Final Project Report: SystemC Based NoC (Network-on-Chip) Modelling Course Project

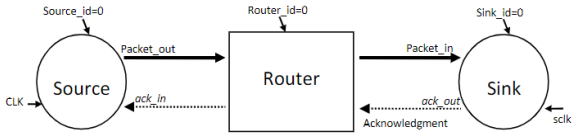
COE838 – System-on-Chip Design

Apurva Patel - 500876938

ABSTRACT--The COE 838 course project focuses on the design/modelling and simulation of a Network-on-Chip (NoC) system using SystemC/C++. This project aims to provide hands-on experience in designing and analysing NoC systems based on the knowledge gained from the previous labs. The initial phase involves understanding the fundamentals of NoC simulation by working with a simple 1×2 mesh NoC design provided, however, the final objective of this project is to develop and design a 4x4 mesh interconnection architecture. The final project report shows the design, approach, theoretical explanations of important components and modules such as the source, sink, and router along with their functionalities and interactions inside the NoC architecture, and the procedures used to execute the project. Along with the theoretical explanations, to support the implementation of this project, the packet routing algorithms, arbitration techniques, and FIFO buffer descriptions are provided in this report. After the analysis, designing and testing a conclusion is drawn from this project. This project is a good exercise to strengthen the basics for further exploration and research in the field of Systems-on-Chip design and network architectures.

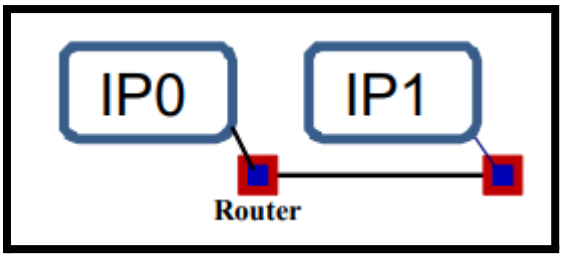
INTRODUCTION

In this project, we explore the complexity of Network-on-Chip (NoC) designs, by utilising SystemC for design implementation and testing. The demand for efficient communication architectures has become significantly important and as the complexity of integrated circuits continues to increase, traditional bus-based communication architectures face challenges in meeting the new requirements of modern applications. In response to this demand, Network-on-Chip (NoC) has emerged as a promising paradigm for on-chip communication, offering scalability, flexibility, and improved performance. Let us begin with understanding what a Network-on-Chip is, it is a communication system/grid that is implemented on a SoC to enable the interconnection of various modules and components within the IC, such as CPU processors, memory units, GPIO's and more. To realise the NoC, different topologies, such as mesh, torus, or hypercube, can be used depending on the requirements of the SoC. There are several factors such as throughput, power consumption, latency, etc. that can decide the use of a specific topology. This is how a NoC implementation looks like:



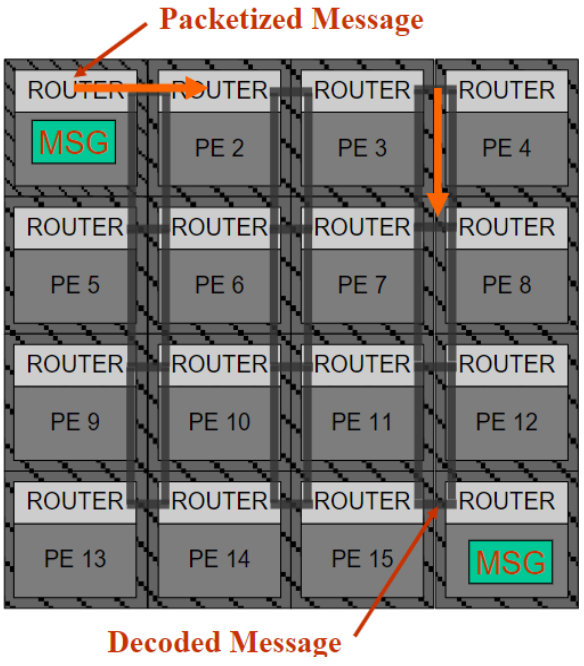
For this project, we explore NoC theory with a particular focus on mesh topologies, including 1×2 and 4×4 configurations, as mentioned in the project manual. Mesh topologies represent a common interconnection structure used in NoC designs due to their simplicity, scalability, and regularity. In a mesh topology, processing elements (PEs) or IP cores are arranged in a grid-like fashion, with each core connected to its

neighbouring cores via communication links or channels. The simplicity of mesh topologies makes them well-suited for both homogeneous and heterogeneous SoC designs. A 1×2 mesh topology consists of two processing elements (PEs) arranged linearly, forming a basic communication path. This minimal configuration provides a foundational understanding of mesh-based NoC architectures, encompassing the essential elements of routers, PEs, and communication links. By modelling and simulating the 1×2 mesh topology, we will gain insights into packet routing, latency, and throughput within a simple NoC environment. This is how a 1x2 mesh topology looks like:



Additionally, various different communication patterns will be discussed, to evaluate the performance and efficiency of the developed NoC design. Furthermore, we will learn about packet structure, routing algorithms, and flow control techniques along with modelling and simulating source, sink, and router modules, and their interactions within the NoC framework.

The below snippet depicts how the the 4x4 Mesh NoC works:



Theory

In this project, we're building different components for a Network-on-Chip (NoC) system that will be part of a larger System-on-Chip (SoC). These components include routers, sources, and sinks. Inside each router, there are smaller parts like arbiters, FIFOs, and crossbars that handle routing and switching. We break down the simulator into modules to represent each part and function of the NoC design. This modular approach focuses on key elements such as packet structure, source and sink modules, and the router module. In the 4x4 mesh, the source and sink modules are essential for data communication. The source module now allows users to choose the source and destination dynamically, empowering them to control traffic flow. Meanwhile, the sink module has been refined to receive and acknowledge packets sent across the network. These enhancements enable the system to handle increased traffic and diverse data patterns effectively. Along with the source and sink, routers are the core elements of the NoC, which require significant enhancements to accommodate the additional connections in the 4x4 mesh. This includes enlarging internal routing tables and refining data streaming logic from various directions. These adjustments improve router efficiency, ensuring optimal performance and avoiding congestion. The modular design, incorporating sub-modules like packet buffering and arbitration, streamlines the enhancement process. Let's discuss these components in the context of developing a 4x4 mesh NoC:

Mesh Topology: In a mesh topology, nodes are arranged in a grid-like fashion, where each node is connected to its adjacent nodes. This arrangement forms a network on which data can be routed from one node to another. The grid structure provides a predictable and scalable architecture for communication within the system. Mesh topologies are commonly used in many systems including computer networks and on-chip interconnects due to their simplicity and efficiency.

Communication Patterns: There are two types of transmission patterns.

- **Uniform Patterns**: In uniform communication patterns, data is transmitted between nodes randomly, without any specific pattern or order. This can simulate a scenario where any node

can communicate with any other node in the network.
- **Neighbouring Patterns**: Neighbouring communication patterns involve data transmission between adjacent nodes in the grid. This pattern is more structured and often represents communication patterns that are common in many applications.

Performance Metrics:

- **Throughput**: Throughput refers to the amount of data transferred successfully between nodes in a given time period. Higher throughput indicates better performance.
- **Latency**: Latency is the time it takes for a packet to travel from its source node to its destination node. Lower latency is desirable as it means faster communication.
- **Energy Consumption**: This metric measures the amount of energy consumed by the NoC during operation. Efficient designs aim to minimise energy consumption.
- **Area Overhead**: Area overhead refers to the additional hardware resources required to implement the NoC. Lower area overhead is preferable as it reduces costs and complexity.

Arbiter: An arbiter is a component responsible for resolving conflicting requests for a shared resource. Arbiters manage access to shared communication resources such as routers or channels. Arbiter designs vary based on factors like fairness, latency, and complexity. Common approaches include round-robin scheduling, priority-based arbitration, and fairness algorithms.

Buffer FIFO: FIFO (First-In-First-Out) buffers are used to temporarily store data packets in a sequential manner. In NoCs, FIFO buffers are often used within routers or switches to queue incoming packets until they can be forwarded to their next hop. Buffer management strategies play a crucial role in maintaining efficient data flow and avoiding congestion within the network.

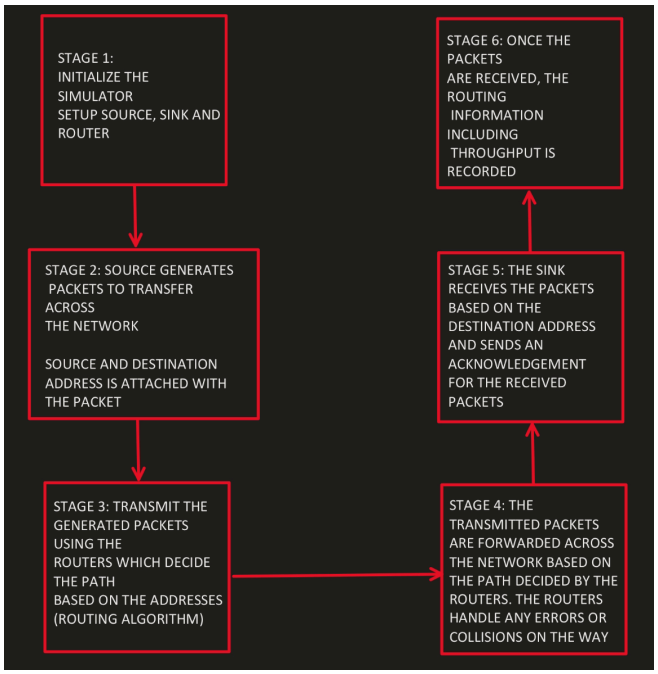
Crossbar: A crossbar switch is a hardware component that allows multiple inputs to be connected to multiple outputs in a non-blocking manner. In NoCs, crossbars are commonly used within routers to enable simultaneous communication between multiple nodes. Crossbars facilitate efficient data routing by providing multiple paths for packets to traverse through the network.

Packet Structure and Communication: The NoC simulator's core modules enable communication between components through ports and handle computational tasks concurrently. Packet structure defines how data moves within the NoC, with headers guiding data routing. This structure is defined in SystemC code, specifying parameters like addresses, packet size, and clock synchronisation.

Source Module: The source module generates packets based on a predefined communication pattern. It handles packet generation, acknowledgment reception, and clock synchronisation to initiate data transmission within the NoC. Similarly, the sink module records incoming packets and sends confirmation signals back to the router.

Router Module: The router module acts as the central nervous system of the NoC, managing packet movement between networked components. It includes smaller routing units like arbiters, crossbars, and FIFO caches to allocate resources and ensure fast connectivity. Understanding concepts like arbitration, routing methods, and component connections is crucial for efficient NoC modelling and simulation.

By exploring the workings of these NoC components, we gain insights into designing efficient on-chip communication systems. Here is a flow chart depicting a summary of the theory and introduction of this project:



Methodology

Expanding the Network-on-Chip (NoC) design from a 1x2 to a 4x4 mesh topology requires a strategic approach to ensure smooth transition and improved network performance. The process begins with adapting the design to accommodate the larger scale, involving a thorough redesign to incorporate more routers, source, and sink modules, and ensuring their connectivity suits the expanded architecture.

After the design adaptation, the focus shifts to enhancing modules, particularly the arbiter module, with advanced routing logic to handle the increased routing decisions in the 4x4 mesh. Simultaneously, upgrades are made to the source and sink modules to manage higher data throughput effectively.

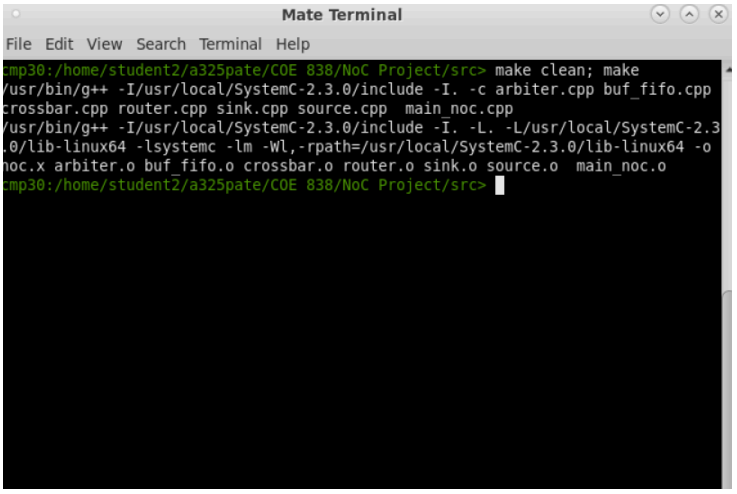
The implementation phase employs SystemC to model and simulate the enhanced NoC design, translating theoretical modifications into a practical framework. This step is crucial for verifying the design's functionality and preparing for thorough testing and analysis.

Detailed simulations are conducted to assess the 4x4 mesh NoC's performance under various scenarios, particularly in managing increased data flow and identifying potential congestion points. These simulations aim to gain a comprehensive understanding of the network's behavior under different conditions and pinpoint areas for further optimization.

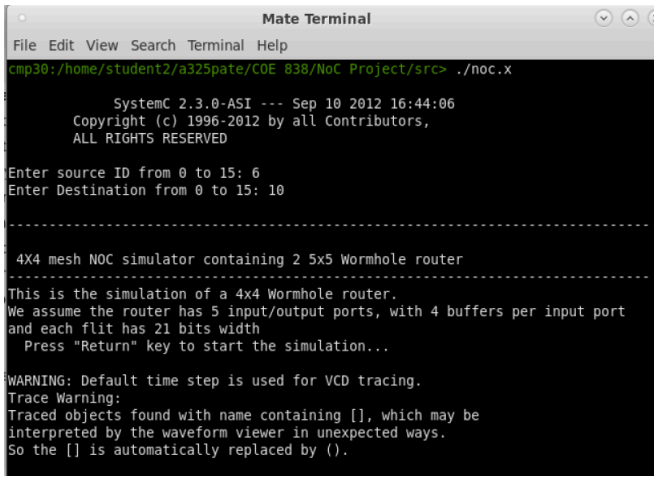
Through this structured methodology, including design adaptation, module enhancement, SystemC implementation, and detailed simulations, the goal is to navigate the complexities of scaling the NoC meticulously. The objective is to ensure that the transition to a 4x4 mesh topology enhances the network's performance, preparing it for future challenges and advancements in NoC technology.

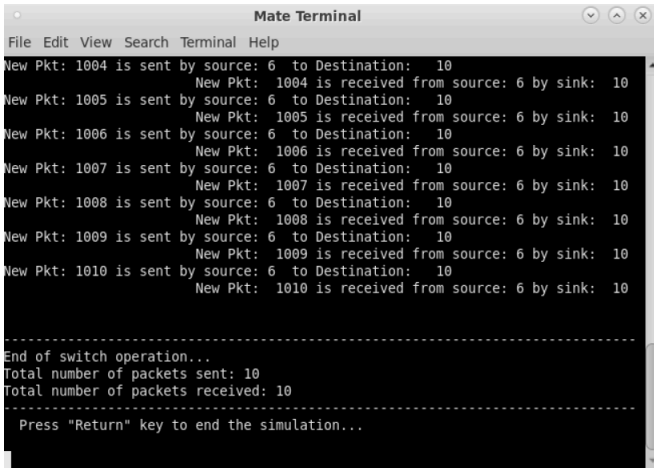
Procedure

> Open a terminal directing to the source code and run a make clean:and make

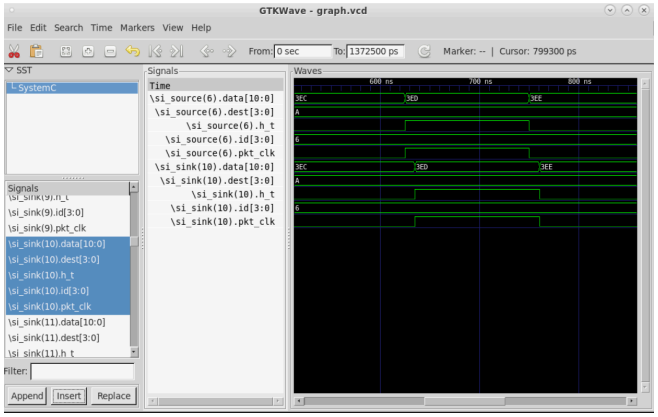


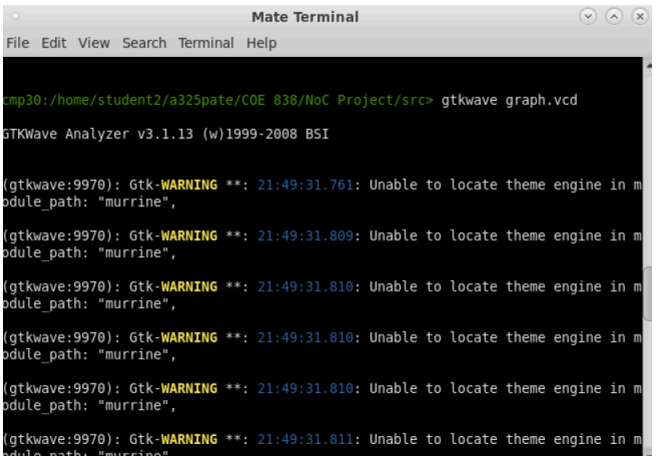
> Run the “./noc.x” command to execute the compiled executable file which will run the NoC simulation. The Sink and Source are user input enabled so the user can select the Source and Destination between 0 and 15.



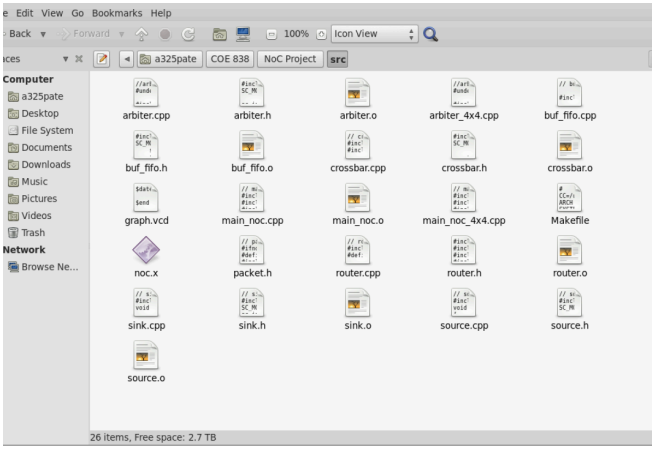


> To analyse the entire process and generate a trace file waveform run: gtkwave graph.vcd

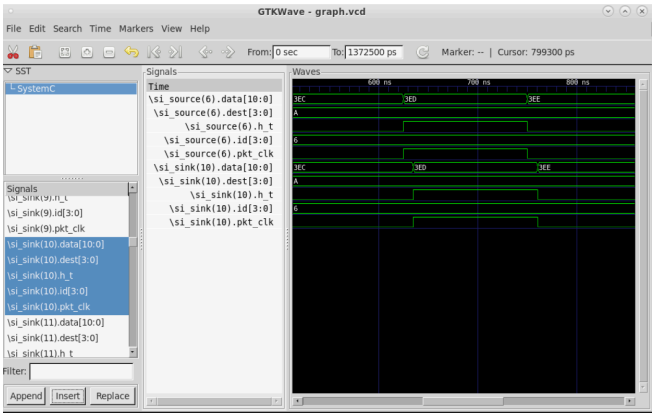




> Overall, these are the files required to execute this project successfully.:



Observation



As you can see the Sink number 10 is selected and Source is 6. The data packets with the source and destination address are generated by the Source and transmitted through the routers, the packets travel forward based on the destination address. If the receiving router has the matching destination address, it will send

back an acknowledgement to the source, marking the end of a transaction. There is a slight delay when the packet is received by the sink, this is due to propagation and routing/switching delays which are caused when the packet switches from one router to the other. Each router is responsible to make sure it is forwarding the packets through the most efficient path.

Conclusion

Overall, the project gave a hands-on experience in understanding and developing an Network-On-Chip simulator. The project's complexity was reduced as the 1x2 Mesh topology simulator was provided for reference and understanding purposes, and the 4x4 Mesh NoC simulator was developed based on the provided design.

References

[1] Khan, Dr. Gul. (n.d.). Course Project: SystemC based NoC (Network-on-Chip) Modeling. COE838: Systems-On-Chip Design.https://courses.torontomu.ca/d2l/le/content/835859/viewContent/5453613/View

[2] Khan, Dr. Gul. (n.d.). Course Notes: Systems-On-Chip Design Lecture Notes. COE838:Systems-On-Chip Design.https://www.ee.torontomu.ca/~courses/coe838/lecture-notes.html

Appendix

> arbiter.cpp:

```
//arbiter.cpp
#undef SC_INCLUDE_FX

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

void arbiter :: func()
{
    sc_uint<1> v_connected_input[5];
//set when input is connected to an
output
    sc_uint<1> v_reserved_output[6];
//set when output is reserved by a
input (one output more for simple
coding)
    sc_uint<3> v_req[5];
    sc_uint<5> v_free;  // status of
output in term of being free
    sc_uint<4> v_id;
    sc_uint<5> v_arbit;
    sc_uint<15> v_select;
    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[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{
                    if(v_id[1] <
req0.read()[1])v_req[0]=4; // go to
south
                    else{
                        if(v_id[1] >
req0.read()[1])v_req[0]=2; //go to
north
                        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_outp
ut[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[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 {
```

```
                    if(v_id[1] <
req1.read()[1])v_req[1]=4; // go to
south
                    else {
                        if(v_id[1] >
req1.read()[1])v_req[1]=2; //go to
north
                        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[0] <
req2.read()[0]) v_req[2]=3; // go to
east
            else {
                if(v_id[0] >
req2.read()[0])v_req[2]=5; //go to
west
                else {
                    if(v_id[1] <
req2.read()[1])v_req[2]=4; // go to
south
                    else {
                        if(v_id[1] >
req2.read()[1])v_req[2]=2; //go to
north
                        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 (!req3.read()[4]) //if
buffer is not empty
        {
            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 {
                    if(v_id[1] <
req3.read()[1])v_req[3]=4; // go to
south
                    else {
                        if(v_id[1] >
req3.read()[1])v_req[3]=2; //go to
north
                        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_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 {
                    if(v_id[1] <
req4.read()[1])v_req[4]=4; // go to
south
                    else {
                        if(v_id[1] >
req4.read()[1])v_req[4]=2; //go to
north
                        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);
    }
}
```

> arbiter.h:

```
#include "systemc.h"
SC_MODULE(arbiter) {

    sc_in<sc_uint<4> > arbiter_id;
    sc_in<sc_uint<7>  >  req0;
    sc_in<sc_uint<7>  >  req1;
    sc_in<sc_uint<7>  >  req2;
    sc_in<sc_uint<7>  >  req3;
    sc_in<sc_uint<7>  >  req4;

    sc_in<bool  >  free_out0;
    sc_in<bool  >  free_out1;
    sc_in<bool  >  free_out2;
    sc_in<bool  >  free_out3;
    sc_in<bool  >  free_out4;

    sc_out<sc_uint<15> >  aselect;

    sc_out<sc_uint<1> >  grant0;
    sc_out<sc_uint<1> >  grant1;
    sc_out<sc_uint<1> >  grant2;
    sc_out<sc_uint<1> >  grant3;
    sc_out<sc_uint<1> >  grant4;

    sc_in<bool> aclk;

    void func();

    SC_CTOR(arbiter)
     {
      SC_THREAD(func);
      sensitive << aclk.neg();
    }
};
```

> main.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[])
{
    sc_signal<packet> si_source[4];
    sc_signal<packet> si_input[16];
    sc_signal<packet> si_zero[16];
    sc_signal<packet> si_sink[4];
    sc_signal<packet> si_output[16];
    sc_signal<bool>
si_ack_src[4],si_ack_ou[16];
    sc_signal<bool>
si_ack_sink[4],si_ack_in[16];
    sc_signal<bool>  si_ack_zero[16];
    sc_signal<sc_uint<4> >
siid0,siid1,siid2,siid3;
    sc_signal<sc_uint<4> >
scid0,scid1, scid2, scid3;
    sc_signal<sc_uint<4> > id0,id1,
id2, id3;
    sc_signal<int> scinput;
    sc_signal<sc_uint<4> >
check,check2,check3,check4;
    sc_signal <packet> sioutput[4];
    int i,j;
    sc_clock s_clock("S_CLOCK", 125,
SC_NS, 0.5, 0.0, SC_NS); // source
clock
    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);

    router router0("router0");
    // hooking up signals to ports by
name
    router0.in0(si_source[0]);
    router0.in1(si_output[2]);
    router0.in2(si_output[4]);
    router0.in3(si_zero[0]);
    router0.in4(si_zero[1]);

    router0.router_id(id0);

    router0.out0(si_sink[0]);
    router0.out2(si_output[0]);
    router0.out3(si_output[1]);
    router0.out1(si_zero[2]);
    router0.out4(si_zero[3]);

    router0.inack0(si_ack_sink[0]);
    router0.inack1(si_ack_in[2]);
    router0.inack2(si_ack_in[4]);
    router0.inack3(si_ack_zero[0]);
    router0.inack4(si_ack_zero[1]);

    router0.outack0(si_ack_src[0]);
    router0.outack2(si_ack_in[0]);
    router0.outack3(si_ack_in[1]);
    router0.outack1(si_ack_zero[2]);
    router0.outack4(si_ack_zero[3]);

    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[6]);
    router1.in3(si_zero[4]);
    router1.in4(si_zero[5]);


    router1.router_id(id1);

    router1.out0(si_sink[1]);
    router1.out4(si_output[2]);
    router1.out3(si_output[3]);
    router1.out1(si_zero[6]);
    router1.out2(si_zero[7]);

    router1.inack0(si_ack_sink[1]);
    router1.inack1(si_ack_in[0]);
    router1.inack2(si_ack_in[6]);
    router1.inack3(si_ack_zero[4]);
    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_zero[6]);
    router1.outack1(si_ack_zero[7]);

    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[1]);
    router2.in2(si_output[7]);
    router2.in3(si_zero[8]);
    router2.in4(si_zero[9]);


    router2.router_id(id2);

    router2.out0(si_sink[2]);
    router2.out1(si_output[4]);
    router2.out2(si_output[5]);
    router2.out3(si_zero[10]);
    router2.out4(si_zero[11]);


    router2.inack0(si_ack_sink[2]);
    router2.inack1(si_ack_in[1]);
    router2.inack2(si_ack_in[7]);
    router2.inack3(si_ack_zero[8]);
    router2.inack4(si_ack_zero[9]);


    router2.outack0(si_ack_src[2]);
    router2.outack1(si_ack_in[4]);
    router2.outack2(si_ack_in[5]);
    router2.outack3(si_ack_zero[10]);
    router2.outack4(si_ack_zero[11]);


    router2.rclk(r_clock);

    router router3("router3");
    // hooking up signals to ports by
name
    router3.in0(si_source[3]);
    router3.in1(si_output[3]);
    router3.in2(si_output[5]);
    router3.in3(si_zero[12]);
    router3.in4(si_zero[13]);

    router3.router_id(id3);
```

```
    router3.out0(si_sink[3]);
    router3.out1(si_output[6]);
    router3.out4(si_output[7]);
    router3.out2(si_zero[14]);
    router3.out3(si_zero[15]);


    router3.inack0(si_ack_sink[3]);
    router3.inack1(si_ack_in[3]);
    router3.inack2(si_ack_in[5]);
    router3.inack3(si_ack_zero[12]);
    router3.inack4(si_ack_zero[13]);


    router3.outack0(si_ack_src[3]);
    router3.outack1(si_ack_in[6]);
    router3.outack4(si_ack_in[7]);
    router3.outack2(si_ack_zero[14]);
    router3.outack3(si_ack_zero[15]);

    router3.rclk(r_clock);

    //need codes

    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]);
```

```
  // 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]");
    //need codes
    //need codes
    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]");
    //need codes
    //need codes

    cout<< "Enter source ID from 0 to
3: ";
    cin >> i;
    cout<<"Enter Destination from 0 to
3: ";
    cin >> j;
    scinput.write(j);

    id0.write(0);
    id1.write(1);
    id2.write(2);
    id3.write(3);

    check.write(i);
```

```
    scid0.write(0);
    scid1.write(1);
    scid2.write(2);
    scid3.write(3);

    siid0.write(0);
    siid1.write(1);
    siid2.write(2);
    siid3.write(3);
    //need codes
    //need codes

    cout << endl;
    cout <<
"-------------------------------------
-------------------------------------
----" << endl;
    cout << endl << " 2X2 mesh NOC
simulator containing 2 5x5 Wormhole
router " << endl;
    cout <<
"-------------------------------------
-------------------------------------
----" << endl;
    cout << "This is the simulation of
a 2x2 Wormhole router.  " << endl;
    cout << "We assume the router has
5 input/output ports, with 4 buffers
per input port " << endl;
    cout << "and each flit has 21 bits
width " << endl;
    cout << "  Press \"Return\" key to
start the 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 << "Total number of
packets sent: " <<  source0.pkt_snt<<
endl;//need codes to be added
    if(j==0)cout << "Total number of
packets received: " <<
sink0.pkt_recv<< endl;//need codes to
be added
    if(i==1)    cout << "Total number
of packets sent: " <<
source1.pkt_snt<< endl;//need codes to
be added
    if(j==1)cout << "Total number of
packets received: " <<
sink1.pkt_recv<< endl;//need codes to
be added
    if(i==2)    cout << "Total number
of packets sent: " <<
source2.pkt_snt<< endl;//need codes to
be added
    if(j==2)cout << "Total number of
packets received: " <<
sink2.pkt_recv<< endl;//need codes to
be added
    if(i==3)    cout << "Total number
of packets sent: " <<
source3.pkt_snt<< endl;//need codes to
be added
    if(j==3)cout << "Total number of
packets received: " <<
sink3.pkt_recv<< endl;//need codes to
be added

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

}
```

> 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


    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=tru
```

```
e; // make tail flit (the packet size
is 5)

packet_out.write(v_packet_out);

            cout << "New Pkt: " <<
v_packet_out.data << " is sent by
source: " << source_id.read()  << "
to Destination:   "<<
v_packet_out.dest <<endl;

exclode:;
                }
            }
        }
    }
```

> source.h:

```
// source.h
#include "packet.h"
SC_MODULE(source) {

  sc_out<packet> packet_out;
  sc_in<sc_uint<4> > source_id;
  sc_in<bool > ach_in;       // input
acknowledgment
  sc_in_clk CLK;
  sc_in<int> d_est; //destination
signal
  sc_in<sc_uint<4> > ch_k; //Check or
Compare variable
  int pkt_snt;           // variable
for recording of packet sent

  void func();

  SC_CTOR(source)
    {
      SC_CTHREAD(func, CLK.pos());
    pkt_snt=0;
    }
};
```

> sink.cpp:

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

    packet v_packet;
    if ( sclk.event() )
ack_out.write(false);
    if (packet_in.event() ) {
        pkt_recv++ ;
        ack_out.write(true);
        v_packet= packet_in.read();
        cout << "           New Pkt:
" << (int)v_packet.data<< " is
received from source: " <<
(int)v_packet.id  << " by sink:  " <<
(int)sink_id.read() << endl;
  }
}
```

>sink.h:

```
// sink.h
#include "packet.h"
SC_MODULE(sink) {
    sc_in<packet> packet_in;    //
input port
    sc_out<bool> ack_out;       //
output port
    sc_in<sc_uint<4> > sink_id;
    sc_in<bool> sclk;
    sc_out<packet> packet_out_sink;
    int pkt_recv;

    void receive_data();
    // Constructor
    SC_CTOR(sink) {
        SC_METHOD(receive_data); //
Method Process
        dont_initialize();
        sensitive << packet_in;
        sensitive << sclk.pos();
        pkt_recv = 0;
    }
```

```
};
```

> router.cpp:

```
// router.cpp
#include "router.h"
#define SIM_NUM 30

void router :: func()
{
  int sim_count;

  // FILE *result;

  // initialization

  sim_count  = 0;

  // result = fopen("result","w");



  // functionality
  while( sim_count++ < SIM_NUM )
    {
       wait();
      if
(in0.event()){pkt_sent++;}//cout <<  "
source "<< in0.read().id << "
router" << router_id.read()<< endl;}
    if
(in1.event()){pkt_sent++;}//cout << "
source "<< in1.read().id << "
router" << router_id.read()<< endl;}
    if
(in2.event()){pkt_sent++;}//cout << "
source "<< in2.read().id << "
router" << router_id.read()<< endl;}
    if
(in3.event()){pkt_sent++;}//cout << "
source "<< in3.read().id << "
router" << router_id.read()<< endl;}
    if
(in4.event()){pkt_sent++;}//cout << "
source "<< in4.read().id << "
router" << router_id.read()<< endl;}

    }

 // sc_stop();

}
```

> router.h:

```
#include "packet.h"
#include "buf_fifo.h"
#include "crossbar.h"
#include "arbiter.h"
SC_MODULE(router) {
    sc_in<packet>  in0;
    sc_in<packet>  in1;
    sc_in<packet>  in2;
    sc_in<packet>  in3;
    sc_in<packet>  in4;

    sc_in<sc_uint<4> > router_id;

    sc_out<packet>  out0;
    sc_out<packet>  out1;
    sc_out<packet>  out2;
    sc_out<packet>  out3;
    sc_out<packet>  out4;

    sc_in<bool>  inack0;
    sc_in<bool>  inack1;
    sc_in<bool>  inack2;
    sc_in<bool>  inack3;
    sc_in<bool>  inack4;

    sc_out<bool>  outack0;
    sc_out<bool>  outack1;
    sc_out<bool>  outack2;
    sc_out<bool>  outack3;
    sc_out<bool>  outack4;

    sc_in<bool> rclk;
```

```
    buf_fifo* buf0;
    buf_fifo* buf1;
    buf_fifo* buf2;
    buf_fifo* buf3;
    buf_fifo* buf4;

    arbiter* arbiter0;
    crossbar* crossbar0;

    sc_signal<sc_uint<7>  >  req_s_0;
    sc_signal<sc_uint<7>  >  req_s_1;
    sc_signal<sc_uint<7>  >  req_s_2;
    sc_signal<sc_uint<7>  >  req_s_3;
    sc_signal<sc_uint<7>  >  req_s_4;

    sc_signal<sc_uint<4>  >  free_s;
    sc_signal<sc_uint<15>  >
select_s;

    sc_signal<sc_uint<1> >  gr_s_0;
    sc_signal<sc_uint<1> >  gr_s_1;
    sc_signal<sc_uint<1> >  gr_s_2;
    sc_signal<sc_uint<1> >  gr_s_3;
    sc_signal<sc_uint<1> >  gr_s_4;

    sc_signal<packet>  re_s_0;
    sc_signal<packet>  re_s_1;
    sc_signal<packet>  re_s_2;
    sc_signal<packet>  re_s_3;
    sc_signal<packet>  re_s_4;

    void func();
    int pkt_sent;

    SC_CTOR(router)
        {
        buf0 = new buf_fifo ("buf0");
        buf0->wr(in0);
        buf0->re(re_s_0);
        buf0->ack(outack0);
        buf0->req(req_s_0);
        buf0->grant(gr_s_0);
        buf0->bclk(rclk);
```

```
        buf1 = new buf_fifo ("buf1");
        buf1->wr(in1);
        buf1->re(re_s_1);
        buf1->ack(outack1);
        buf1->req(req_s_1);
        buf1->grant(gr_s_1);
        buf1->bclk(rclk);

        buf2 = new buf_fifo ("buf2");
        buf2->wr(in2);
        buf2->re(re_s_2);
        buf2->ack(outack2);
        buf2->req(req_s_2);
        buf2->grant(gr_s_2);
        buf2->bclk(rclk);

        buf3 = new buf_fifo ("buf3");
        buf3->wr(in3);
        buf3->re(re_s_3);
        buf3->ack(outack3);
        buf3->req(req_s_3);
        buf3->grant(gr_s_3);
        buf3->bclk(rclk);

        buf4 = new buf_fifo ("buf4");
        buf4->wr(in4);
        buf4->re(re_s_4);
        buf4->ack(outack4);
        buf4->req(req_s_4);
        buf4->grant(gr_s_4);
        buf4->bclk(rclk);

        arbiter0 = new arbiter
("arbiter0");


arbiter0->arbiter_id(router_id);

        arbiter0->free_out0(inack0);
        arbiter0->free_out1(inack1);
        arbiter0->free_out2(inack2);
        arbiter0->free_out3(inack3);
        arbiter0->free_out4(inack4);
```

```
        arbiter0->req0(req_s_0);
        arbiter0->req1(req_s_1);
        arbiter0->req2(req_s_2);
        arbiter0->req3(req_s_3);
        arbiter0->req4(req_s_4);

        arbiter0->grant0(gr_s_0);
        arbiter0->grant1(gr_s_1);
        arbiter0->grant2(gr_s_2);
        arbiter0->grant3(gr_s_3);
        arbiter0->grant4(gr_s_4);

        arbiter0->aselect(select_s);

        arbiter0->aclk(rclk);

        crossbar0 = new crossbar
("crossbar0");
        crossbar0->i0(re_s_0);
        crossbar0->i1(re_s_1);
        crossbar0->i2(re_s_2);
        crossbar0->i3(re_s_3);
        crossbar0->i4(re_s_4);

        crossbar0->o0(out0);
        crossbar0->o1(out1);
        crossbar0->o2(out2);
        crossbar0->o3(out3);
        crossbar0->o4(out4);

        crossbar0->config(select_s);

        SC_THREAD(func);
        sensitive << in0 << in1 << in2
<< in3 << in4;
        pkt_sent = 0;
    }
};
```

> packet.h:

```
// packet.h file
```

```
#ifndef PACKET
#define PACKET
#include "systemc.h"
struct packet {
       sc_uint<11> data;
       sc_uint<4> id;           //
packet source ID
       sc_uint<4> dest;         //
packet destination ID
       sc_uint<1> pkt_clk;      // bit
for changing the new packet condition
       sc_uint<1> h_t;          //
header or tail flit("1" represents
tail)
       inline bool operator == (const
packet& rhs) const
        {
        return (rhs.data == data &&
rhs.id == id && rhs.dest == dest &&
rhs.pkt_clk == pkt_clk && rhs.h_t ==
h_t);
        }

};

inline
ostream&
operator << ( ostream& os, const
packet& a )
{
    os << "streaming of struct packet
not implemented";
    return os;
}

inline
void
#if defined(SC_API_VERSION_STRING)
    sc_trace( sc_trace_file* tf, const
packet& a, const std::string& name )
#else
    sc_trace( sc_trace_file* tf, const
packet& a, const sc_string& name )
#endif
```

```
{
  sc_trace( tf, a.data, name + ".data"
);
  sc_trace( tf, a.id, name + ".id" );
  sc_trace( tf, a.dest, name + ".dest"
);
  sc_trace( tf, a.pkt_clk, name +
".pkt_clk" );
  sc_trace( tf, a.h_t, name + ".h_t"
);
}


#endif
```

> 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 <<
"---------------------------------wron
g 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 <<
"---------------------------------w
rong 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 <<
"---------------------------------wron
g 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 <<
"---------------------------------wro
ng 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 ;
            }
        }
    }
```

```
}
```

> crossbar.h:

```
#include "packet.h"
SC_MODULE(crossbar) {

    sc_in<packet>  i0;
    sc_in<packet>  i1;
    sc_in<packet>  i2;
    sc_in<packet>  i3;
    sc_in<packet>  i4;

    sc_out<packet>  o0;
    sc_out<packet>  o1;
    sc_out<packet>  o2; //error here
    sc_out<packet>  o3; // error here
    sc_out<packet>  o4;

    sc_in<sc_uint<15> >  config;

    void func();
    SC_CTOR(crossbar)
        {
        SC_THREAD(func);
        sensitive << i0;
        sensitive << i1;
        sensitive << i2;
        sensitive << i3;
        sensitive << i4;
    }
};
```

> buf_FIFO.cpp:

```
// buf_fifo.cpp

#include "buf_fifo.h"

    void fifo::packet_in(const packet&
data_packet)
    {
      registers[reg_num++] =
data_packet;
      empty = false;
```

```
      if (reg_num == 4) full = true;
    }

    packet fifo::packet_out()
    {
        reg_num--;
        packet temp;
        temp = registers[0];
        if (reg_num == 0) empty =
true;
        else
    {
        registers[0] = registers[1];
        registers[1] = registers[2];
        registers[2] = registers[3];
        }
       full = false;
      return(temp);
    }

void buf_fifo :: func()
{
    fifo q0;     // define a FIFO
buffer
    q0.reg_num = 0;
    q0.full  = false;
    q0.empty = true;
    packet b_temp;

req.write((q0.registers[0].h_t,q0.empt
y, q0.registers[0].dest));// this
outputs header or tail flit while
empty and destination ID

    // functionality
    while( true )
    {
        wait();


        /////received packet/////
        if (wr.event())
        {
            q0.packet_in(wr.read());
            ack.write(q0.full);

req.write((q0.registers[0].h_t,
q0.empty, q0.registers[0].dest));
        }
        ///// sent packets out/////
        if (bclk.event())
        {
            if(grant.read() == 1)
            {
                b_temp =
q0.packet_out();
                re.write(b_temp);
                ack.write(q0.full);

req.write((q0.registers[0].h_t,
q0.empty, q0.registers[0].dest));
            }
        }
    }
}
```

> buf_FIFO.h:

```
#include "packet.h"
SC_MODULE(buf_fifo) {
    sc_in   <packet>         wr;
    sc_out  <packet>         re;
    sc_in   <sc_uint<1> >    grant;
    sc_out  <sc_uint<7> >    req;
    sc_out  <bool>           ack;
    sc_in   <bool>           bclk;

    void func();

    SC_CTOR(buf_fifo)
     {
      SC_THREAD(func);
      sensitive << wr;
      sensitive<< bclk.pos();
    }
};

struct fifo {
```

```
public:

   packet registers[4];
   bool full;
   bool empty;
   int reg_num;

   // constructor

   fifo()
    {
      full = false;
      empty = true;
      reg_num = 0;
    };

    // methods

    void packet_in(const packet&
data_packet);
    packet packet_out();
};
```

>