4 Mesh

Figure 3 depicts the actual connection of the routers, forming a 4x4 mesh NoC. Note that the IP core at the top left corner has an id of 0, while the bottom right has an id of 15. This mesh relies on the XY routing algorithm to transport the packets – in other words, a packet is routed in the horizontal axis until it has reached the column of the destination, and then routed vertically towards the destination row. The functionality of this algorithm for a packet travelling in the x-axis is showcased in Figure 4. Put it simply, the router will compare the source id and destination. When both ids are identical, it implies that the router itself is the destination; otherwise, the packet is sent to the left link when the destination is less than source id, or to the right link when the destination is greater than source id.

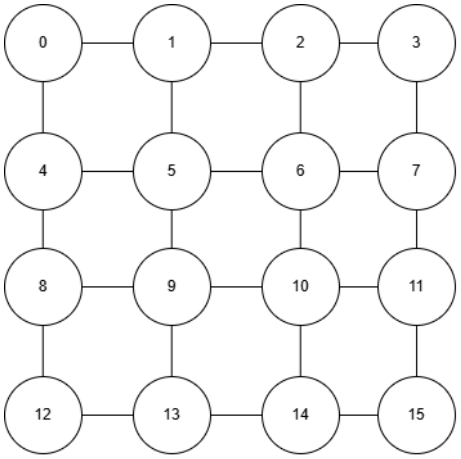

Fig. 3 Interconnection structure of a 4x4 mesh

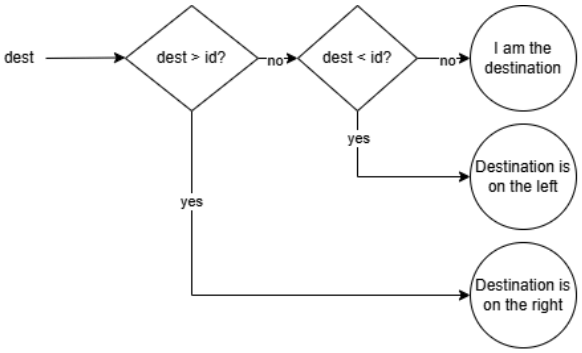

Fig. 4 XY routing algorithm for the mesh

D. 4x4 Torus

The 4x4 torus in Figure 5 is constructed by simply adding new links in the 4x4 mesh such that its edges are wrapped around forming a closed loop. As shown in Figure 6, the XY routing algorithm has been slightly modified to take into account those additional channels. Notice that that router now takes priority of the shortest path to deliver the packet by comparing the number of hops taken when the packet is sent to the loop channel and when it is sent directly to the destination.

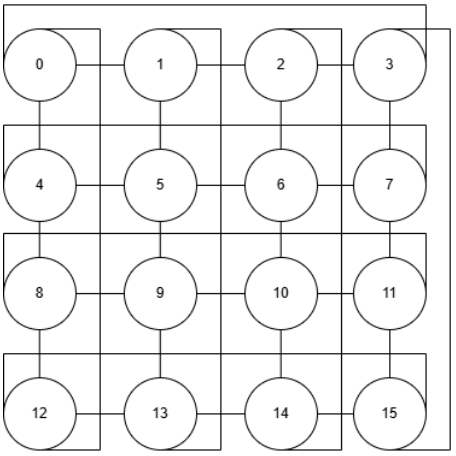

Fig. 5 Interconnection structure of a 4x4 torus

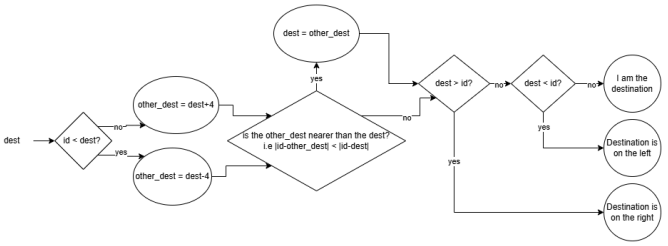

Fig. 6 XY routing algorithm for the torus

IV. Result

A. 1x2 NoC

The waveform in Figure 7 shows the behavior of the 1x2 NoC when the source starts sending several packets into the router. It is apparent that a message goes through three different stages before it can be retrieved by the sink. Let us consider the first packet sent, that is, 3E9:

1)Source to router1 (5ns): the source module generates and sends the packet into router1.

2)Router1 to router2 (10ns): once router1 takes hold of the packet, it then delivers it to router2.

3)Router2 to sink (15ns): finally, router2 notifies the sink that a new packet is available to be read.

It should be noted that the number of clock cycles consumed by each stage may vary depending on the traffic intensity within the NoC. In particular, the higher the communication traffic is, the slower the packet will be delivered due to contention delay.

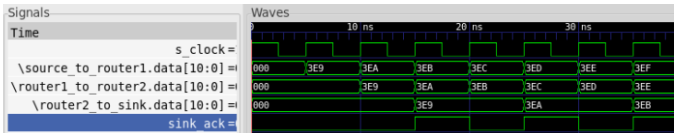

Fig. 7 Waveform of the 1x2 NoC

B. Uniform and Neighbouring Pattern

The uniform and neighbouring patterns that were used to analyze the performance of the NoC systems are tabulated in Table 1. In order to obtain the worst-case performance, each source was made to generate 250 packets starting at the first clock cycle to mimic a traffic congestion.

Table. 1 Source and destination ID of the uniform and neighbouring patterns

Source ID	Uniform: Dest. ID	Neighbour: Dest. ID
0	6	1
1	2	0
2	5	3
3	11	2
4	3	5
5	14	4
6	12	7
7	1	6
8	4	9
9	15	8
10	7	11
11	0	10
12	10	13
13	8	12
14	13	15
15	9	14

C. Mesh: Functionality

To test the functionality of the 4x4 mesh, a single packet is generated in IP0 with a destination id of 15. In Figure 8 which shows the path traversed, it is apparent that the packet is firstly transported horizontally from router 0 to 3, and then vertically from router 3 to 15 – as expected from an XY routing algorithm.

```
WARNING: Default time step is used for VCD tracing.
1 cycle: Source 0 sent packet with (dest = 15, data = 1)
2 cycle: Router 0 received packet with (id = 0, data = 1)
3 cycle: Router 1 received packet with (id = 0, data = 1)
4 cycle: Router 2 received packet with (id = 0, data = 1)
5 cycle: Router 3 received packet with (id = 0, data = 1)
6 cycle: Router 7 received packet with (id = 0, data = 1)
7 cycle: Router 11 received packet with (id = 0, data = 1)
8 cycle: Router 15 received packet with (id = 0, data = 1)
9 cycle: Sink 15 received packet with (id = 0, data = 1)
```

Fig. 7 Path traversed of the packet in a 4x4x mesh

D. Mesh: Uniform

In Figure 8, only a handful of IP cores are able to send and receive packets at once – in fact, during the 900th cycle, the throughput of the NoC is around 6 packets every 2 clock cycle. This implies that the mesh system suffers from link congestion; some of the IP cores monopolize a portion of the NoC channels, forcing packets from other IP cores to wait temporarily. As shown in Figure 9, the overall average packet delay is 33.17 cycles.

```
939 cycle: Source 0 sent packet with (dest = 6, data = 939)
939 cycle: Source 4 sent packet with (dest = 11, data = 939)
939 cycle: Source 7 sent packet with (dest = 1, data = 939)
939 cycle: Source 9 sent packet with (dest = 15, data = 939)
939 cycle: Source 11 sent packet with (dest = 0, data = 939)
939 cycle: Source 15 sent packet with (dest = 9, data = 939)
940 cycle: Sink 1 received packet with (id = 7, data = 911)
940 cycle: Sink 6 received packet with (id = 0, data = 911)
940 cycle: Sink 9 received packet with (id = 15, data = 911)
940 cycle: Sink 0 received packet with (id = 11, data = 897)
940 cycle: Sink 15 received packet with (id = 9, data = 911)
941 cycle: Sink 11 received packet with (id = 4, data = 905)
941 cycle: Source 0 sent packet with (dest = 6, data = 941)
941 cycle: Source 4 sent packet with (dest = 11, data = 941)
941 cycle: Source 7 sent packet with (dest = 1, data = 941)
941 cycle: Source 9 sent packet with (dest = 15, data = 941)
941 cycle: Source 11 sent packet with (dest = 0, data = 941)
941 cycle: Source 15 sent packet with (dest = 9, data = 941)
942 cycle: Sink 1 received packet with (id = 7, data = 913)
942 cycle: Sink 6 received packet with (id = 0, data = 913)
942 cycle: Sink 9 received packet with (id = 15, data = 913)
942 cycle: Sink 0 received packet with (id = 11, data = 899)
942 cycle: Sink 15 received packet with (id = 9, data = 913)
943 cycle: Sink 11 received packet with (id = 4, data = 907)
```

Fig. 8 Simulated mesh with a uniform pattern

```
Average packet delay in IP0: 72.724
Average packet delay in IP1: 43.508
Average packet delay in IP2: 14.688
Average packet delay in IP3: 27.8
Average packet delay in IP4: 14.688
Average packet delay in IP5: 21.316
Average packet delay in IP6: 44.084
Average packet delay in IP7: 21.316
Average packet delay in IP8: 21.316
Average packet delay in IP9: 44.084
Average packet delay in IP10: 27.8
Average packet delay in IP11: 50.588
Average packet delay in IP12: 34.14
Average packet delay in IP13: 14.688
Average packet delay in IP14: 34.14
Average packet delay in IP15: 43.892
# of packets sent: 4000
# of packets received: 4000
Total average packet delay: 33.1733
```

Fig. 9 Average packet delay of the mesh with a uniform pattern

E. Mesh: Neighbour

In Figure 10, it is evident that the neighbouring pattern transfers data more effectively: the throughput of the NoC is now 16 packets per 2 cycles. This is no surprise since the data is restricted to only be sent between adjacent or neighbouring nodes – as such, link congestion no longer exists. For this pattern, the performance bottleneck is the capacity of the routers to buffer the incoming packets. From Figure 11, the average packet delay is 14.688, which is almost half of the packet delay experienced by a mesh whose communication pattern is uniform.

```
479 cycle: Source 3 sent packet with (dest = 2, data = 479)
479 cycle: Source 4 sent packet with (dest = 5, data = 479)
479 cycle: Source 5 sent packet with (dest = 4, data = 479)
479 cycle: Source 6 sent packet with (dest = 7, data = 479)
479 cycle: Source 0 sent packet with (dest = 1, data = 479)
479 cycle: Source 7 sent packet with (dest = 6, data = 479)
479 cycle: Source 1 sent packet with (dest = 0, data = 479)
479 cycle: Source 8 sent packet with (dest = 9, data = 479)
479 cycle: Source 9 sent packet with (dest = 8, data = 479)
479 cycle: Source 10 sent packet with (dest = 11, data = 479)
479 cycle: Source 11 sent packet with (dest = 10, data = 479)
479 cycle: Source 12 sent packet with (dest = 13, data = 479)
479 cycle: Source 13 sent packet with (dest = 12, data = 479)
479 cycle: Source 14 sent packet with (dest = 15, data = 479)
479 cycle: Source 2 sent packet with (dest = 3, data = 479)
479 cycle: Source 15 sent packet with (dest = 14, data = 479)
480 cycle: Sink 2 received packet with (id = 3, data = 465)
480 cycle: Sink 3 received packet with (id = 2, data = 465)
480 cycle: Sink 0 received packet with (id = 1, data = 465)
480 cycle: Sink 4 received packet with (id = 5, data = 465)
480 cycle: Sink 5 received packet with (id = 4, data = 465)
480 cycle: Sink 6 received packet with (id = 7, data = 465)
480 cycle: Sink 7 received packet with (id = 6, data = 465)
480 cycle: Sink 8 received packet with (id = 9, data = 465)
480 cycle: Sink 9 received packet with (id = 8, data = 465)
480 cycle: Sink 10 received packet with (id = 11, data = 465)
480 cycle: Sink 11 received packet with (id = 10, data = 465)
480 cycle: Sink 12 received packet with (id = 13, data = 465)
480 cycle: Sink 1 received packet with (id = 0, data = 465)
480 cycle: Sink 13 received packet with (id = 12, data = 465)
480 cycle: Sink 14 received packet with (id = 15, data = 465)
480 cycle: Sink 15 received packet with (id = 14, data = 465)
```

Fig. 10 Simulated mesh with a neighbouring pattern

```
Average packet delay in IP0: 14.688
Average packet delay in IP1: 14.688
Average packet delay in IP2: 14.688
Average packet delay in IP3: 14.688
Average packet delay in IP4: 14.688
Average packet delay in IP5: 14.688
Average packet delay in IP6: 14.688
Average packet delay in IP7: 14.688
Average packet delay in IP8: 14.688
Average packet delay in IP9: 14.688
Average packet delay in IP10: 14.688
Average packet delay in IP11: 14.688
Average packet delay in IP12: 14.688
Average packet delay in IP13: 14.688
Average packet delay in IP14: 14.688
Average packet delay in IP15: 14.688
# of packets sent: 4000
# of packets received: 4000
Total average packet delay: 14.688
```

Fig. 11 Average packet delay of the mesh with a neighbouring pattern

F. Torus: Functionality

The functionality of the constructed 4x4 torus was tested by the same method as the mesh – that is, by sending a single packet to sink 15 from source 0. In Figure 12, it is evident that torus experiences less packet delay in comparison to the mesh; the data traverses through the horizontal loop channel (between router 0 and router 3) and vertical loop channel (between router 3 and router 15), quickly reaching the destination after only 4 cycles.

```
WARNING: Default time step is used for VCD tracing.
1 cycle: Source 0 sent packet with (dest = 15, data = 1)
2 cycle: Router 0 received packet with (id = 0, data = 1)
3 cycle: Router 3 received packet with (id = 0, data = 1)
4 cycle: Router 15 received packet with (id = 0, data = 1)
5 cycle: Sink 15 received packet with (id = 0, data = 1)
```

Fig. 12 Simulated mesh with a neighbouring pattern

G. Torus: Uniform

In Figure 13, the torus system experiences a link congestion when the uniform pattern is generated, similar to the mesh. However, an improvement in the packet delay can be observed in Figure 14. Compared to a mesh that has an average packet delay of 33 cycles, a packet of torus is able to reach its destination approximately 28.7 cycles. This reduction in delay is thanks to the loop channels, allowing a packet to reach the sink with a less number of hops.

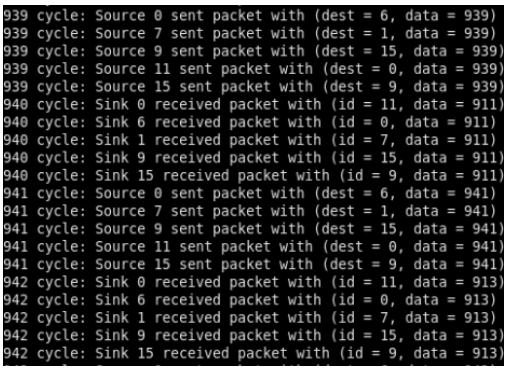

Fig. 13 Simulated torus with a uniform pattern

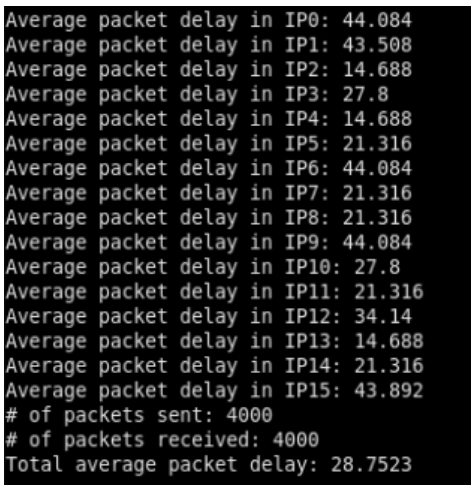

Fig. 14 Average packet delay of the torus with a uniform pattern

H. Torus: Neighbour

In regards to a neighbouring pattern, the average packet delay of the torus is presented in Figure 15. Notice that the delay remains unchanged: it is still a 14.688 clock cycle. This indicates that the packet delay of this pattern is independent of the interconnection structure of an NoC system.

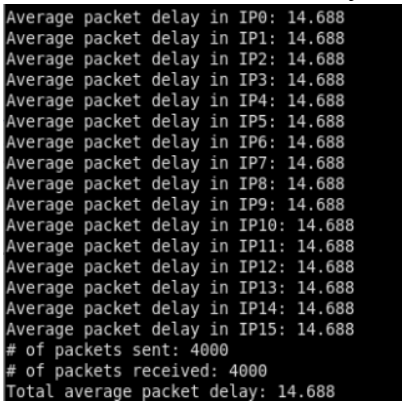

Fig. 15 Simulated torus with a neighbouring pattern

V. Conclusion

The data transfer speed of an NoC system is dependent on the communication pattern generated between the IP cores. In a uniform communication pattern, a NoC system experiences link congestion, increasing the duration in which a packet travels from a source to sink. For this pattern, the average packet delay of 4x4 mesh and 4x4 torus is 33 and 28 clock cycles, respectively. In contrast to a neighbouring pattern, the packet can freely traverse through a channel without any competition with other packets, reducing the packet delay significantly. The delay experienced for both the topologies is 14 clock cycles – almost half of the delay observed in the uniform.

VI. Appendix

A. source.cpp

```
// source.cpp
#define MESSAGE_SIZE 5
#define NUMBER_OF_MESSAGES 50

#include "source.h"
void source:: func()
        {
        packet v_packet_out;
        v_packet_out.data=1000; // e.g.
        v_packet_out.pkt_clk = '0';  // an imaginary clock for
packets

        while(true)
                {
                wait();
                clk_count++;
                if(!ach_in.read() and pkt_snt/MESSAGE_SIZE <
NUMBER_OF_MESSAGES)
                        {
                        //v_packet_out.data =
v_packet_out.data +source_id.read()+ 1  ; // made a desired data
      v_packet_out.data = clk_count;
                        v_packet_out.id = source_id.read();
                        v_packet_out.dest = dest_id.read();


                                        // assign
destination

                        if(v_packet_out.id ==
v_packet_out.dest) goto exclode;        // prevent from reciving flits
by itself
                        v_packet_out.pkt_clk=
~v_packet_out.pkt_clk ;  // add an imaginary clock to each flit
```

```
                        v_packet_out.h_t=false;
                        pkt_snt++;

                        // make tail flit (the packet size is 5)
                        if((pkt_snt%MESSAGE_SIZE)==0){
                                v_packet_out.h_t=true;
                        }
                        packet_out.write(v_packet_out);

                        cout << clk_count << " cycle: Source "
<< (int) v_packet_out.id << " sent packet with (dest = " << (int)
v_packet_out.dest  << ", data = "<< (int) v_packet_out.data << ")"
<<endl;
                        exclode:;
                        }
                }
        }
```

B. SINK.CPP

```
// sink.cpp
#include "sink.h"
void sink::receive_data(){

        packet v_packet;
        if ( sclk.event() ){ack_out.write(false); clk_count++;}
        if (packet_in.event() )
        {
                pkt_recv++ ;
                ack_out.write(true);
                v_packet= packet_in.read();
                cout << clk_count << " cycle: Sink " <<
(int)sink_id.read() << " received packet with (id = " << (int)
v_packet.id  << ", data = "<< (int) v_packet.data << ")" << endl;

    sum_delay += clk_count-(int)v_packet.data;

        }
}
```

C. ARBITER.CPP

```
//arbiter.cpp
#undef SC_INCLUDE_FX

#include "packet.h"
#include "arbiter.h"

void arbiter :: func()
{
        sc_uint<1> v_connected_input[5]; //set when input is
connected to an output
        sc_uint<1> v_reserved_output[6]; //set when output is
reserved by a input (one output more for simple coding)
        sc_uint<3> v_req[5];
        sc_uint<5> v_free;  // status of output in term of being free
        sc_uint<4> v_id;
        sc_uint<5> v_arbit;
        sc_uint<15> v_select;
        sc_uint<4> header0, header1, header2, header3, header4;

        for(int
i=0;i<5;i++){v_connected_input[i]=0;v_reserved_output[i]=0;v_req[
i]=0;}
        v_free = 31;  // '11111'
        v_arbit = 0;
        v_select = 0;
        int dest_x, dest_y;
        int id_x, id_y;


        // functionality

        while( true )
        {
                wait();
                grant0.write(0);
        // reset grant
                grant1.write(0);
        // reset grant
                grant2.write(0);
        // reset grant
                grant3.write(0);
        // reset grant
                grant4.write(0);
        // reset grant
                if (!free_out0.read()) {v_free = v_free | 1 ; }  //
set the bit 0 showing the output 0 is free
                if (!free_out1.read()) {v_free = v_free | 2 ; }
                if (!free_out2.read()) {v_free = v_free | 4 ; }
                if (!free_out3.read()) {v_free = v_free | 8 ; }
                if (!free_out4.read()) {v_free = v_free | 16 ;}

                v_id = arbiter_id.read();
                id_x = v_id%4;
                id_y = v_id/4;
                if (!req0.read()[4]) //if FIFO buffer is not empty
                {

                        if(!v_connected_input[0]){
                                header0 =
req0.read().range(3,0);
                        }

                        dest_x = header0%4;
                        dest_y = header0/4;

                        if(dest_x > id_x) v_req[0]=3; // go to
east
                        else if(dest_x  < id_x) v_req[0]=5; //go
to west
                        else if(dest_y < id_y) v_req[0]=2; //go
to north
```

```
                else if(dest_y > id_y) v_req[0]=4; // go to south
                else v_req[0]=1; // that is the destination

                switch (v_req[0]) {
                        case 1:     v_arbit=v_free & 1; break;
                        case 2: v_arbit=v_free &  2; break;
                        case 3: v_arbit=v_free &  4; break;
                        case 4: v_arbit=v_free &  8; break;
                        case 5: v_arbit=v_free &  16; break;
                        default: break ;
                }
                if(!v_connected_input[0]) // if input is not connected
                {
                        if (v_reserved_output[v_req[0]])v_arbit=0; // if the requested output was reserved, go to next input
                }
                if(v_arbit!=0){
                        grant0.write(1);
                // set grant
                        v_select.range(2,0) = v_req[0];
                        v_free  = v_free & (~v_arbit);   // inactive the related output
                        v_connected_input[0]=1; // input 0 is connected
                        v_reserved_output[v_req[0]]=1; // output is reserved
                        if(req0.read()[5]){
                        v_connected_input[0]=0;v_reserved_output[v_req[0]]=0;} // if it is tail flit, reset connection and reservation
                        }
                }
                if (!req1.read()[4]) //if buffer is not empty
                {
                        if(!v_connected_input[1]){
                                header1 = req1.read().range(3,0);
                        }

                        dest_x = header1%4;
                        dest_y = header1/4;

                        if(dest_x > id_x) v_req[1]=3; // go to east
                        else if(dest_x  < id_x) v_req[1]=5; //go to west
                        else if(dest_y < id_y) v_req[1]=2; //go to north
                        else if(dest_y > id_y) v_req[1]=4; // go to south
                        else v_req[1]=1; // that is the destination

                        switch (v_req[1]) {
                                case 1:     v_arbit=v_free & 1; break;
                                case 2: v_arbit=v_free &  2; break;
                                case 3: v_arbit=v_free &  4; break;
                                case 4: v_arbit=v_free &  8; break;
                                case 5: v_arbit=v_free &  16; break;
                                default: break ;
                        }
                        if(!v_connected_input[1]) // if input is not connected
                        {
                                if (v_reserved_output[v_req[1]])v_arbit=0; // if the requested output was reserved, go to next input
                        }
                        if(v_arbit!=0){
                        // if there is any free output
                                grant1.write(1);       // set grant
                                v_select.range(5,3) = v_req[1];
                                v_free  = v_free & (~v_arbit);   // inactive the related outputs
                                v_connected_input[1]=1; // input 1 is connected
                                v_reserved_output[v_req[1]]=1; // output is reserved
                                if(req1.read()[5]){v_connected_input[1]=0;v_reserved_output[v_req[1]]=0;} // if it is tail flit, reset connection and reservation
                        }
                }
                if (!req2.read()[4]) //if buffer is not empty
                {
                        if(!v_connected_input[2]){
                                header2 = req2.read().range(3,0);
                        }

                        dest_x = header2%4;
                        dest_y = header2/4;
```

```
                        if(dest_x > id_x) v_req[2]=3; // go to
east
                        else if(dest_x  < id_x) v_req[2]=5; //go
to west
                        else if(dest_y < id_y) v_req[2]=2; //go
to north
                        else if(dest_y > id_y) v_req[2]=4; // go
to south
                        else v_req[2]=1; // that is the
destination

                        switch (v_req[2]) {
                                case 1:     v_arbit=v_free &
1; break;
                                case 2: v_arbit=v_free &  2;
break;
                                case 3: v_arbit=v_free &  4;
break;
                                case 4: v_arbit=v_free &  8;
break;
                                case 5: v_arbit=v_free &  16;
break;
                                default: break ;
                        }
                        if(!v_connected_input[2]) // if input is
not connected
                        {
                                if
(v_reserved_output[v_req[2]])v_arbit=0; // if the requested output
was reserved, go to next input
                        }
                        if(v_arbit!=0){
                                grant2.write(1);       // set
grant
                                v_select.range(8,6) =
v_req[2];
                                v_free  = v_free &
(~v_arbit);   // inactive the related outputs
                                v_connected_input[2]=1; //
input 1 is connected

v_reserved_output[v_req[2]]=1; // output is reserved

if(req2.read()[5]){v_connected_input[2]=0;v_reserved_output[v_req[
2]]=0;} // if it is tail flit, reset connection and reservation
                        }
                }

                if (!req3.read()[4]) //if buffer is not empty
                {
                        if(!v_connected_input[3]){
                                header3 =
req3.read().range(3,0);
                        }

                        dest_x = header3%4;
                        dest_y = header3/4;

                        if(dest_x > id_x) v_req[3]=3; // go to
east
                        else if(dest_x  < id_x) v_req[3]=5; //go
to west
                        else if(dest_y < id_y) v_req[3]=2; //go
to north
                        else if(dest_y > id_y) v_req[3]=4; // go
to south
                        else v_req[3]=1; // that is the
destination

                        switch (v_req[3]) {
                                case 1:     v_arbit=v_free &
1; break;
                                case 2: v_arbit=v_free &  2;
break;
                                case 3: v_arbit=v_free &  4;
break;
                                case 4: v_arbit=v_free &  8;
break;
                                case 5: v_arbit=v_free &  16;
break;
                                default: break ;
                        }
                        if(!v_connected_input[3]) // if input is
not connected
                        {
                                if
(v_reserved_output[v_req[3]])v_arbit=0; // if the requested output
was reserved, go to next input
                        }
                        if(v_arbit!=0){
                                grant3.write(1);       // set
grant
                                v_select.range(11,9) =
v_req[3];
                                v_free  = v_free &
(~v_arbit);   // inactive the related outputs
                                v_connected_input[3]=1; //
input 3 is connected

v_reserved_output[v_req[3]]=1; // output is reserved

if(req3.read()[5]){v_connected_input[3]=0;v_reserved_output[v_req[
3]]=0;} // if it is tail flit, reset connection and reservation
                        }
                }

                if (!req4.read()[4]) //if buffer is not empty
                {
                        if(!v_connected_input[4]){
                                header4 =
req4.read().range(3,0);
                        }
```

```
                        dest_x = header4%4;
                        dest_y = header4/4;


                        if(dest_x > id_x) v_req[4]=3; // go to
east
                        else if(dest_x  < id_x) v_req[4]=5; //go
to west
                        else if(dest_y < id_y) v_req[4]=2; //go
to north
                        else if(dest_y > id_y) v_req[4]=4; // go
to south
                        else v_req[4]=1; // that is the
destination

                        switch (v_req[4]) {
                                case 1:     v_arbit=v_free &
1; break;
                                case 2: v_arbit=v_free &  2;
break;
                                case 3: v_arbit=v_free &  4;
break;
                                case 4: v_arbit=v_free &  8;
break;
                                case 5: v_arbit=v_free &  16;
break;
                                default: break ;
                        }
                        if(!v_connected_input[4]) // if input is
not connected
                        {
                                if
(v_reserved_output[v_req[4]])v_arbit=0; // if the requested output
was reserved, go to next input
                        }
                        if(v_arbit!=0){
                                grant4.write(1);       // set
grant
                                v_select.range(14,12) =
v_req[4];
                                v_free  = v_free &
(~v_arbit);   // inactive the related outputs
                                v_connected_input[4]=1; //
input 4 is connected

v_reserved_output[v_req[4]]=1; // output is reserved

if(req4.read()[5]){v_connected_input[4]=0;v_reserved_output[v_req[
4]]=0;} // if it is tail flit, reset connection and reservation
                        }
                }
        aselect.write(v_select);
        }
}
```

D. 4x4 MESH

```
// main.cpp
#define CLK_S 5

#include "systemc.h"
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include "packet.h"
#include "ip.h"


int sc_main(int argc, char *argv[])
{
//int dest[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,15,14};
//int dest[] = {15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0};
int dest[] = {1,0,3,2,5,4,7,6,9,8,11,10,13,12,15,14};
//int dest[] = {6, 2, 5, 14, 11, 3, 12, 1, 4, 15, 7, 0, 10, 8, 13, 9};


sc_signal<packet>
in_zeroData_0_0,si_data1,si_data9,in_zeroData_0_3,out_zeroData_0
_0,si_data0,si_data8,out_zeroData_0_3,in_zeroData_1_0,si_data17,s
i_data11,out_zeroData_1_0,si_data16,si_data10,in_zeroData_2_0,si_
data33,si_data13,out_zeroData_2_0,si_data32,si_data12,in_zeroData
_3_0,in_zeroData_3_1,si_data15,out_zeroData_3_0,out_zeroData_3
_1,si_data14,si_data3,si_data25,in_zeroData_4_3,si_data2,si_data24,
out_zeroData_4_3,si_data19,si_data27,si_data18,si_data26,si_data35
,si_data29,si_data34,si_data28,in_zeroData_7_1,si_data31,out_zero
Data_7_1,si_data30,si_data5,si_data41,in_zeroData_8_3,si_data4,si_
data40,out_zeroData_8_3,si_data21,si_data43,si_data20,si_data42,si
_data37,si_data45,si_data36,si_data44,in_zeroData_11_1,si_data47,o
ut_zeroData_11_1,si_data46,si_data7,in_zeroData_12_2,in_zeroData
_12_3,si_data6,out_zeroData_12_2,out_zeroData_12_3,si_data23,in
_zeroData_13_2,si_data22,out_zeroData_13_2,si_data39,in_zeroDat
a_14_2,si_data38,out_zeroData_14_2,in_zeroData_15_1,in_zeroDat
a_15_2,out_zeroData_15_1,out_zeroData_15_2;

sc_signal<bool>
in_zeroAck_0_0,si_ack1,si_ack9,in_zeroAck_0_3,out_zeroAck_0_0,
si_ack0,si_ack8,out_zeroAck_0_3,in_zeroAck_1_0,si_ack17,si_ack1
1,out_zeroAck_1_0,si_ack16,si_ack10,in_zeroAck_2_0,si_ack33,si_
ack13,out_zeroAck_2_0,si_ack32,si_ack12,in_zeroAck_3_0,in_zero
Ack_3_1,si_ack15,out_zeroAck_3_0,out_zeroAck_3_1,si_ack14,si_
ack3,si_ack25,in_zeroAck_4_3,si_ack2,si_ack24,out_zeroAck_4_3,s
i_ack19,si_ack27,si_ack18,si_ack26,si_ack35,si_ack29,si_ack34,si_a
ck28,in_zeroAck_7_1,si_ack31,out_zeroAck_7_1,si_ack30,si_ack5,s
i_ack41,in_zeroAck_8_3,si_ack4,si_ack40,out_zeroAck_8_3,si_ack2
1,si_ack43,si_ack20,si_ack42,si_ack37,si_ack45,si_ack36,si_ack44,i
n_zeroAck_11_1,si_ack47,out_zeroAck_11_1,si_ack46,si_ack7,in_z
eroAck_12_2,in_zeroAck_12_3,si_ack6,out_zeroAck_12_2,out_zero
Ack_12_3,si_ack23,in_zeroAck_13_2,si_ack22,out_zeroAck_13_2,s
i_ack39,in_zeroAck_14_2,si_ack38,out_zeroAck_14_2,in_zeroAck_
15_1,in_zeroAck_15_2,out_zeroAck_15_1,out_zeroAck_15_2;
```

```
sc_signal<sc_uint<4> >
id0,id1,id2,id3,id4,id5,id6,id7,id8,id9,id10,id11,id12,id13,id14,id15;

sc_signal<sc_uint<4> >
dest0,dest1,dest2,dest3,dest4,dest5,dest6,dest7,dest8,dest9,dest10,des
t11,dest12,dest13,dest14,dest15;

sc_clock myClock("myClock", CLK_S, SC_NS, 0.5, 0.0, SC_NS);

ip ip0("ip0");
ip0.id(id0);
ip0.clk(myClock);
ip0.in_dataN(in_zeroData_0_0);
ip0.in_ackN(in_zeroAck_0_0);
ip0.in_dataE(si_data1);
ip0.in_ackE(si_ack1);
ip0.in_dataS(si_data9);
ip0.in_ackS(si_ack9);
ip0.in_dataW(in_zeroData_0_3);
ip0.in_ackW(in_zeroAck_0_3);
ip0.out_dataN(out_zeroData_0_0);
ip0.out_ackN(out_zeroAck_0_0);
ip0.out_dataE(si_data0);
ip0.out_ackE(si_ack0);
ip0.out_dataS(si_data8);
ip0.out_ackS(si_ack8);
ip0.out_dataW(out_zeroData_0_3);
ip0.out_ackW(out_zeroAck_0_3);
ip0.dest_id(dest0);
id0.write(0);
dest0.write(dest[0]);

ip ip1("ip1");
ip1.id(id1);
ip1.clk(myClock);
ip1.in_dataN(in_zeroData_1_0);
ip1.in_ackN(in_zeroAck_1_0);
ip1.in_dataE(si_data17);
ip1.in_ackE(si_ack17);
ip1.in_dataS(si_data11);
ip1.in_ackS(si_ack11);
ip1.in_dataW(si_data0);
ip1.in_ackW(si_ack0);
ip1.out_dataN(out_zeroData_1_0);
ip1.out_ackN(out_zeroAck_1_0);
ip1.out_dataE(si_data16);
ip1.out_ackE(si_ack16);
ip1.out_dataS(si_data10);
ip1.out_ackS(si_ack10);
ip1.out_dataW(si_data1);
ip1.out_ackW(si_ack1);
ip1.dest_id(dest1);
id1.write(1);
dest1.write(dest[1]);

ip ip2("ip2");
ip2.id(id2);
ip2.clk(myClock);
ip2.in_dataN(in_zeroData_2_0);
ip2.in_ackN(in_zeroAck_2_0);
ip2.in_dataE(si_data33);
ip2.in_ackE(si_ack33);
ip2.in_dataS(si_data13);
ip2.in_ackS(si_ack13);
ip2.in_dataW(si_data16);
ip2.in_ackW(si_ack16);
ip2.out_dataN(out_zeroData_2_0);
ip2.out_ackN(out_zeroAck_2_0);
ip2.out_dataE(si_data32);
ip2.out_ackE(si_ack32);
ip2.out_dataS(si_data12);
ip2.out_ackS(si_ack12);
ip2.out_dataW(si_data17);
ip2.out_ackW(si_ack17);
ip2.dest_id(dest2);
id2.write(2);
dest2.write(dest[2]);

ip ip3("ip3");
ip3.id(id3);
ip3.clk(myClock);
ip3.in_dataN(in_zeroData_3_0);
ip3.in_ackN(in_zeroAck_3_0);
ip3.in_dataE(in_zeroData_3_1);
ip3.in_ackE(in_zeroAck_3_1);
ip3.in_dataS(si_data15);
ip3.in_ackS(si_ack15);
ip3.in_dataW(si_data32);
ip3.in_ackW(si_ack32);
ip3.out_dataN(out_zeroData_3_0);
ip3.out_ackN(out_zeroAck_3_0);
ip3.out_dataE(out_zeroData_3_1);
ip3.out_ackE(out_zeroAck_3_1);
ip3.out_dataS(si_data14);
ip3.out_ackS(si_ack14);
ip3.out_dataW(si_data33);
ip3.out_ackW(si_ack33);
ip3.dest_id(dest3);
id3.write(3);
dest3.write(dest[3]);

ip ip4("ip4");
ip4.id(id4);
ip4.clk(myClock);
ip4.in_dataN(si_data8);
ip4.in_ackN(si_ack8);
ip4.in_dataE(si_data3);
ip4.in_ackE(si_ack3);
ip4.in_dataS(si_data25);
ip4.in_ackS(si_ack25);
ip4.in_dataW(in_zeroData_4_3);
```

```
ip4.in_ackW(in_zeroAck_4_3);
ip4.out_dataN(si_data9);
ip4.out_ackN(si_ack9);
ip4.out_dataE(si_data2);
ip4.out_ackE(si_ack2);
ip4.out_dataS(si_data24);
ip4.out_ackS(si_ack24);
ip4.out_dataW(out_zeroData_4_3);
ip4.out_ackW(out_zeroAck_4_3);
ip4.dest_id(dest4);
id4.write(4);
dest4.write(dest[4]);

ip ip5("ip5");
ip5.id(id5);
ip5.clk(myClock);
ip5.in_dataN(si_data10);
ip5.in_ackN(si_ack10);
ip5.in_dataE(si_data19);
ip5.in_ackE(si_ack19);
ip5.in_dataS(si_data27);
ip5.in_ackS(si_ack27);
ip5.in_dataW(si_data2);
ip5.in_ackW(si_ack2);
ip5.out_dataN(si_data11);
ip5.out_ackN(si_ack11);
ip5.out_dataE(si_data18);
ip5.out_ackE(si_ack18);
ip5.out_dataS(si_data26);
ip5.out_ackS(si_ack26);
ip5.out_dataW(si_data3);
ip5.out_ackW(si_ack3);
ip5.dest_id(dest5);
id5.write(5);
dest5.write(dest[5]);

ip ip6("ip6");
ip6.id(id6);
ip6.clk(myClock);
ip6.in_dataN(si_data12);
ip6.in_ackN(si_ack12);
ip6.in_dataE(si_data35);
ip6.in_ackE(si_ack35);
ip6.in_dataS(si_data29);
ip6.in_ackS(si_ack29);
ip6.in_dataW(si_data18);
ip6.in_ackW(si_ack18);
ip6.out_dataN(si_data13);
ip6.out_ackN(si_ack13);
ip6.out_dataE(si_data34);
ip6.out_ackE(si_ack34);
ip6.out_dataS(si_data28);
ip6.out_ackS(si_ack28);
ip6.out_dataW(si_data19);
ip6.out_ackW(si_ack19);
ip6.dest_id(dest6);
id6.write(6);
dest6.write(dest[6]);

ip ip7("ip7");
ip7.id(id7);
ip7.clk(myClock);
ip7.in_dataN(si_data14);
ip7.in_ackN(si_ack14);
ip7.in_dataE(in_zeroData_7_1);
ip7.in_ackE(in_zeroAck_7_1);
ip7.in_dataS(si_data31);
ip7.in_ackS(si_ack31);
ip7.in_dataW(si_data34);
ip7.in_ackW(si_ack34);
ip7.out_dataN(si_data15);
ip7.out_ackN(si_ack15);
ip7.out_dataE(out_zeroData_7_1);
ip7.out_ackE(out_zeroAck_7_1);
ip7.out_dataS(si_data30);
ip7.out_ackS(si_ack30);
ip7.out_dataW(si_data35);
ip7.out_ackW(si_ack35);
ip7.dest_id(dest7);
id7.write(7);
dest7.write(dest[7]);

ip ip8("ip8");
ip8.id(id8);
ip8.clk(myClock);
ip8.in_dataN(si_data24);
ip8.in_ackN(si_ack24);
ip8.in_dataE(si_data5);
ip8.in_ackE(si_ack5);
ip8.in_dataS(si_data41);
ip8.in_ackS(si_ack41);
ip8.in_dataW(in_zeroData_8_3);
ip8.in_ackW(in_zeroAck_8_3);
ip8.out_dataN(si_data25);
ip8.out_ackN(si_ack25);
ip8.out_dataE(si_data4);
ip8.out_ackE(si_ack4);
ip8.out_dataS(si_data40);
ip8.out_ackS(si_ack40);
ip8.out_dataW(out_zeroData_8_3);
ip8.out_ackW(out_zeroAck_8_3);
ip8.dest_id(dest8);
id8.write(8);
dest8.write(dest[8]);

ip ip9("ip9");
ip9.id(id9);
ip9.clk(myClock);
ip9.in_dataN(si_data26);
ip9.in_ackN(si_ack26);
ip9.in_dataE(si_data21);
ip9.in_ackE(si_ack21);
```

```
ip9.in_dataS(si_data43);
ip9.in_ackS(si_ack43);
ip9.in_dataW(si_data4);
ip9.in_ackW(si_ack4);
ip9.out_dataN(si_data27);
ip9.out_ackN(si_ack27);
ip9.out_dataE(si_data20);
ip9.out_ackE(si_ack20);
ip9.out_dataS(si_data42);
ip9.out_ackS(si_ack42);
ip9.out_dataW(si_data5);
ip9.out_ackW(si_ack5);
ip9.dest_id(dest9);
id9.write(9);
dest9.write(dest[9]);

ip ip10("ip10");
ip10.id(id10);
ip10.clk(myClock);
ip10.in_dataN(si_data28);
ip10.in_ackN(si_ack28);
ip10.in_dataE(si_data37);
ip10.in_ackE(si_ack37);
ip10.in_dataS(si_data45);
ip10.in_ackS(si_ack45);
ip10.in_dataW(si_data20);
ip10.in_ackW(si_ack20);
ip10.out_dataN(si_data29);
ip10.out_ackN(si_ack29);
ip10.out_dataE(si_data36);
ip10.out_ackE(si_ack36);
ip10.out_dataS(si_data44);
ip10.out_ackS(si_ack44);
ip10.out_dataW(si_data21);
ip10.out_ackW(si_ack21);
ip10.dest_id(dest10);
id10.write(10);
dest10.write(dest[10]);

ip ip11("ip11");
ip11.id(id11);
ip11.clk(myClock);
ip11.in_dataN(si_data30);
ip11.in_ackN(si_ack30);
ip11.in_dataE(in_zeroData_11_1);
ip11.in_ackE(in_zeroAck_11_1);
ip11.in_dataS(si_data47);
ip11.in_ackS(si_ack47);
ip11.in_dataW(si_data36);
ip11.in_ackW(si_ack36);
ip11.out_dataN(si_data31);
ip11.out_ackN(si_ack31);
ip11.out_dataE(out_zeroData_11_1);
ip11.out_ackE(out_zeroAck_11_1);
ip11.out_dataS(si_data46);
ip11.out_ackS(si_ack46);
ip11.out_dataW(si_data37);
ip11.out_ackW(si_ack37);
ip11.dest_id(dest11);
id11.write(11);
dest11.write(dest[11]);

ip ip12("ip12");
ip12.id(id12);
ip12.clk(myClock);
ip12.in_dataN(si_data40);
ip12.in_ackN(si_ack40);
ip12.in_dataE(si_data7);
ip12.in_ackE(si_ack7);
ip12.in_dataS(in_zeroData_12_2);
ip12.in_ackS(in_zeroAck_12_2);
ip12.in_dataW(in_zeroData_12_3);
ip12.in_ackW(in_zeroAck_12_3);
ip12.out_dataN(si_data41);
ip12.out_ackN(si_ack41);
ip12.out_dataE(si_data6);
ip12.out_ackE(si_ack6);
ip12.out_dataS(out_zeroData_12_2);
ip12.out_ackS(out_zeroAck_12_2);
ip12.out_dataW(out_zeroData_12_3);
ip12.out_ackW(out_zeroAck_12_3);
ip12.dest_id(dest12);
id12.write(12);
dest12.write(dest[12]);

ip ip13("ip13");
ip13.id(id13);
ip13.clk(myClock);
ip13.in_dataN(si_data42);
ip13.in_ackN(si_ack42);
ip13.in_dataE(si_data23);
ip13.in_ackE(si_ack23);
ip13.in_dataS(in_zeroData_13_2);
ip13.in_ackS(in_zeroAck_13_2);
ip13.in_dataW(si_data6);
ip13.in_ackW(si_ack6);
ip13.out_dataN(si_data43);
ip13.out_ackN(si_ack43);
ip13.out_dataE(si_data22);
ip13.out_ackE(si_ack22);
ip13.out_dataS(out_zeroData_13_2);
ip13.out_ackS(out_zeroAck_13_2);
ip13.out_dataW(si_data7);
ip13.out_ackW(si_ack7);
ip13.dest_id(dest13);
id13.write(13);
dest13.write(dest[13]);

ip ip14("ip14");
ip14.id(id14);
ip14.clk(myClock);
ip14.in_dataN(si_data44);
```

```
ip14.in_ackN(si_ack44);
ip14.in_dataE(si_data39);
ip14.in_ackE(si_ack39);
ip14.in_dataS(in_zeroData_14_2);
ip14.in_ackS(in_zeroAck_14_2);
ip14.in_dataW(si_data22);
ip14.in_ackW(si_ack22);
ip14.out_dataN(si_data45);
ip14.out_ackN(si_ack45);
ip14.out_dataE(si_data38);
ip14.out_ackE(si_ack38);
ip14.out_dataS(out_zeroData_14_2);
ip14.out_ackS(out_zeroAck_14_2);
ip14.out_dataW(si_data23);
ip14.out_ackW(si_ack23);
ip14.dest_id(dest14);
id14.write(14);
dest14.write(dest[14]);

ip ip15("ip15");
ip15.id(id15);
ip15.clk(myClock);
ip15.in_dataN(si_data46);
ip15.in_ackN(si_ack46);
ip15.in_dataE(in_zeroData_15_1);
ip15.in_ackE(in_zeroAck_15_1);
ip15.in_dataS(in_zeroData_15_2);
ip15.in_ackS(in_zeroAck_15_2);
ip15.in_dataW(si_data38);
ip15.in_ackW(si_ack38);
ip15.out_dataN(si_data47);
ip15.out_ackN(si_ack47);
ip15.out_dataE(out_zeroData_15_1);
ip15.out_ackE(out_zeroAck_15_1);
ip15.out_dataS(out_zeroData_15_2);
ip15.out_ackS(out_zeroAck_15_2);
ip15.out_dataW(si_data39);
ip15.out_ackW(si_ack39);
ip15.dest_id(dest15);
id15.write(15);
dest15.write(dest[15]);

        sc_trace_file *tf = sc_create_vcd_trace_file("graph");
        sc_trace(tf, myClock, "clk");

        sc_trace(tf, si_data4, "data");

        cout << endl;
        cout <<
"--------------------------------------------------------------------------------"
<< endl;
        cout << "  Press \"Return\" key to start the simulation..." <<
endl << endl;

        getchar();
        sc_start(10000, SC_NS);
```

```
float sum = 0;
int N = 0;
cout << endl;
if(ip0.mySink->pkt_recv != 0){cout << "Average packet delay in
IP0: " << 1.0*ip0.mySink->sum_delay/ip0.mySink->pkt_recv <<
endl;
sum += 1.0*ip0.mySink->sum_delay/ip0.mySink->pkt_recv;
N++;};
if(ip1.mySink->pkt_recv != 0){cout << "Average packet delay in
IP1: " << 1.0*ip1.mySink->sum_delay/ip1.mySink->pkt_recv <<
endl;
sum += 1.0*ip1.mySink->sum_delay/ip1.mySink->pkt_recv;
N++;};
if(ip2.mySink->pkt_recv != 0){cout << "Average packet delay in
IP2: " << 1.0*ip2.mySink->sum_delay/ip2.mySink->pkt_recv <<
endl;
sum += 1.0*ip2.mySink->sum_delay/ip2.mySink->pkt_recv;
N++;};
if(ip3.mySink->pkt_recv != 0){cout << "Average packet delay in
IP3: " << 1.0*ip3.mySink->sum_delay/ip3.mySink->pkt_recv <<
endl;
sum += 1.0*ip3.mySink->sum_delay/ip3.mySink->pkt_recv;
N++;};
if(ip4.mySink->pkt_recv != 0){cout << "Average packet delay in
IP4: " << 1.0*ip4.mySink->sum_delay/ip4.mySink->pkt_recv <<
endl;
sum += 1.0*ip4.mySink->sum_delay/ip4.mySink->pkt_recv;
N++;};
if(ip5.mySink->pkt_recv != 0){cout << "Average packet delay in
IP5: " << 1.0*ip5.mySink->sum_delay/ip5.mySink->pkt_recv <<
endl;
sum += 1.0*ip5.mySink->sum_delay/ip5.mySink->pkt_recv;
N++;};
if(ip6.mySink->pkt_recv != 0){cout << "Average packet delay in
IP6: " << 1.0*ip6.mySink->sum_delay/ip6.mySink->pkt_recv <<
endl;
sum += 1.0*ip6.mySink->sum_delay/ip6.mySink->pkt_recv;
N++;};
if(ip7.mySink->pkt_recv != 0){cout << "Average packet delay in
IP7: " << 1.0*ip7.mySink->sum_delay/ip7.mySink->pkt_recv <<
endl;
sum += 1.0*ip7.mySink->sum_delay/ip7.mySink->pkt_recv;
N++;};
if(ip8.mySink->pkt_recv != 0){cout << "Average packet delay in
IP8: " << 1.0*ip8.mySink->sum_delay/ip8.mySink->pkt_recv <<
endl;
sum += 1.0*ip8.mySink->sum_delay/ip8.mySink->pkt_recv;
N++;};
if(ip9.mySink->pkt_recv != 0){cout << "Average packet delay in
IP9: " << 1.0*ip9.mySink->sum_delay/ip9.mySink->pkt_recv <<
endl;
sum += 1.0*ip9.mySink->sum_delay/ip9.mySink->pkt_recv;
N++;};
```

```
if(ip10.mySink->pkt_recv != 0){cout << "Average packet delay in
IP10: " << 1.0*ip10.mySink->sum_delay/ip10.mySink->pkt_recv <<
endl;
sum += 1.0*ip10.mySink->sum_delay/ip10.mySink->pkt_recv;
N++;};
if(ip11.mySink->pkt_recv != 0){cout << "Average packet delay in
IP11: " << 1.0*ip11.mySink->sum_delay/ip11.mySink->pkt_recv <<
endl;
sum += 1.0*ip11.mySink->sum_delay/ip11.mySink->pkt_recv;
N++;};
if(ip12.mySink->pkt_recv != 0){cout << "Average packet delay in
IP12: " << 1.0*ip12.mySink->sum_delay/ip12.mySink->pkt_recv <<
endl;
sum += 1.0*ip12.mySink->sum_delay/ip12.mySink->pkt_recv;
N++;};
if(ip13.mySink->pkt_recv != 0){cout << "Average packet delay in
IP13: " << 1.0*ip13.mySink->sum_delay/ip13.mySink->pkt_recv <<
endl;
sum += 1.0*ip13.mySink->sum_delay/ip13.mySink->pkt_recv;
N++;};
if(ip14.mySink->pkt_recv != 0){cout << "Average packet delay in
IP14: " << 1.0*ip14.mySink->sum_delay/ip14.mySink->pkt_recv <<
endl;
sum += 1.0*ip14.mySink->sum_delay/ip14.mySink->pkt_recv;
N++;};
if(ip15.mySink->pkt_recv != 0){cout << "Average packet delay in
IP15: " << 1.0*ip15.mySink->sum_delay/ip15.mySink->pkt_recv <<
endl;
sum += 1.0*ip15.mySink->sum_delay/ip15.mySink->pkt_recv;
N++;};
cout << "# of packets sent: " <<
ip0.mySource->pkt_snt+ip1.mySource->pkt_snt+ip2.mySource->pkt
_snt+ip3.mySource->pkt_snt+ip4.mySource->pkt_snt+ip5.mySource
->pkt_snt+ip6.mySource->pkt_snt+ip7.mySource->pkt_snt+ip8.mySource->pkt_snt+ip9.mySource->pkt_snt+ip10.mySource->pkt_snt+ip
11.mySource->pkt_snt+ip12.mySource->pkt_snt+ip13.mySource->p
kt_snt+ip14.mySource->pkt_snt+ip15.mySource->pkt_snt << endl;
cout << "# of packets received: " <<
ip0.mySink->pkt_recv+ip1.mySink->pkt_recv+ip2.mySink->pkt_rec
v+ip3.mySink->pkt_recv+ip4.mySink->pkt_recv+ip5.mySink->pkt_r
ecv+ip6.mySink->pkt_recv+ip7.mySink->pkt_recv+ip8.mySink->pk
t_recv+ip9.mySink->pkt_recv+ip10.mySink->pkt_recv+ip11.mySink
->pkt_recv+ip12.mySink->pkt_recv+ip13.mySink->pkt_recv+ip14.
mySink->pkt_recv+ip15.mySink->pkt_recv << endl;
if(N != 0) cout << "Total average packet delay: " << sum/N << endl;

            sc_close_vcd_trace_file(tf);

            cout << endl << endl <<
"--------------------------------------------------------------------------------"
<< endl;
   cout << "  Press \"Return\" key to end the simulation..." << endl
<< endl;
            getchar();
 return 0;
}
```

E. 4x4 TORUS

```
// main.cpp
#define CLK_S 5

#include "systemc.h"
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include "packet.h"
#include "ip.h"


int sc_main(int argc, char *argv[])
{
//int dest[] = {15,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
//int dest[] = {15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0};
//int dest[] = {1,0,3,2,5,4,7,6,9,8,11,10,13,12,15,14};
int dest[] = {6, 2, 5, 14, 11, 3, 12, 1, 4, 15, 7, 0, 10, 8, 13, 9};

sc_signal<packet>
si_data50,si_data1,si_data9,si_data48,si_data51,si_data0,si_data8,si_
data49,si_data54,si_data17,si_data11,si_data55,si_data16,si_data10,s
i_data58,si_data33,si_data13,si_data59,si_data32,si_data12,si_data6
2,si_data15,si_data63,si_data14,si_data3,si_data25,si_data52,si_data
2,si_data24,si_data53,si_data19,si_data27,si_data18,si_data26,si_dat
a35,si_data29,si_data34,si_data28,si_data31,si_data30,si_data5,si_da
ta41,si_data56,si_data4,si_data40,si_data57,si_data21,si_data43,si_d
ata20,si_data42,si_data37,si_data45,si_data36,si_data44,si_data47,si
_data46,si_data7,si_data60,si_data6,si_data61,si_data23,si_data22,si
_data39,si_data38;

sc_signal<bool>
si_ack50,si_ack1,si_ack9,si_ack48,si_ack51,si_ack0,si_ack8,si_ack4
9,si_ack54,si_ack17,si_ack11,si_ack55,si_ack16,si_ack10,si_ack58,s
i_ack33,si_ack13,si_ack59,si_ack32,si_ack12,si_ack62,si_ack15,si_a
ck63,si_ack14,si_ack3,si_ack25,si_ack52,si_ack2,si_ack24,si_ack53,
si_ack19,si_ack27,si_ack18,si_ack26,si_ack35,si_ack29,si_ack34,si_
ack28,si_ack31,si_ack30,si_ack5,si_ack41,si_ack56,si_ack4,si_ack4
0,si_ack57,si_ack21,si_ack43,si_ack20,si_ack42,si_ack37,si_ack45,s
i_ack36,si_ack44,si_ack47,si_ack46,si_ack7,si_ack60,si_ack6,si_ack
61,si_ack23,si_ack22,si_ack39,si_ack38;

sc_signal<sc_uint<4> >
id0,id1,id2,id3,id4,id5,id6,id7,id8,id9,id10,id11,id12,id13,id14,id15;

sc_signal<sc_uint<4> >
dest0,dest1,dest2,dest3,dest4,dest5,dest6,dest7,dest8,dest9,dest10,des
t11,dest12,dest13,dest14,dest15;

sc_clock myClock("myClock", CLK_S, SC_NS, 0.5, 0.0, SC_NS);

ip ip0("ip0");
ip0.id(id0);
ip0.clk(myClock);
```

```
ip0.in_dataN(si_data50);
ip0.in_ackN(si_ack50);
ip0.in_dataE(si_data1);
ip0.in_ackE(si_ack1);
ip0.in_dataS(si_data9);
ip0.in_ackS(si_ack9);
ip0.in_dataW(si_data48);
ip0.in_ackW(si_ack48);
ip0.out_dataN(si_data51);
ip0.out_ackN(si_ack51);
ip0.out_dataE(si_data0);
ip0.out_ackE(si_ack0);
ip0.out_dataS(si_data8);
ip0.out_ackS(si_ack8);
ip0.out_dataW(si_data49);
ip0.out_ackW(si_ack49);
ip0.dest_id(dest0);
id0.write(0);
dest0.write(dest[0]);

ip ip1("ip1");
ip1.id(id1);
ip1.clk(myClock);
ip1.in_dataN(si_data54);
ip1.in_ackN(si_ack54);
ip1.in_dataE(si_data17);
ip1.in_ackE(si_ack17);
ip1.in_dataS(si_data11);
ip1.in_ackS(si_ack11);
ip1.in_dataW(si_data0);
ip1.in_ackW(si_ack0);
ip1.out_dataN(si_data55);
ip1.out_ackN(si_ack55);
ip1.out_dataE(si_data16);
ip1.out_ackE(si_ack16);
ip1.out_dataS(si_data10);
ip1.out_ackS(si_ack10);
ip1.out_dataW(si_data1);
ip1.out_ackW(si_ack1);
ip1.dest_id(dest1);
id1.write(1);
dest1.write(dest[1]);

ip ip2("ip2");
ip2.id(id2);
ip2.clk(myClock);
ip2.in_dataN(si_data58);
ip2.in_ackN(si_ack58);
ip2.in_dataE(si_data33);
ip2.in_ackE(si_ack33);
ip2.in_dataS(si_data13);
ip2.in_ackS(si_ack13);
ip2.in_dataW(si_data16);
ip2.in_ackW(si_ack16);
ip2.out_dataN(si_data59);
ip2.out_ackN(si_ack59);
ip2.out_dataE(si_data32);
ip2.out_ackE(si_ack32);
ip2.out_dataS(si_data12);
ip2.out_ackS(si_ack12);
ip2.out_dataW(si_data17);
ip2.out_ackW(si_ack17);
ip2.dest_id(dest2);
id2.write(2);
dest2.write(dest[2]);

ip ip3("ip3");
ip3.id(id3);
ip3.clk(myClock);
ip3.in_dataN(si_data62);
ip3.in_ackN(si_ack62);
ip3.in_dataE(si_data49);
ip3.in_ackE(si_ack49);
ip3.in_dataS(si_data15);
ip3.in_ackS(si_ack15);
ip3.in_dataW(si_data32);
ip3.in_ackW(si_ack32);
ip3.out_dataN(si_data63);
ip3.out_ackN(si_ack63);
ip3.out_dataE(si_data48);
ip3.out_ackE(si_ack48);
ip3.out_dataS(si_data14);
ip3.out_ackS(si_ack14);
ip3.out_dataW(si_data33);
ip3.out_ackW(si_ack33);
ip3.dest_id(dest3);
id3.write(3);
dest3.write(dest[3]);

ip ip4("ip4");
ip4.id(id4);
ip4.clk(myClock);
ip4.in_dataN(si_data8);
ip4.in_ackN(si_ack8);
ip4.in_dataE(si_data3);
ip4.in_ackE(si_ack3);
ip4.in_dataS(si_data25);
ip4.in_ackS(si_ack25);
ip4.in_dataW(si_data52);
ip4.in_ackW(si_ack52);
ip4.out_dataN(si_data9);
ip4.out_ackN(si_ack9);
ip4.out_dataE(si_data2);
ip4.out_ackE(si_ack2);
ip4.out_dataS(si_data24);
ip4.out_ackS(si_ack24);
ip4.out_dataW(si_data53);
ip4.out_ackW(si_ack53);
ip4.dest_id(dest4);
id4.write(4);
dest4.write(dest[4]);
```

```
ip ip5("ip5");
ip5.id(id5);
ip5.clk(myClock);
ip5.in_dataN(si_data10);
ip5.in_ackN(si_ack10);
ip5.in_dataE(si_data19);
ip5.in_ackE(si_ack19);
ip5.in_dataS(si_data27);
ip5.in_ackS(si_ack27);
ip5.in_dataW(si_data2);
ip5.in_ackW(si_ack2);
ip5.out_dataN(si_data11);
ip5.out_ackN(si_ack11);
ip5.out_dataE(si_data18);
ip5.out_ackE(si_ack18);
ip5.out_dataS(si_data26);
ip5.out_ackS(si_ack26);
ip5.out_dataW(si_data3);
ip5.out_ackW(si_ack3);
ip5.dest_id(dest5);
id5.write(5);
dest5.write(dest[5]);

ip ip6("ip6");
ip6.id(id6);
ip6.clk(myClock);
ip6.in_dataN(si_data12);
ip6.in_ackN(si_ack12);
ip6.in_dataE(si_data35);
ip6.in_ackE(si_ack35);
ip6.in_dataS(si_data29);
ip6.in_ackS(si_ack29);
ip6.in_dataW(si_data18);
ip6.in_ackW(si_ack18);
ip6.out_dataN(si_data13);
ip6.out_ackN(si_ack13);
ip6.out_dataE(si_data34);
ip6.out_ackE(si_ack34);
ip6.out_dataS(si_data28);
ip6.out_ackS(si_ack28);
ip6.out_dataW(si_data19);
ip6.out_ackW(si_ack19);
ip6.dest_id(dest6);
id6.write(6);
dest6.write(dest[6]);

ip ip7("ip7");
ip7.id(id7);
ip7.clk(myClock);
ip7.in_dataN(si_data14);
ip7.in_ackN(si_ack14);
ip7.in_dataE(si_data53);
ip7.in_ackE(si_ack53);
ip7.in_dataS(si_data31);
ip7.in_ackS(si_ack31);
ip7.in_dataW(si_data34);
ip7.in_ackW(si_ack34);
ip7.out_dataN(si_data15);
ip7.out_ackN(si_ack15);
ip7.out_dataE(si_data52);
ip7.out_ackE(si_ack52);
ip7.out_dataS(si_data30);
ip7.out_ackS(si_ack30);
ip7.out_dataW(si_data35);
ip7.out_ackW(si_ack35);
ip7.dest_id(dest7);
id7.write(7);
dest7.write(dest[7]);

ip ip8("ip8");
ip8.id(id8);
ip8.clk(myClock);
ip8.in_dataN(si_data24);
ip8.in_ackN(si_ack24);
ip8.in_dataE(si_data5);
ip8.in_ackE(si_ack5);
ip8.in_dataS(si_data41);
ip8.in_ackS(si_ack41);
ip8.in_dataW(si_data56);
ip8.in_ackW(si_ack56);
ip8.out_dataN(si_data25);
ip8.out_ackN(si_ack25);
ip8.out_dataE(si_data4);
ip8.out_ackE(si_ack4);
ip8.out_dataS(si_data40);
ip8.out_ackS(si_ack40);
ip8.out_dataW(si_data57);
ip8.out_ackW(si_ack57);
ip8.dest_id(dest8);
id8.write(8);
dest8.write(dest[8]);

ip ip9("ip9");
ip9.id(id9);
ip9.clk(myClock);
ip9.in_dataN(si_data26);
ip9.in_ackN(si_ack26);
ip9.in_dataE(si_data21);
ip9.in_ackE(si_ack21);
ip9.in_dataS(si_data43);
ip9.in_ackS(si_ack43);
ip9.in_dataW(si_data4);
ip9.in_ackW(si_ack4);
ip9.out_dataN(si_data27);
ip9.out_ackN(si_ack27);
ip9.out_dataE(si_data20);
ip9.out_ackE(si_ack20);
ip9.out_dataS(si_data42);
ip9.out_ackS(si_ack42);
ip9.out_dataW(si_data5);
ip9.out_ackW(si_ack5);
ip9.dest_id(dest9);
```

```
id9.write(9);
dest9.write(dest[9]);

ip ip10("ip10");
ip10.id(id10);
ip10.clk(myClock);
ip10.in_dataN(si_data28);
ip10.in_ackN(si_ack28);
ip10.in_dataE(si_data37);
ip10.in_ackE(si_ack37);
ip10.in_dataS(si_data45);
ip10.in_ackS(si_ack45);
ip10.in_dataW(si_data20);
ip10.in_ackW(si_ack20);
ip10.out_dataN(si_data29);
ip10.out_ackN(si_ack29);
ip10.out_dataE(si_data36);
ip10.out_ackE(si_ack36);
ip10.out_dataS(si_data44);
ip10.out_ackS(si_ack44);
ip10.out_dataW(si_data21);
ip10.out_ackW(si_ack21);
ip10.dest_id(dest10);
id10.write(10);
dest10.write(dest[10]);

ip ip11("ip11");
ip11.id(id11);
ip11.clk(myClock);
ip11.in_dataN(si_data30);
ip11.in_ackN(si_ack30);
ip11.in_dataE(si_data57);
ip11.in_ackE(si_ack57);
ip11.in_dataS(si_data47);
ip11.in_ackS(si_ack47);
ip11.in_dataW(si_data36);
ip11.in_ackW(si_ack36);
ip11.out_dataN(si_data31);
ip11.out_ackN(si_ack31);
ip11.out_dataE(si_data56);
ip11.out_ackE(si_ack56);
ip11.out_dataS(si_data46);
ip11.out_ackS(si_ack46);
ip11.out_dataW(si_data37);
ip11.out_ackW(si_ack37);
ip11.dest_id(dest11);
id11.write(11);
dest11.write(dest[11]);

ip ip12("ip12");
ip12.id(id12);
ip12.clk(myClock);
ip12.in_dataN(si_data40);
ip12.in_ackN(si_ack40);
ip12.in_dataE(si_data7);
ip12.in_ackE(si_ack7);
ip12.in_dataS(si_data51);
ip12.in_ackS(si_ack51);
ip12.in_dataW(si_data60);
ip12.in_ackW(si_ack60);
ip12.out_dataN(si_data41);
ip12.out_ackN(si_ack41);
ip12.out_dataE(si_data6);
ip12.out_ackE(si_ack6);
ip12.out_dataS(si_data50);
ip12.out_ackS(si_ack50);
ip12.out_dataW(si_data61);
ip12.out_ackW(si_ack61);
ip12.dest_id(dest12);
id12.write(12);
dest12.write(dest[12]);

ip ip13("ip13");
ip13.id(id13);
ip13.clk(myClock);
ip13.in_dataN(si_data42);
ip13.in_ackN(si_ack42);
ip13.in_dataE(si_data23);
ip13.in_ackE(si_ack23);
ip13.in_dataS(si_data55);
ip13.in_ackS(si_ack55);
ip13.in_dataW(si_data6);
ip13.in_ackW(si_ack6);
ip13.out_dataN(si_data43);
ip13.out_ackN(si_ack43);
ip13.out_dataE(si_data22);
ip13.out_ackE(si_ack22);
ip13.out_dataS(si_data54);
ip13.out_ackS(si_ack54);
ip13.out_dataW(si_data7);
ip13.out_ackW(si_ack7);
ip13.dest_id(dest13);
id13.write(13);
dest13.write(dest[13]);

ip ip14("ip14");
ip14.id(id14);
ip14.clk(myClock);
ip14.in_dataN(si_data44);
ip14.in_ackN(si_ack44);
ip14.in_dataE(si_data39);
ip14.in_ackE(si_ack39);
ip14.in_dataS(si_data59);
ip14.in_ackS(si_ack59);
ip14.in_dataW(si_data22);
ip14.in_ackW(si_ack22);
ip14.out_dataN(si_data45);
ip14.out_ackN(si_ack45);
ip14.out_dataE(si_data38);
ip14.out_ackE(si_ack38);
ip14.out_dataS(si_data58);
ip14.out_ackS(si_ack58);
```

```
ip14.out_dataW(si_data23);
ip14.out_ackW(si_ack23);
ip14.dest_id(dest14);
id14.write(14);
dest14.write(dest[14]);

ip ip15("ip15");
ip15.id(id15);
ip15.clk(myClock);
ip15.in_dataN(si_data46);
ip15.in_ackN(si_ack46);
ip15.in_dataE(si_data61);
ip15.in_ackE(si_ack61);
ip15.in_dataS(si_data63);
ip15.in_ackS(si_ack63);
ip15.in_dataW(si_data38);
ip15.in_ackW(si_ack38);
ip15.out_dataN(si_data47);
ip15.out_ackN(si_ack47);
ip15.out_dataE(si_data60);
ip15.out_ackE(si_ack60);
ip15.out_dataS(si_data62);
ip15.out_ackS(si_ack62);
ip15.out_dataW(si_data39);
ip15.out_ackW(si_ack39);
ip15.dest_id(dest15);
id15.write(15);
dest15.write(dest[15]);

        sc_trace_file *tf = sc_create_vcd_trace_file("graph");
        sc_trace(tf, myClock, "clk");

        sc_trace(tf, si_data4, "data");

        cout << endl;
        cout <<
"--------------------------------------------------------------------------------"
<< endl;
        cout << "  Press \"Return\" key to start the simulation..." <<
endl << endl;

        getchar();
        sc_start(10000, SC_NS);

float sum = 0;
int N = 0;
cout << endl;
if(ip0.mySink->pkt_recv != 0){cout << "Average packet delay in
IP0: " << 1.0*ip0.mySink->sum_delay/ip0.mySink->pkt_recv <<
endl;
sum += 1.0*ip0.mySink->sum_delay/ip0.mySink->pkt_recv;
N++;};
if(ip1.mySink->pkt_recv != 0){cout << "Average packet delay in
IP1: " << 1.0*ip1.mySink->sum_delay/ip1.mySink->pkt_recv <<
endl;
sum += 1.0*ip1.mySink->sum_delay/ip1.mySink->pkt_recv;
N++;};
if(ip2.mySink->pkt_recv != 0){cout << "Average packet delay in
IP2: " << 1.0*ip2.mySink->sum_delay/ip2.mySink->pkt_recv <<
endl;
sum += 1.0*ip2.mySink->sum_delay/ip2.mySink->pkt_recv;
N++;};
if(ip3.mySink->pkt_recv != 0){cout << "Average packet delay in
IP3: " << 1.0*ip3.mySink->sum_delay/ip3.mySink->pkt_recv <<
endl;
sum += 1.0*ip3.mySink->sum_delay/ip3.mySink->pkt_recv;
N++;};
if(ip4.mySink->pkt_recv != 0){cout << "Average packet delay in
IP4: " << 1.0*ip4.mySink->sum_delay/ip4.mySink->pkt_recv <<
endl;
sum += 1.0*ip4.mySink->sum_delay/ip4.mySink->pkt_recv;
N++;};
if(ip5.mySink->pkt_recv != 0){cout << "Average packet delay in
IP5: " << 1.0*ip5.mySink->sum_delay/ip5.mySink->pkt_recv <<
endl;
sum += 1.0*ip5.mySink->sum_delay/ip5.mySink->pkt_recv;
N++;};
if(ip6.mySink->pkt_recv != 0){cout << "Average packet delay in
IP6: " << 1.0*ip6.mySink->sum_delay/ip6.mySink->pkt_recv <<
endl;
sum += 1.0*ip6.mySink->sum_delay/ip6.mySink->pkt_recv;
N++;};
if(ip7.mySink->pkt_recv != 0){cout << "Average packet delay in
IP7: " << 1.0*ip7.mySink->sum_delay/ip7.mySink->pkt_recv <<
endl;
sum += 1.0*ip7.mySink->sum_delay/ip7.mySink->pkt_recv;
N++;};
if(ip8.mySink->pkt_recv != 0){cout << "Average packet delay in
IP8: " << 1.0*ip8.mySink->sum_delay/ip8.mySink->pkt_recv <<
endl;
sum += 1.0*ip8.mySink->sum_delay/ip8.mySink->pkt_recv;
N++;};
if(ip9.mySink->pkt_recv != 0){cout << "Average packet delay in
IP9: " << 1.0*ip9.mySink->sum_delay/ip9.mySink->pkt_recv <<
endl;
sum += 1.0*ip9.mySink->sum_delay/ip9.mySink->pkt_recv;
N++;};
if(ip10.mySink->pkt_recv != 0){cout << "Average packet delay in
IP10: " << 1.0*ip10.mySink->sum_delay/ip10.mySink->pkt_recv <<
endl;
sum += 1.0*ip10.mySink->sum_delay/ip10.mySink->pkt_recv;
N++;};
if(ip11.mySink->pkt_recv != 0){cout << "Average packet delay in
IP11: " << 1.0*ip11.mySink->sum_delay/ip11.mySink->pkt_recv <<
endl;
sum += 1.0*ip11.mySink->sum_delay/ip11.mySink->pkt_recv;
N++;};
if(ip12.mySink->pkt_recv != 0){cout << "Average packet delay in
IP12: " << 1.0*ip12.mySink->sum_delay/ip12.mySink->pkt_recv <<
endl;
sum += 1.0*ip12.mySink->sum_delay/ip12.mySink->pkt_recv;
N++;};
```

```
if(ip13.mySink->pkt_recv != 0){cout << "Average packet delay in
IP13: " << 1.0*ip13.mySink->sum_delay/ip13.mySink->pkt_recv <<
endl;
sum += 1.0*ip13.mySink->sum_delay/ip13.mySink->pkt_recv;
N++;};
if(ip14.mySink->pkt_recv != 0){cout << "Average packet delay in
IP14: " << 1.0*ip14.mySink->sum_delay/ip14.mySink->pkt_recv <<
endl;
sum += 1.0*ip14.mySink->sum_delay/ip14.mySink->pkt_recv;
N++;};
if(ip15.mySink->pkt_recv != 0){cout << "Average packet delay in
IP15: " << 1.0*ip15.mySink->sum_delay/ip15.mySink->pkt_recv <<
endl;
sum += 1.0*ip15.mySink->sum_delay/ip15.mySink->pkt_recv;
N++;};
cout << "# of packets sent: " <<
ip0.mySource->pkt_snt+ip1.mySource->pkt_snt+ip2.mySource->pkt
_snt+ip3.mySource->pkt_snt+ip4.mySource->pkt_snt+ip5.mySource
->pkt_snt+ip6.mySource->pkt_snt+ip7.mySource->pkt_snt+ip8.myS
ource->pkt_snt+ip9.mySource->pkt_snt+ip10.mySource->pkt_snt+ip
11.mySource->pkt_snt+ip12.mySource->pkt_snt+ip13.mySource->p
kt_snt+ip14.mySource->pkt_snt+ip15.mySource->pkt_snt << endl;
cout << "# of packets received: " <<
ip0.mySink->pkt_recv+ip1.mySink->pkt_recv+ip2.mySink->pkt_rec
v+ip3.mySink->pkt_recv+ip4.mySink->pkt_recv+ip5.mySink->pkt_r
ecv+ip6.mySink->pkt_recv+ip7.mySink->pkt_recv+ip8.mySink->pk
t_recv+ip9.mySink->pkt_recv+ip10.mySink->pkt_recv+ip11.mySink
->pkt_recv+ip12.mySink->pkt_recv+ip13.mySink->pkt_recv+ip14.
mySink->pkt_recv+ip15.mySink->pkt_recv << endl;
if(N != 0) cout << "Total average packet delay: " << sum/N << endl;

        sc_close_vcd_trace_file(tf);

        cout << endl << endl <<
"--------------------------------------------------------------------------------"
<< endl;
   cout << "  Press \"Return\" key to end the simulation..." << endl
<< endl;
        getchar();
  return 0;

}
```