COE838: SystemC based NOC Final Report

Charran Thangeswaran

Department of Electrical, Computer, and Biomedical Engineering

Toronto Metropolitan University

Toronto, ON

charran.thangeswaran@torontomu.ca

***Abstract*—** This project involves modeling and simulating an NoC (Network-on-Chip) using SystemC. This NoC will consist of routers (switches) and IPs (hardware modules) for implementation. These are all provided in the form of SystemC code for a basic 1x2 mesh NoC. Specifically, the code for the packet structure, source module, sink module, router module, arbiter module, FIFO buffer module, crossbar switch module, and the main simulation module are all provided. Using the information provided in the project manual and understanding the 1x2 mesh NoC code, a fully-functional 4x4 mesh NoC was developed. This involved modifying various files to ensure proper communication between the source and the sink cores in a 4x4 mesh topology.

I. Introduction

In this project, the NoC is developed using SystemC, while incorporating concepts gained through course material. An NoC is a packet switching communication network between modules in an SoC by using routers. NoCs are invaluable components for modern multi-core systems, offering scalable and efficient communication between IP cores. Like most hardware components, their efficiency is dependent on the software being executed. For this project, SystemC is used for modeling and simulating NoC structures. The initial phase involves analyzing the provided 1x2 mesh NoC, which includes the routers/switches, hardware modules, and interconnects. Understanding the architecture of these components and the packet routing mechanisms lead to being able to expand the design to a 4x4 mesh NoC and test its functionality by generating various types of communication patterns (uniform and neighbouring). By leveraging SystemC's event-driven simulation capabilities, this project aims to provide insights into NoC design trade-offs, including arbitration strategies, buffer sizing, and topology selection.

II. Past Work

The NoC simulation involved using concepts obtained from labs 1, 2a, and 2b in terms of creating and executing the programs via the terminal window.

Specifically, the programs for the 1x2 mesh NoC were developed in SystemC. Using the experience gained from the aforementioned labs, the .cpp and .h files were modified appropriately to collectively form the overall 4x4 mesh NoC.

Furthermore, the Makefile was created by incorporating all the .cpp files as source files (SRCS) to create the corresponding object files (OBJS) as a program (PROGRAM), as was done for labs 1, 2, and 2a. In this case, the .cpp files for the modules involved in the NoC design were used. Ultimately, the Makefile was needed to create the program to produce the .vcd file.

Particularly for labs 1 and 2a, a .vcd file needed to be produced to display the simulation results via GTKWave. This is the most crucial part of the NoC as this is where the simulation results are displayed to ensure the results were produced correctly. To create and close the .vcd file, they must be done on the sc_main.cpp files (in this case main_noc.cpp), by using the SystemC functions, sc_create_vcd_trace_file, sc_close_vcd_trace_file and sc_trace where applicable.

III. Methodology

The development of the NoC involved creating phases based on the objectives listed in the project manual and undergoing multiple tests for each phase.

In the first phase, several tests were conducted to ensure the packets were actually sent to routers. Converting a 1x2 mesh NoC into a 4x4 mesh NoC involved increasing the NoC's size, which also meant increasing the complexity. To achieve this, the main_noc.cpp and arbiter.cpp were primarily modified to update connections to 16 routers and extend the XY routing for a larger mesh respectively.

After that was verified, the next phase involved tracking the time taken for a packet to be sent by the source and received by the sink. Naturally, this involved modifying the source.cpp and sink.cpp files by including a data type in SystemC, sc_time, in each file to retrieve the simulation time for when a packet is sent and when it is received. These times were also displayed on the terminal window, which is confirmed when observing the results displayed on the GTKWave simulation.

Speaking of GTKWave, the final phase was to verify the results were produced accurately on the simulation software. The main_noc.cpp file displays the information on the terminal window regarding what packets were sent and received and the times in which the sending and receiving occurred. But for those to be accurately displayed, the other .cpp files were verified to ensure that the selected source was routing the packets to the selected sink/destination.

As a result of these efforts, the 4x4 mesh NoC simulation was able to run successfully.

IV. Design

First, it was important to understand the packet structure in order to successfully conduct the simulation of the NoC. The source module is responsible for generating the packets, each packet consisting of at least two flits: a header (for routing) and a payload. The header flit contains the source and sink addresses, which are sized based on the number of NoC cores, and influences the size of the FIFO buffer. An imaginary clock bit flips between 0 and 1 to ensure event-driven simulation by distinguishing identical flits. Finally, the tail/header bit determines the end of a packet (where it is set high in the final flit) while payload flits carry data alongside these control bits.

Going over the modules, the first one that I changed was the arbiter module (arbiter.cpp). First thing I did was change the v_req array such that it included 4 bits per element, instead of 3 bits. I also had to change the routing logic by adding more nested conditions to check bits 0, 1, 2 and 3 of v_id instead of just bits 0 and 1, given that the size of mesh NoC is going from the original 1x2 to 4x4.

I also changed the crossbar switch module (crossbar.cpp) by first making sure that all routes were supported and reached. In the original code, some values were missing depending on the corresponding v_cross variable. For example, for v_cross = i0.read(), a case is not implemented for writing o0.

For the source and sink modules (source.cpp and sink.cpp), the only thing I added was the times, t_sent and t_recv, to indicate the times in which the packet was sent by the source and received by the sink. This let me know through the terminal window so I could also easily confirm on the GTKWave software.

Naturally, the NoC Simulator Main module went through the most changes. Given that a 4x4 mesh NoC is being developed from a 1x2 version, the network size needed to change from 2 sources, sinks, and routers each to 16 of each. In order to do this, I changed the signal arrays values of si_source, si_input, and si_sink from 4 to 16. Next, I changed the si_output and all the si_ack_ signals from 16 to 64. With that, I am setting up the NoC to now be a 4x4 mesh topology instead of the original 1x2 mesh topology. Afterwards, I added source and sink IDs scaling from 0-15. Thanks to the comments from the code, I learned that the modules can be connected by hooking up the ports to the signals either by name or through positional notation. In my case, I did it by name and I did it for each source, router, and sink, using the previous 1x2 mesh NoC code for reference. I also modified the text that appears on the terminal window by changing certain cout statements. More importantly, I included user prompts to make a user-inputted source send 10 packets (given the time) to a user-inputted sink/destination. At the end of the simulation, the terminal window displays which packets were sent by the source and which packets were received by the sink, which also includes text to display which source is sending a packet to which sink in detail through the cout statements implemented in the source.cpp and sink.cpp codes. After that is all displayed, the user can invoke the command to generate the .vcd file through GTKWave for simulation to showcase the results.

V. Experimental Results

The results of the NoC simulation are shown with the following images below.

```
cmp18:/home/student1/cthanges/Desktop/COE 838/SystemC_4x4_NoC> noc.x

           SystemC 2.3.0-ASI --- Sep 10 2012 16:44:06
       Copyright (c) 1996-2012 by all Contributors,
       ALL RIGHTS RESERVED

Source ID (0-15): 2
Destination/Sink ID (0-15): 3

--------------------------------------------------------------------------------

     SystemC 4X4 mesh NOC simulator (By: Charran Thangeswaran)
--------------------------------------------------------------------------------
This simulator contains 2 5x5 wormhole routers.
Assume the router has 5 I/O ports with 4 buffers per input port.
and each flit has 21 bits width.
  Press "Enter" or "Return" to begin simulation...

WARNING: Default time step is used for VCD tracing.
Trace Warning:
Traced objects found with name containing [], which may be
interpreted by the waveform viewer in unexpected ways.
So the [] is automatically replaced by ().
```

Figure 1: Start of SystemC 4x4 mesh NoC simulation

```
t: 125 ns  ||  Packet: 1001 is sent by Source: 2 to Destination/Sink: 3

t: 135 ns  ||  Packet:  1001 is received from Source: 2 by Destination/Sink:  3

t: 250 ns  ||  Packet: 1002 is sent by Source: 2 to Destination/Sink: 3

t: 260 ns  ||  Packet:  1002 is received from Source: 2 by Destination/Sink:  3

t: 375 ns  ||  Packet: 1003 is sent by Source: 2 to Destination/Sink: 3

t: 385 ns  ||  Packet:  1003 is received from Source: 2 by Destination/Sink:  3

t: 500 ns  ||  Packet: 1004 is sent by Source: 2 to Destination/Sink: 3

t: 510 ns  ||  Packet:  1004 is received from Source: 2 by Destination/Sink:  3

t: 625 ns  ||  Packet: 1005 is sent by Source: 2 to Destination/Sink: 3

t: 635 ns  ||  Packet:  1005 is received from Source: 2 by Destination/Sink:  3

t: 750 ns  ||  Packet: 1006 is sent by Source: 2 to Destination/Sink: 3

t: 760 ns  ||  Packet:  1006 is received from Source: 2 by Destination/Sink:  3

t: 875 ns  ||  Packet: 1007 is sent by Source: 2 to Destination/Sink: 3

t: 885 ns  ||  Packet:  1007 is received from Source: 2 by Destination/Sink:  3

t: 1 us  ||  Packet: 1008 is sent by Source: 2 to Destination/Sink: 3

t: 1010 ns  ||  Packet:  1008 is received from Source: 2 by Destination/Sink:  3

t: 1125 ns  ||  Packet: 1009 is sent by Source: 2 to Destination/Sink: 3

t: 1135 ns  ||  Packet:  1009 is received from Source: 2 by Destination/Sink:  3

t: 1250 ns  ||  Packet: 1010 is sent by Source: 2 to Destination/Sink: 3

t: 1260 ns  ||  Packet:  1010 is received from Source: 2 by Destination/Sink:  3


--------------------------------------------------------------------------------
End of switch operation...
Packets sent: 10
Packets received: 10
```

Figure 2: 10 Packets transmitted between Source 2 and Sink 3

In this case, I set the source ID to 2 and destination/sink ID to 3. This means that source #2 is going to distribute 10 packets to sink #3 via the routers.

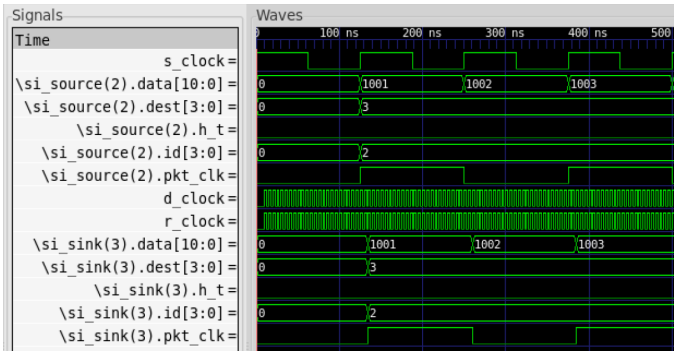

Figure 3: GTKWave Simulation Results

As shown in Figure 3, the packets being transmitted and times they are sent and received at by the source and sink respectively match perfectly with the results shown in Figure 2. I lined up the signals based on the logic of the NoC. First, I started with the source-related signals. Starting off with the source clock (s_clock) and following that with all the signals for source 2 (si_source(2)). After every 125 ns, a packet is sent from the source to the destination (i.e. sink 3). This shows that the sending of the packets are triggered on the rising edge of the source clock, which is as intended. The signals for the destination and id (si_source(2).dest and si_source(2).id respectively) are displayed for the entire duration of the 10 packets being sent (approximately 1372.5 ns) to further prove that source 2 is sending 10 packets to sink 3. After those signals, I added the destination/sink and router clocks (d_clock and r_clock), which operate exactly the same, given that the routers are being used to transmit the packets from the source to the destination/sink, so naturally the sink clock will operate exactly as the router clock. Following those clock signals, I added the sink 3 (si_sink(3)) signals to show the 10 packets being received, along with the signals for the destination and id (si_sink(3).dest and si_sink(3).id respectively) are displayed for the entire duration of the 10 packets being sent. The simulation in Figure 3 also proves that the time in which a packet is sent by the source, the sink receives it 10 ns later; exactly as listed in the terminal window in Figure 2.

VI. Conclusion

By using SystemC and knowledge gained through course material, the simulation of the 4x4 mesh NoC was successful. This NoC simulator models a 4x4 mesh network where 16 source nodes generate packets that travel through interconnected routers to reach 16 destination/sink nodes. Each router contains input buffers (FIFO buffer module) to store any incoming packets, an arbiter (arbiter module) that implements the XY routing to determine the output port, and a crossbar (crossbar switch module) that forwards the packets accordingly. Sources (source module) generate the packets with incrementing data and specified destinations, routers (router module) track these packet flows while managing contention and wormhole routing (where packets are split into flits), and sinks (sink module) receive and acknowledge the packets. This allows the NoC simulator main module to coordinate the entire NoC system, initializing components, connecting signals, managing IDs, and collecting the data/results produced, while different clocks simulate asynchronous operation between the sources, routers, and sinks. Although I was not able to convert the 4x4 mesh NoC into a torus topology, I still learned a lot about how NoCs work, which I am confident will help me in my future endeavours working with embedded systems and SoCs.

VII. References

Introduction to SystemC. COE838: Systems-on-Chip Design. (n.d.). https://www.ecb.torontomu.ca/%7Ecourses/coe838/labs/lab1.pdf

SystemC based NoC (Network-on-Chip) Modeling Course Project. COE838/EE8221: Systems-on-Chip Design. (n.d.). https://www.ecb.torontomu.ca/~courses/coe838/labs/NoC-Project.pdf

VIII. Appendix

The appendix consists only of the .cpp codes changed as mentioned in the Design section of the report. All other files were unchanged and thus not included.

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<4> 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;
	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;


	// 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();
		if (!req0.read()[4]) //if FIFO buffer is not empty

		{
			//if(!v_connected_input[0]) // if input is not connected i.e. it is header
			if(v_id[1] < req0.read()[1]) v_req[0]=3; // go to east
			else {
				if(v_id[1] > req0.read()[1])v_req[0]=5; //go to west
				else{
					if(v_id[3] < req0.read()[3])v_req[0]=4; // go to south
					else{
						if(v_id[3] > req0.read()[3])v_req[0]=2; //go to north
						else{
	if(v_id[2] < req0.read()[2])v_req[0]=4; // go to south
	else{
		if(v_id[2] > req0.read()[2])v_req[0]=2; //go to north
		else{
			if(v_id[0] < req0.read()[0]) v_req[0]=3;// go to east
			else{
				if(v_id[0] > req0.read()[0])v_req[0]=5; //go to west
				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 // isnt this always 0 if its been intialized as 0.
			{
				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]) // if input is not connected i.e. it is header
			if(v_id[1] < req1.read()[1]) v_req[1]=3; // go to east
			else {
				if(v_id[1] > req1.read()[1])v_req[1]=5; //go to west
				else {
					if(v_id[3] < req1.read()[3])v_req[1]=4; // go to south
					else {
						if(v_id[3] > req1.read()[3])v_req[1]=2; //go to north
						else{
	if(v_id[2] < req1.read()[2])v_req[1]=4; // go to south
	else{
		if(v_id[2] > req1.read()[2])v_req[1]=2; //go to north
		else{
			if(v_id[0] < req1.read()[0]) v_req[1]=3; // go to east
			else{
				if(v_id[0] > req1.read()[0])v_req[1]=5; //go to west
				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]) // if input is not connected i.e. it is header
                                if(v_id[1] < req2.read()[1]) v_req[2]=3; // go to east
                                else {
                                        if(v_id[1] > req2.read()[1])v_req[2]=5; //go to west
                                        else {
                                                if(v_id[3] < req2.read()[3])v_req[2]=4; // go to south
                                                else {
                                                        if(v_id[3] > req2.read()[3])v_req[2]=2; //go to north
                                                        else{
        if(v_id[2] < req2.read()[2])v_req[2]=4; // go to south
        else{
                if(v_id[2] > req2.read()[2])v_req[2]=2; //go to north
                else{
                        if(v_id[0] < req2.read()[0]) v_req[2]=3; // go to east
                        else{
                                if(v_id[1] > req2.read()[1])v_req[2]=5; //go to west
                                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]) // if input is not connected i.e. it is header
                        if(v_id[1] < req3.read()[1]) v_req[3]=3; // go to east
                        else {
                                if(v_id[1] > req3.read()[1])v_req[3]=5; //go to west
                                else {
                                        if(v_id[3] < req3.read()[3])v_req[3]=4; // go to south
                                        else {
                                                if(v_id[3] > req3.read()[3])v_req[3]=2; //go to north
                                                else{
        if(v_id[2] < req3.read()[2])v_req[3]=4; // go to south
        else{
                if(v_id[2] > req3.read()[2])v_req[3]=2; //go to north
                else{
                        if(v_id[0] < req3.read()[0]) v_req[3]=3; // go to east
                        else{
                                if(v_id[0] > req3.read()[0])v_req[3]=5; //go to west
                                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]) // if input is not
connected i.e. it is header
                                if(v_id[1] <  req4.read()[1])  v_req[4]=3;  //
go to east
                                else {
                                        if(v_id[1]                          >
req4.read()[1])v_req[4]=5; //go to west
                                        else {
                                                if(v_id[3]              <
req4.read()[3])v_req[4]=4; // go to south
                                                else {
                                                        if(v_id[3]
> req4.read()[3])v_req[4]=2; //go to north
                                                        else{

        if(v_id[2] < req4.read()[2])v_req[4]=4; // go to south

        else{

                if(v_id[2] > req4.read()[2])v_req[4]=2; //go to north

                else{

                        if(v_id[0] <  req4.read()[0])  v_req[4]=3;  //
go to east
                                        else{

                                                if(v_id[0]                          >
req4.read()[0])v_req[4]=5; //go to west

                                                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);
        }
    }
```

crossbar.cpp

```
// crossbar.cpp
#include "packet.h"
#include "crossbar.h"

void crossbar :: func()
        {
        packet v_cross0;
        packet v_cross1;
        packet v_cross2;
        packet v_cross3;
        packet v_cross4;
        sc_uint<15>  v_config;

        // functionality
```

```
	while( true )
	{
		wait();
		v_config = config.read();
		if (i0.event())
		{
			v_cross0 = i0.read();
			switch (v_config(2,0)) {
				case 1:	o0.write(v_cross0);
break;
				case	2:	o1.write(v_cross0);
break;
				case	3:	o2.write(v_cross0);
break;
				case	4:	o3.write(v_cross0);
break;
				case 5:	o4.write(v_cross0);
break;
				default:	cout	<<
"---------------------------------wrong destination   " <<endl ;break ;
				}
		}
		if (i1.event())
		{
			v_cross1 = i1.read();
			switch (v_config(5,3)) {
				case 1:	o0.write(v_cross1);
break;
				case 2:	o1.write(v_cross1);
break;
				case	3:	o2.write(v_cross1);
break;
				case	4:	o3.write(v_cross1);
break;
				case	5:	o4.write(v_cross1);
break;
				default:	cout	<<
"-----------------------------------wrong destination   " <<endl;  break ;
				}
		}
		if (i2.event())
		{
			v_cross2 = i2.read();
			switch (v_config(8,6)) {
				case 1:	o0.write(v_cross2);
break;
				case	2:	o1.write(v_cross2);
break;
				case	3:	o2.write(v_cross2);
break;
				case	4:	o3.write(v_cross2);
break;
				case	5:	o4.write(v_cross2);
break;
				default:	cout	<<
"---------------------------------wrong destination   " <<endl; break ;
				}
		}
		if (i3.event())
		{
			v_cross3 = i3.read();
			switch (v_config(11,9)) {
				case 1:	o0.write(v_cross3);
break;
				case	2:	o1.write(v_cross3);
break;
				case	3:	o2.write(v_cross3);
break;
				case	4:	o3.write(v_cross3);
break;
				case	5:	o4.write(v_cross3);
break;
				default:	cout	<<
"---------------------------------wrong destination   " <<endl; break ;
				}
		}
		if (i4.event())
		{
			v_cross4 = i4.read();
			switch (v_config(14,12)) {
				case 1:	o0.write(v_cross4);
break;
				case	2:	o1.write(v_cross4);
break;
				case	3:	o2.write(v_cross4);
break;
				case	4:	o3.write(v_cross4);
break;
				case	5:	o4.write(v_cross4);
break;
				default:	cout	<<
"-------------------------------------wrong destination" <<endl ;break ;
				}
		}
	}
}
```

source.cpp

```
// source.cpp
#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

	sc_time t_sent;

	while(true)
	{
		wait();
		if(!ach_in.read())
		{
			if(ch_k.read() == source_id.read())
			{
				v_packet_out.data	=
v_packet_out.data + 1  ; // made a desired data
				v_packet_out.id	=
source_id.read();
				v_packet_out.dest=  d_est.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++;

if((pkt_snt%5)==0)v_packet_out.h_t=true; // make tail flit (the packet size is
5)
				packet_out.write(v_packet_out);
				t_sent = sc_time_stamp();
				cout	<<	"\nt:	"	<<
sc_time_stamp() << "  ||  Packet: " << v_packet_out.data << " is sent by
Source:  "  <<  source_id.read()	<<  "  to  Destination/Sink:  "<<
v_packet_out.dest <<endl;
   exclode:;
			}
		}
	}
}
```

sink.cpp

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

        packet v_packet;

        sc_time t_recv;

        if ( sclk.event() ) ack_out.write(false);
        if (packet_in.event() ) {

                pkt_recv++ ;
                ack_out.write(true);
                v_packet= packet_in.read();
                t_recv = sc_time_stamp();
                cout << "\nt: " << sc_time_stamp() << "  ||  Packet: "
<< (int)v_packet.data<< " is received from Source: " << (int)v_packet.id << "
by Destination/Sink:  "  <<  (int)sink_id.read()  <<  endl  <<
"_____
_____";

  }
}
```

main_noc.cpp

```
// main.cpp
#include "systemc.h"
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include "packet.h"
#include "source.h"
#include "sink.h"
#include "router.h"


int sc_main(int argc, char *argv[])
{
        // Define signals for interfacing modules
        sc_signal<packet> si_source[16];
        sc_signal<packet> si_input[16];
        sc_signal<packet> si_zero[64];
        sc_signal<packet> si_sink[16];
        sc_signal<packet> si_output[64];

        // Define acknowledge signals for handshake protocol between
modules
        sc_signal<bool>  si_ack_src[64],si_ack_ou[64];
        sc_signal<bool>  si_ack_sink[16],si_ack_in[64];
        sc_signal<bool>  si_ack_zero[64];


        sc_signal<sc_uint<4>                                          >
siid0,siid1,siid2,siid3,siid4,siid5,siid6,siid7,siid8,siid9,siid10,siid11,siid12,sii
d13,siid14,siid15;
        sc_signal<sc_uint<4>  >  scid0,scid1,  scid2,  scid3,scid4,scid5,
scid6, scid7,scid8,scid9, scid10, scid11,scid12,scid13, scid14, scid15;
        sc_signal<sc_uint<4> > id0,id1, id2, id3, id4,id5, id6, id7,id8,id9,
id10, id11,id12,id13, id14, id15;
        sc_signal<int> scinput;
        sc_signal<sc_uint<4> > check;
        sc_signal <packet> sioutput[16];
        int i,j;
        sc_clock s_clock("S_CLOCK", 125, SC_NS, 0.5, 0.0, SC_NS); //
source clock
        //sc_clock s_clock("S_CLOCK", 5, SC_NS, 0.5, 10.0, SC_NS); //
source clk = router clk (Interim Report)
        sc_clock  r_clock("R_CLOCK",  5,  SC_NS,  0.5,  10.0,  SC_NS);
        // router clock
        sc_clock  d_clock("D_CLOCK",  5,  SC_NS,  0.5,  10.0,  SC_NS);
        // destination clock

        // Module instiatiations follow
        // Note that modules can be connected by hooking up ports
        // to signals by name or by using a positional notation


        source source0("source0");
        source0(si_source[0],       scid0,       si_ack_src[0],       s_clock,
scinput,check);


   source source1("source1");
        source1(si_source[1],       scid1,       si_ack_src[1],       s_clock,
scinput,check);

        source source2("source2");
        source2(si_source[2],       scid2,       si_ack_src[2],       s_clock,
scinput,check);


        source source3("source3");
        source3(si_source[3],       scid3,       si_ack_src[3],       s_clock,
scinput,check);

        source source4("source4");
        source4(si_source[4],       scid4,       si_ack_src[4],       s_clock,
scinput,check);


   source source5("source5");
        source5(si_source[5],       scid5,       si_ack_src[5],       s_clock,
scinput,check);

        source source6("source6");
        source6(si_source[6],       scid6,       si_ack_src[6],       s_clock,
scinput,check);


        source source7("source7");
        source7(si_source[7],       scid7,       si_ack_src[7],       s_clock,
scinput,check);

        source source8("source8");
        source8(si_source[8],       scid8,       si_ack_src[8],       s_clock,
scinput,check);


   source source9("source9");
        source9(si_source[9],       scid9,       si_ack_src[9],       s_clock,
scinput,check);

        source source10("source10");
        source10(si_source[10],      scid10,      si_ack_src[10],      s_clock,
scinput,check);


        source source11("source11");
        source11(si_source[11],      scid11,      si_ack_src[11],      s_clock,
scinput,check);

        source source12("source12");
        source12(si_source[12],      scid12,      si_ack_src[12],      s_clock,
scinput,check);


   source source13("source13");
        source13(si_source[13],      scid13,      si_ack_src[13],      s_clock,
scinput,check);

        source source14("source14");
```

```
        source14(si_source[14],  scid14,  si_ack_src[14],  s_clock,
scinput,check);

        source source15("source15");
        source15(si_source[15],  scid15,  si_ack_src[15],  s_clock,
scinput,check);

        router router0("router0");
        // hooking up signals to ports by name
        router0.in0(si_source[0]);
        router0.in1(si_output[2]);
        router0.in2(si_output[12]);
        router0.in3(si_zero[1]);
        router0.in4(si_zero[2]);

        router0.router_id(id0);
        //router0.router_id(0);

        router0.out0(si_sink[0]);
        router0.out2(si_output[0]);
        router0.out3(si_output[1]);
        router0.out1(si_zero[3]);
        router0.out4(si_zero[4]);

        router0.inack0(si_ack_sink[0]);
        router0.inack1(si_ack_in[2]);
        router0.inack2(si_ack_in[12]);
        router0.inack3(si_ack_zero[1]);
        router0.inack4(si_ack_zero[2]);

        router0.outack0(si_ack_src[0]);
        router0.outack2(si_ack_in[0]);
        router0.outack3(si_ack_in[1]);
        router0.outack1(si_ack_zero[3]);
        router0.outack4(si_ack_zero[4]);

        router0.rclk(r_clock);

        router router1("router1");
        // hooking up signals to ports by name
        router1.in0(si_source[1]);
        router1.in1(si_output[0]);
        router1.in2(si_output[7]);
        router1.in3(si_output[16]);
        router1.in4(si_zero[5]);


        router1.router_id(id1);
        //router1.router_id(1);

        router1.out0(si_sink[1]);
        router1.out4(si_output[2]);
        router1.out3(si_output[3]);
        router1.out2(si_output[4]);
        router1.out1(si_zero[6]);

        router1.inack0(si_ack_sink[1]);
        router1.inack1(si_ack_in[0]);
        router1.inack2(si_ack_in[7]);
        router1.inack3(si_ack_in[16]);
        router1.inack4(si_ack_zero[5]);

        router1.outack0(si_ack_src[1]);
        router1.outack4(si_ack_in[2]);
        router1.outack3(si_ack_in[3]);
        router1.outack2(si_ack_in[4]);
        router1.outack1(si_ack_zero[6]);

        router1.rclk(r_clock);
        //need 64 code statement
        router router2("router2");
        // hooking up signals to ports by name
        router2.in0(si_source[2]);
        router2.in1(si_output[4]);
        router2.in2(si_output[9]);
        router2.in3(si_output[17]);
        router2.in4(si_zero[7]);


        router2.router_id(id2);
        //router2.router_id(2);

        router2.out0(si_sink[2]);
        router2.out1(si_zero[19]);
        router2.out2(si_output[5]);
        router2.out3(si_output[6]);
        router2.out4(si_output[7]);


        router2.inack0(si_ack_sink[2]);
        router2.inack1(si_ack_in[4]);
        router2.inack2(si_ack_in[9]);
        router2.inack3(si_ack_in[17]);
        router2.inack4(si_ack_zero[7]);


        router2.outack0(si_ack_src[2]);
        router2.outack1(si_ack_zero[19]);
        router2.outack2(si_ack_in[5]);
        router2.outack3(si_ack_in[6]);
        router2.outack4(si_ack_in[7]);


        router2.rclk(r_clock);

        router router3("router3");
        // hooking up signals to ports by name
        router3.in0(si_source[3]);
        router3.in1(si_output[5]);
        router3.in2(si_output[21]);
        router3.in3(si_zero[8]);
        router3.in4(si_zero[9]);

        router3.router_id(id3);
        //router3.router_id(3);

        router3.out0(si_sink[3]);
        router3.out3(si_output[8]);
        router3.out4(si_output[9]);
        router3.out2(si_zero[20]);
        router3.out1(si_zero[21]);


        router3.inack0(si_ack_sink[3]);
        router3.inack1(si_ack_in[5]);
        router3.inack2(si_ack_in[21]);
        router3.inack3(si_ack_zero[8]);
        router3.inack4(si_ack_zero[9]);


        router3.outack0(si_ack_src[3]);
        router3.outack3(si_ack_in[8]);
        router3.outack4(si_ack_in[9]);
        router3.outack2(si_ack_zero[20]);
        router3.outack1(si_ack_zero[21]);

        router3.rclk(r_clock);

        router router4("router4");
        // hooking up signals to ports by name
        router4.in0(si_source[4]);
```

```
router4.in1(si_output[1]);
router4.in2(si_output[13]);
router4.in3(si_output[26]);
router4.in4(si_zero[10]);

router4.router_id(id4);
//router0.router_id(0);

router4.out0(si_sink[4]);
router4.out2(si_output[10]);
router4.out3(si_output[11]);
router4.out1(si_output[12]);
router4.out4(si_zero[22]);

router4.inack0(si_ack_sink[4]);
router4.inack1(si_ack_in[1]);
router4.inack2(si_ack_in[13]);
router4.inack3(si_ack_in[26]);
router4.inack4(si_ack_zero[10]);

router4.outack0(si_ack_src[4]);
router4.outack2(si_ack_in[10]);
router4.outack3(si_ack_in[11]);
router4.outack1(si_ack_in[12]);
router4.outack4(si_ack_zero[22]);

router4.rclk(r_clock);

router router5("router5");
// hooking up signals to ports by name
router5.in0(si_source[5]);
router5.in1(si_output[3]);
router5.in2(si_output[10]);
router5.in3(si_output[20]);
router5.in4(si_output[30]);


router5.router_id(id5);
//router1.router_id(1);

router5.out0(si_sink[5]);
router5.out4(si_output[13]);
router5.out3(si_output[14]);
router5.out2(si_output[15]);
router5.out1(si_output[16]);

router5.inack0(si_ack_sink[5]);
router5.inack1(si_ack_in[3]);
router5.inack2(si_ack_in[10]);
router5.inack3(si_ack_in[20]);
router5.inack4(si_ack_in[30]);

router5.outack0(si_ack_src[5]);
router5.outack4(si_ack_in[13]);
router5.outack3(si_ack_in[14]);
router5.outack2(si_ack_in[15]);
router5.outack1(si_ack_in[16]);

router5.rclk(r_clock);
//need 64 code statement

router router6("router6");
// hooking up signals to ports by name
router6.in0(si_source[6]);
router6.in1(si_output[6]);
router6.in2(si_output[15]);
router6.in3(si_output[22]);
router6.in4(si_output[31]);


router6.router_id(id6);
//router2.router_id(2);

router6.out0(si_sink[6]);
router6.out1(si_output[17]);
router6.out2(si_output[18]);
router6.out3(si_output[19]);
router6.out4(si_output[20]);


router6.inack0(si_ack_sink[6]);
router6.inack1(si_ack_in[6]);
router6.inack2(si_ack_in[15]);
router6.inack3(si_ack_in[22]);
router6.inack4(si_ack_in[31]);


router6.outack0(si_ack_src[6]);
router6.outack1(si_ack_in[17]);
router6.outack2(si_ack_in[18]);
router6.outack3(si_ack_in[19]);
router6.outack4(si_ack_in[20]);


router6.rclk(r_clock);

router router7("router7");
// hooking up signals to ports by name
router7.in0(si_source[7]);
router7.in1(si_output[8]);
router7.in2(si_output[18]);
router7.in3(si_output[35]);
router7.in4(si_zero[11]);

router7.router_id(id7);
//router3.router_id(3);

router7.out0(si_sink[7]);
router7.out1(si_output[21]);
router7.out4(si_output[22]);
router7.out3(si_output[23]);
router7.out2(si_zero[23]);


router7.inack0(si_ack_sink[7]);
router7.inack1(si_ack_in[8]);
router7.inack2(si_ack_in[18]);
router7.inack3(si_ack_in[35]);
router7.inack4(si_ack_zero[11]);


router7.outack0(si_ack_src[7]);
router7.outack1(si_ack_in[21]);
router7.outack4(si_ack_in[22]);
router7.outack3(si_ack_in[23]);
router7.outack2(si_ack_zero[23]);

router7.rclk(r_clock);

//----------------------------------------------------------------------

router router8("router8");
// hooking up signals to ports by name
router8.in0(si_source[8]);
router8.in1(si_output[11]);
router8.in2(si_output[27]);
router8.in3(si_output[38]);
router8.in4(si_zero[12]);

router8.router_id(id8);
//router0.router_id(0);

router8.out0(si_sink[8]);
router8.out2(si_output[24]);
```

```
router8.out3(si_output[25]);
router8.out1(si_output[26]);
router8.out4(si_zero[24]);

router8.inack0(si_ack_sink[8]);
router8.inack1(si_ack_in[11]);
router8.inack2(si_ack_in[27]);
router8.inack3(si_ack_in[38]);
router8.inack4(si_ack_zero[12]);

router8.outack0(si_ack_src[8]);
router8.outack2(si_ack_in[24]);
router8.outack3(si_ack_in[25]);
router8.outack1(si_ack_in[26]);
router8.outack4(si_ack_zero[24]);

router8.rclk(r_clock);

router router9("router9");
// hooking up signals to ports by name
router9.in0(si_source[9]);
router9.in1(si_output[14]);
router9.in2(si_output[24]);
router9.in3(si_output[34]);
router9.in4(si_output[41]);


router9.router_id(id9);
//router1.router_id(1);

router9.out0(si_sink[9]);
router9.out4(si_output[27]);
router9.out3(si_output[28]);
router9.out2(si_output[29]);
router9.out1(si_output[30]);

router9.inack0(si_ack_sink[9]);
router9.inack1(si_ack_in[14]);
router9.inack2(si_ack_in[24]);
router9.inack3(si_ack_in[34]);
router9.inack4(si_ack_in[41]);

router9.outack0(si_ack_src[9]);
router9.outack4(si_ack_in[27]);
router9.outack3(si_ack_in[28]);
router9.outack2(si_ack_in[29]);
router9.outack1(si_ack_in[30]);

router9.rclk(r_clock);
//need 64 code statement

router router10("router10");
// hooking up signals to ports by name
router10.in0(si_source[10]);
router10.in1(si_output[19]);
router10.in2(si_output[29]);
router10.in3(si_output[36]);
router10.in4(si_output[43]);


router10.router_id(id10);
//router2.router_id(2);

router10.out0(si_sink[10]);
router10.out1(si_output[31]);
router10.out2(si_output[32]);
router10.out3(si_output[33]);
router10.out4(si_output[34]);


router10.inack0(si_ack_sink[10]);
router10.inack1(si_ack_in[19]);
router10.inack2(si_ack_in[29]);
router10.inack3(si_ack_in[36]);
router10.inack4(si_ack_in[43]);


router10.outack0(si_ack_src[10]);
router10.outack1(si_ack_in[31]);
router10.outack2(si_ack_in[32]);
router10.outack3(si_ack_in[33]);
router10.outack4(si_ack_in[34]);


router10.rclk(r_clock);

router router11("router11");
// hooking up signals to ports by name
router11.in0(si_source[11]);
router11.in1(si_output[23]);
router11.in2(si_output[32]);
router11.in3(si_output[46]);
router11.in4(si_zero[13]);

router11.router_id(id11);
//router3.router_id(3);

router11.out0(si_sink[11]);
router11.out1(si_output[35]);
router11.out4(si_output[36]);
router11.out3(si_output[37]);
router11.out2(si_zero[25]);


router11.inack0(si_ack_sink[11]);
router11.inack1(si_ack_in[23]);
router11.inack2(si_ack_in[32]);
router11.inack3(si_ack_in[46]);
router11.inack4(si_ack_zero[13]);


router11.outack0(si_ack_src[11]);
router11.outack1(si_ack_in[35]);
router11.outack4(si_ack_in[36]);
router11.outack3(si_ack_in[37]);
router11.outack2(si_ack_zero[25]);

router11.rclk(r_clock);

//-------------------------------------------------

router router12("router12");
// hooking up signals to ports by name
router12.in0(si_source[12]);
router12.in1(si_output[25]);
router12.in2(si_output[40]);
router12.in3(si_zero[14]);
router12.in4(si_zero[15]);

router12.router_id(id12);
//router0.router_id(0);

router12.out0(si_sink[12]);
router12.out1(si_output[38]);
router12.out2(si_output[39]);
router12.out3(si_zero[26]);
router12.out4(si_zero[27]);

router12.inack0(si_ack_sink[12]);
router12.inack1(si_ack_in[25]);
router12.inack2(si_ack_in[40]);
router12.inack3(si_ack_zero[14]);
router12.inack4(si_ack_zero[15]);
```

```
router12.outack0(si_ack_src[12]);
router12.outack1(si_ack_in[38]);
router12.outack2(si_ack_in[39]);
router12.outack3(si_ack_zero[26]);
router12.outack4(si_ack_zero[27]);

router12.rclk(r_clock);

router router13("router13");
// hooking up signals to ports by name
router13.in0(si_source[13]);
router13.in1(si_output[28]);
router13.in2(si_output[39]);
router13.in3(si_output[45]);
router13.in4(si_zero[16]);


router13.router_id(id13);
//router1.router_id(1);

router13.out0(si_sink[13]);
router13.out4(si_output[40]);
router13.out1(si_output[41]);
router13.out2(si_output[42]);
router13.out3(si_zero[28]);

router13.inack0(si_ack_sink[13]);
router13.inack1(si_ack_in[28]);
router13.inack2(si_ack_in[39]);
router13.inack3(si_ack_in[45]);
router13.inack4(si_ack_zero[16]);

router13.outack0(si_ack_src[13]);
router13.outack4(si_ack_in[40]);
router13.outack1(si_ack_in[41]);
router13.outack2(si_ack_in[42]);
router13.outack3(si_ack_zero[28]);

router13.rclk(r_clock);
//need 64 code statement

router router14("router14");
// hooking up signals to ports by name
router14.in0(si_source[14]);
router14.in1(si_output[33]);
router14.in2(si_output[42]);
router14.in3(si_output[47]);
router14.in4(si_zero[17]);


router14.router_id(id14);
//router2.router_id(2);

router14.out0(si_sink[14]);
router14.out1(si_output[43]);
router14.out2(si_output[44]);
router14.out4(si_output[45]);
router14.out3(si_zero[29]);


router14.inack0(si_ack_sink[14]);
router14.inack1(si_ack_in[33]);
router14.inack2(si_ack_in[42]);
router14.inack3(si_ack_in[47]);
router14.inack4(si_ack_zero[17]);


router14.outack0(si_ack_src[14]);
router14.outack1(si_ack_in[43]);
router14.outack2(si_ack_in[44]);
router14.outack4(si_ack_in[45]);
router14.outack3(si_ack_zero[29]);

router14.rclk(r_clock);

router router15("router15");
// hooking up signals to ports by name
router15.in0(si_source[15]);
router15.in1(si_output[37]);
router15.in2(si_output[44]);
router15.in3(si_zero[18]);
router15.in4(si_zero[19]);

router15.router_id(id15);
//router3.router_id(3);

router15.out0(si_sink[15]);
router15.out1(si_output[46]);
router15.out4(si_output[47]);
router15.out2(si_zero[30]);
router15.out3(si_zero[31]);


router15.inack0(si_ack_sink[15]);
router15.inack1(si_ack_in[37]);
router15.inack2(si_ack_in[44]);
router15.inack3(si_ack_zero[18]);
router15.inack4(si_ack_zero[19]);


router15.outack0(si_ack_src[15]);
router15.outack1(si_ack_in[46]);
router15.outack4(si_ack_in[47]);
router15.outack2(si_ack_zero[30]);
router15.outack3(si_ack_zero[31]);

router15.rclk(r_clock);



sink sink0("sink0");
sink0(si_sink[0], si_ack_sink[0], siid0, d_clock, sioutput[0]);


sink sink1("sink1");
sink1(si_sink[1], si_ack_sink[1], siid1, d_clock, sioutput[1]);


sink sink2("sink2");
sink2(si_sink[2], si_ack_sink[2], siid2, d_clock, sioutput[2]);


sink sink3("sink3");
sink3(si_sink[3], si_ack_sink[3], siid3, d_clock, sioutput[3]);


sink sink4("sink4");
sink4(si_sink[4], si_ack_sink[4], siid4, d_clock, sioutput[4]);


sink sink5("sink5");
sink5(si_sink[5], si_ack_sink[5], siid5, d_clock, sioutput[5]);


sink sink6("sink6");
sink6(si_sink[6], si_ack_sink[6], siid6, d_clock, sioutput[6]);


sink sink7("sink7");
sink7(si_sink[7], si_ack_sink[7], siid7, d_clock, sioutput[7]);

sink sink8("sink8");
sink8(si_sink[8], si_ack_sink[8], siid8, d_clock, sioutput[8]);
```

```
	sink sink9("sink9");
	sink9(si_sink[9], si_ack_sink[9], siid9, d_clock, sioutput[9]);


	sink sink10("sink10");
	sink10(si_sink[10],	si_ack_sink[10],	siid10,	d_clock,
sioutput[10]);


	sink sink11("sink11");
	sink11(si_sink[11], si_ack_sink[11], siid11, d_clock, sioutput[11]);

	sink sink12("sink12");
	sink12(si_sink[12],	si_ack_sink[12],	siid12,	d_clock,
sioutput[12]);


	sink sink13("sink13");
	sink13(si_sink[13],	si_ack_sink[13],	siid13,	d_clock,
sioutput[13]);


	sink sink14("sink14");
	sink14(si_sink[14],	si_ack_sink[14],	siid14,	d_clock,
sioutput[14]);


	sink sink15("sink15");
	sink15(si_sink[15],	si_ack_sink[15],	siid15,	d_clock,
sioutput[15]);



//sc_start(0, SC_NS);
  // tracing:
	// trace file creation
	sc_trace_file *tf = sc_create_vcd_trace_file("graph");
	// External Signals
	sc_trace(tf, s_clock, "s_clock");
	sc_trace(tf, d_clock, "d_clock");
	sc_trace(tf, r_clock, "r_clock");
	sc_trace(tf, si_source[0], "si_source[0]");
	sc_trace(tf, si_source[1], "si_source[1]");
	sc_trace(tf, si_source[2], "si_source[2]");
	sc_trace(tf, si_source[3], "si_source[3]");
	sc_trace(tf, si_source[4], "si_source[4]");
	sc_trace(tf, si_source[5], "si_source[5]");
	sc_trace(tf, si_source[6], "si_source[6]");
	sc_trace(tf, si_source[7], "si_source[7]");
	sc_trace(tf, si_source[8], "si_source[8]");
	sc_trace(tf, si_source[9], "si_source[9]");
	sc_trace(tf, si_source[10], "si_source[10]");
	sc_trace(tf, si_source[11], "si_source[11]");
	sc_trace(tf, si_source[12], "si_source[12]");
	sc_trace(tf, si_source[13], "si_source[13]");
	sc_trace(tf, si_source[14], "si_source[14]");
	sc_trace(tf, si_source[15], "si_source[15]");
	sc_trace(tf, si_sink[0], "si_sink[0]");
	sc_trace(tf, si_sink[1], "si_sink[1]");
	sc_trace(tf, si_sink[2], "si_sink[2]");
	sc_trace(tf, si_sink[3], "si_sink[3]");
	sc_trace(tf, si_sink[4], "si_sink[4]");
	sc_trace(tf, si_sink[5], "si_sink[5]");
	sc_trace(tf, si_sink[6], "si_sink[6]");
	sc_trace(tf, si_sink[7], "si_sink[7]");
	sc_trace(tf, si_sink[8], "si_sink[8]");
	sc_trace(tf, si_sink[9], "si_sink[9]");
	sc_trace(tf, si_sink[10], "si_sink[10]");
	sc_trace(tf, si_sink[11], "si_sink[11]");
	sc_trace(tf, si_sink[12], "si_sink[12]");
	sc_trace(tf, si_sink[13], "si_sink[13]");
	sc_trace(tf, si_sink[14], "si_sink[14]");
	sc_trace(tf, si_sink[15], "si_sink[15]");

	cout<< "Source ID (0-15): ";
	cin >> i;
	cout<< "Destination/Sink ID (0-15): ";
	cin >> j;
	scinput.write(j);

	id0.write(0);
	id1.write(1);
	id2.write(2);
	id3.write(3);
	id4.write(4);
	id5.write(5);
	id6.write(6);
	id7.write(7);
	id8.write(8);
	id9.write(9);
	id10.write(10);
	id11.write(11);
	id12.write(12);
	id13.write(13);
	id14.write(14);
	id15.write(15);

	check.write(i);

	scid0.write(0);
	scid1.write(1);
	scid2.write(2);
	scid3.write(3);
	scid4.write(4);
	scid5.write(5);
	scid6.write(6);
	scid7.write(7);
	scid8.write(8);
	scid9.write(9);
	scid10.write(10);
	scid11.write(11);
	scid12.write(12);
	scid13.write(13);
	scid14.write(14);
	scid15.write(15);

	siid0.write(0);
	siid1.write(1);
	siid2.write(2);
	siid3.write(3);
	siid4.write(4);
	siid5.write(5);
	siid6.write(6);
	siid7.write(7);
	siid8.write(8);
	siid9.write(9);
	siid10.write(10);
	siid11.write(11);
	siid12.write(12);
	siid13.write(13);
	siid14.write(14);
	siid15.write(15);

	cout << endl;
	cout
<<
"--------------------------------------------------------------------------------" << endl;
	cout << endl << "	SystemC 4X4 mesh NOC simulator (By:
Charran Thangeswaran) " << endl;
	cout
<<
"--------------------------------------------------------------------------------" << endl;
```

```
	cout << "This simulator contains 2 5x5 wormhole routers.  " << endl;
	cout << "Assume the router has 5 I/O ports with 4 buffers per input port. " << endl;
	cout << "and each flit has 21 bits width. " << endl;
	cout << "   Press \"Enter\" or \"Return\" to begin simulation..." << endl << endl;

	getchar();
	sc_start(10*125+124,SC_NS); // during [(10*125)+124] ns 10 packets will be sent and received

	sc_close_vcd_trace_file(tf);

	cout << endl << endl << "--------------------------------------------------------------------------------" << endl;
	cout << "End of switch operation..." << endl;
	if(i==0)cout << "Packets sent: " << source0.pkt_snt<< endl;

	if(j==0)cout << "Packets received: " << sink0.pkt_recv<< endl;

	if(i==1)cout << "Packets sent: " << source1.pkt_snt<< endl;

	if(j==1)cout << "Packets received: " << sink1.pkt_recv<< endl;

	if(i==2)cout << "Packets sent: " << source2.pkt_snt<< endl;

	if(j==2)cout << "Packets received: " << sink2.pkt_recv<< endl;

	if(i==3)cout << "Packets sent: " << source3.pkt_snt<< endl;

	if(j==3)cout << "Packets received: " << sink3.pkt_recv<< endl;

	if(i==4)cout << "Packets sent: " << source4.pkt_snt<< endl;

	if(j==4)cout << "Packets received: " << sink4.pkt_recv<< endl;

	if(i==5)cout << "Packets sent: " << source5.pkt_snt<< endl;

	if(j==5)cout << "Packets received: " << sink5.pkt_recv<< endl;

	if(i==6)cout << "Packets sent: " << source6.pkt_snt<< endl;

	if(j==6)cout << "Packets received: " << sink6.pkt_recv<< endl;

	if(i==7)cout << "Packets sent: " << source7.pkt_snt<< endl;

	if(j==7)cout << "Packets received: " << sink7.pkt_recv<< endl;

	if(i==8)cout << "Packets sent: " << source8.pkt_snt<< endl;
	if(j==8)cout << "Packets received: " << sink8.pkt_recv<< endl;
	if(i==9)cout << "Packets sent: " << source9.pkt_snt<< endl;
	if(j==9)cout << "Packets received: " << sink9.pkt_recv<< endl;
	if(i==10)cout << "Packets sent: " << source10.pkt_snt<< endl;
	if(j==10)cout << "Packets received: " << sink10.pkt_recv<< endl;
	if(i==11)cout << "Packets sent: " << source11.pkt_snt<< endl;
	if(j==11)cout << "Packets received: " << sink11.pkt_recv<< endl;
	if(i==12)cout << "Packets sent: " << source12.pkt_snt<< endl;
	if(j==12)cout << "Packets received: " << sink12.pkt_recv<< endl;
	if(i==13)cout << "Packets sent: " << source13.pkt_snt<< endl;
	if(j==13)cout << "Packets received: " << sink13.pkt_recv<< endl;
	if(i==14)cout << "Packets sent: " << source14.pkt_snt<< endl;
	if(j==14)cout << "Packets received: " << sink14.pkt_recv<< endl;
	if(i==15)cout << "Packets sent: " << source15.pkt_snt<< endl;
	if(j==15)cout << "Packets received: " << sink15.pkt_recv<< endl;

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

}
```