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.

1. **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.

1. **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.

1. **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.

1. **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.

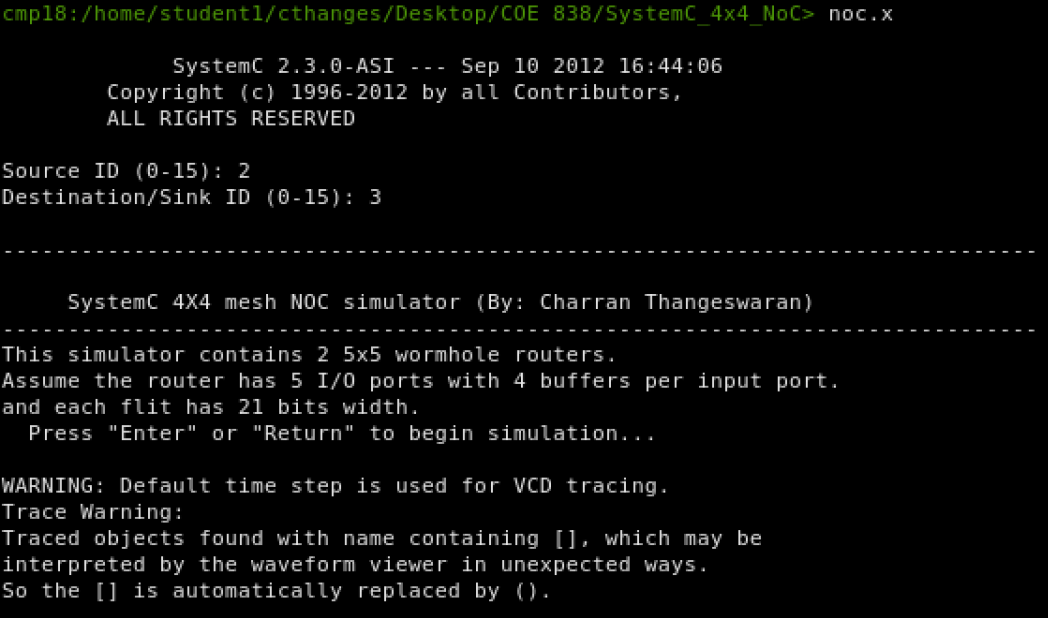
I also changed the crossbar switch module (crossbar.cpp) by first making sure that all routes were supported and reached. In the original code, some values were missing depending on the corresponding v\_cross variable. For example, for v\_cross = i0.read(), a case is not implemented for writing o0.

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.

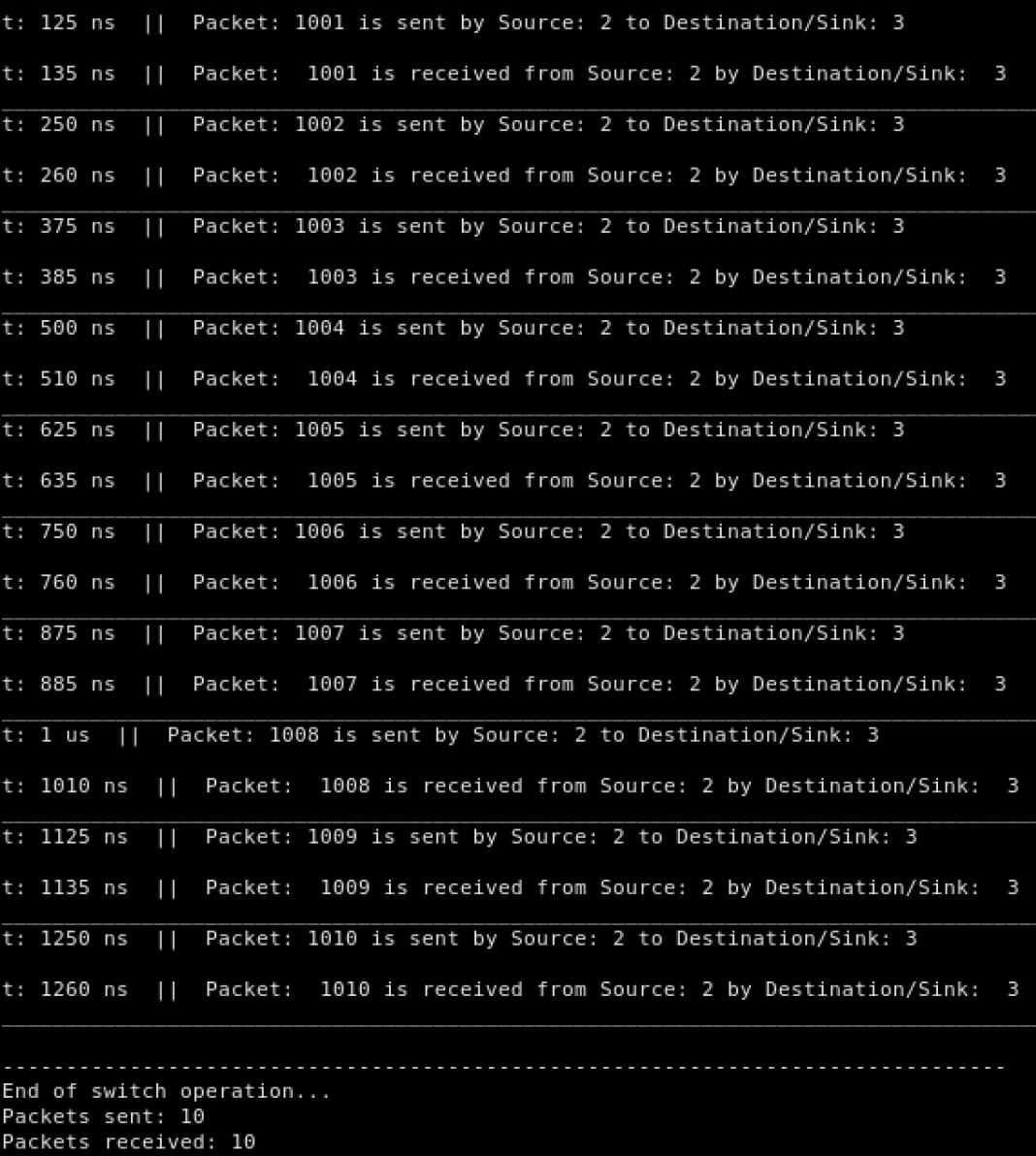
Naturally, the NoC Simulator Main module went through the most changes. Given that a 4x4 mesh NoC is being developed from a 1x2 version, the network size needed to change from 2 sources, sinks, and routers each to 16 of each. In order to do this, I changed the signal arrays values of si\_source, si\_input, 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.

1. **Experimental Results**

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

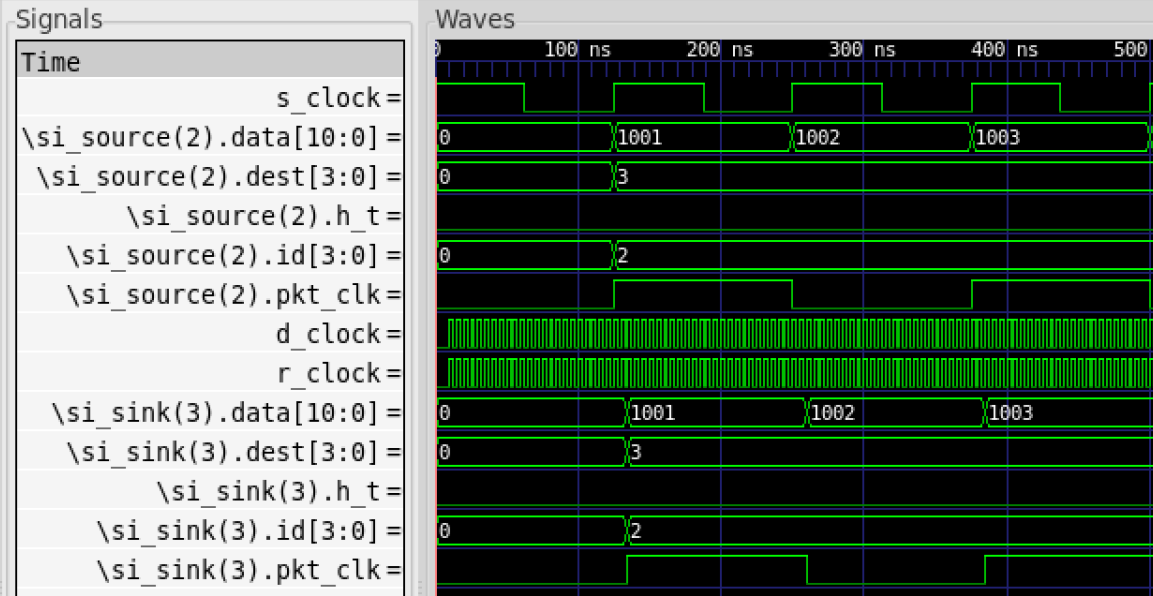


*Figure 1: Start of SystemC 4x4 mesh NoC simulation*



*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.

1. **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.

1. **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

1. **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,siid13,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;

}