Ryerson University
Department of Electrical, Computer & Biomedical Engineering
Faculty of Engineering & Architectural Science

COE838 – Systems-on-Chip

LAB REPORT

Semester/Year:	Winter 2025
Lab Number:	Interim Report

Instructor:	Dr. Gul Khan
Section No:	032
Submission Date:	2025-03-29
Due Date:	2025-03-26

Student Name	Student ID	Signature
Carlo Dumlao	501018239	C.D

Interim Report: SystemC based NoC

Carlo Dumlao

Toronto Metropolitan University

I. Introduction

The objective of the initial weeks of the project is to model and design a 1x2 mesh topology in SystemC, which will provide a great foundation for developing the actual 4x4 mesh and 4x4 torus topology. To simplify, this 1x2 mesh NoC was constructed to contain only two routers, as well as a source and sink module to represent the two IP cores. For simulation purposes, the source module was designed to always generate a random packet, and then be received by the sink module. The behaviour and performance of the overall NoC was analyzed by generating the waveform using the gtkwave program.

II. Theoretical

A. Source Module

The flow diagram of the source module is shown in Figure 1. Notice that the source module firstly checks whether the router FIFO is full or not. If it is full, the source will remain at idle until the router gives permission to send data again. On the other hand, if it is not full, the source will immediately generate a new packet to be sent to the sink module. The packet has a length of 21 bits and formatted as follows:

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

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

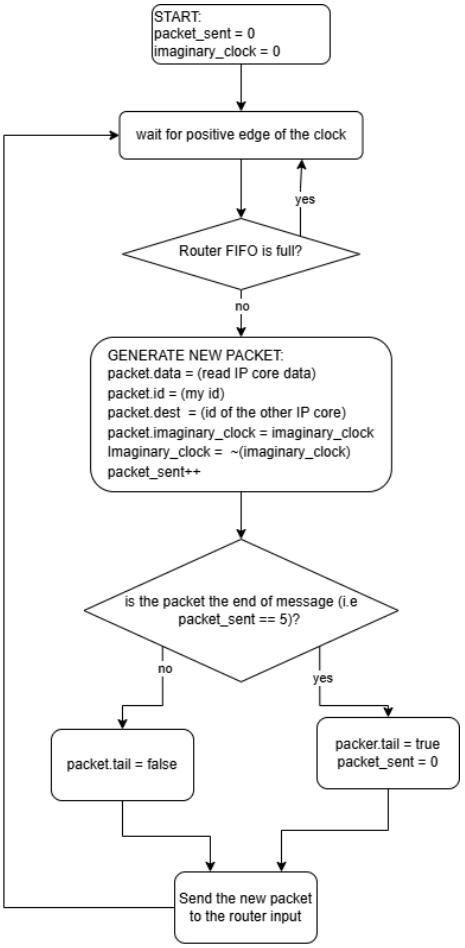

Fig. 1 Flow diagram of the source module

B. Router Architecture

The complete architecture of the router is presented in Figure 2 below. Three components are crucial for the functionality of this module, including a FIFO, arbiter, and crossbar.The FIFO component contains four registers which are used to temporarily store the incoming packets. The arbiter is responsible for configuring the crossbar such that a packet from the FIFO is correctly delivered to the correct path (or destination). Meanwhile, the crossbar is similar to a multiplexer which physically connects the FIFO into one of the external data outputs of the router.

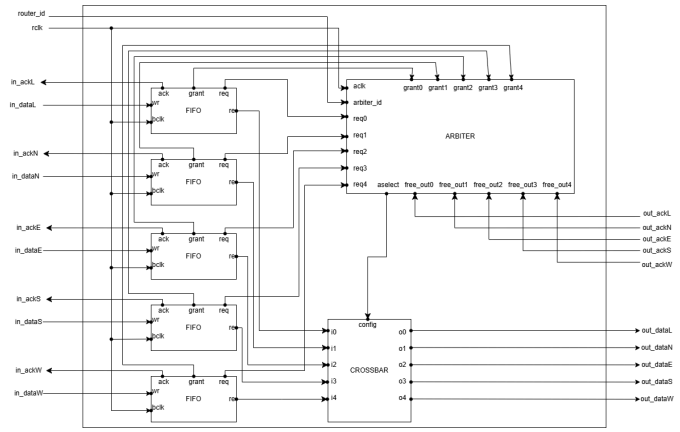

Fig. 2 Complete architecture of the router

III. Works Progress

A. Waveform

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

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

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

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

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

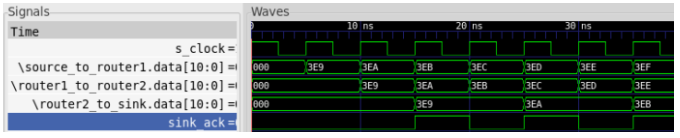

Fig. 3 Generated waveform of the 1x2 NoC system

B. Performance

The average packet delay can be calculated and printed into the terminal by introducing a new variable to both the source and sink modules:

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

As shown in Figure 4, it requires 75 clock cycles to successfully deliver a message (5 packets) from the source into the sink – assuming that the routers are experiencing a heavy communication traffic (i.e. buffers are occasionally full).

```
New Pkt: 183 is sent by source: 0  to Destination:   1
                         New Pkt:  169 is received from source: 0 by sink:  1
Flint delay: 184-169=15
New Pkt: 185 is sent by source: 0  to Destination:   1
                         New Pkt:  171 is received from source: 0 by sink:  1
Flint delay: 186-171=15
New Pkt: 187 is sent by source: 0  to Destination:   1
                         New Pkt:  173 is received from source: 0 by sink:  1
Flint delay: 188-173=15
New Pkt: 189 is sent by source: 0  to Destination:   1
                         New Pkt:  175 is received from source: 0 by sink:  1
Flint delay: 190-175=15
New Pkt: 191 is sent by source: 0  to Destination:   1
                         New Pkt:  177 is received from source: 0 by sink:  1
Flint delay: 192-177=15
Packet delay: 75
```

Fig. 4 Output terminal showing the calculated packet delay.

C. Modified Arbiter

In the original arbiter module, each body flit is required to carry a destination ID so that the module is able to determine the right path – however, this method is redundant. Figure 5 shows the improved version of the module. Notice that when the header packet is detected (i.e. a new message), its destination ID is temporarily stored in a 2-bit *header0* variable. The module would then make use of this value to determine the destination of the incoming payload packets. Conversely, once a tail packet is detected, the variable is reseted and prepared for the next header packet.

```
if(!v_connected_input[0]){
    header0 = req0.read().range(1,0);
}

if(v_id[0] < header0[0]) v_req[0]=3; // go to east
else {
    if(v_id[0] > header0[0])v_req[0]=5; //go to west
    else{
        if(v_id[1] < header0[1])v_req[0]=4; // go to south
        else{
            if(v_id[1] > header0[1])v_req[0]=2; //go to north
            else v_req[0]=1; // that is the destination
        }
    }
}
```

Fig. 5 Portion of the arbiter code responsible for identify the destination

IV. Works Pending

The works pending are as follows:

- Design and model a more complex and larger NoC – that is, a 4x4 mesh and 4x4 torus topology.
- Simulate their functionality and behaviour with various types of communication patterns, including a uniform and neighbouring pattern.
- Likewise, obtain their performance such as the average packet delay.

V. Tentative Plan

A. 4x4 Mesh Topology

1) Integrate the designed source, sink, and router to form a single module that would represent the complete IP core within the NoC. (Note that the IP core used in this report is incomplete, only consisting of a source or a sink).
2) To ensure that all of the signals are correctly connected, the NoC will be expanded gradually from a 1x2 topology to a 4x4 topology:
    a) Combine the created IP cores to form a 1x2 mesh system.
    b) Using the existing 1x2 mesh system, develop a 2x2 mesh module.
    c) Combine 4 2x2 mesh modules to create the desired 4x4 mesh topology.
3) Generate the necessary packets that would accomplish the uniform and neighbouring pattern.

B. 4x4 Torus Topology

1) Slightly modify the connections of the 4x4 mesh such that the edges of the NoC are wrapped around forming a closed loop.
2) Determine an appropriate algorithm that would be implemented in the arbiter; modify the arbiter module such that it also takes into account the new loop channels when determining the correct and shortest path (ie. Local, North, East, South, and West) to deliver the packet.

Appendix

A. arbiter.cpp

```
//arbiter.cpp
#undef SC_INCLUDE_FX

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

void arbiter :: func()
{
          sc_uint<1> v_connected_input[5]; //set when input is
connected to an output
          sc_uint<1> v_reserved_output[6]; //set when output is
reserved by a input (one output more for simple coding)
          sc_uint<3> v_req[5];
          sc_uint<5> v_free;  // status of output in term of being
free
          sc_uint<4> v_id;
          sc_uint<5> v_arbit;
          sc_uint<15> v_select;
          sc_uint<2> header0, header1, header2, header3, header4;

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


          // 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]){
                                        header0 =
req0.read().range(1,0);
                              }

                              if(v_id[0] < header0[0]) v_req[0]=3; //
go to east
                              else {
                                        if(v_id[0] >
header0[0])v_req[0]=5; //go to west
                                        else{
                                                  if(v_id[1] <
header0[1])v_req[0]=4; // go to south
                                                  else{

if(v_id[1] > header0[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
                                {
                                        if
(v_reserved_output[v_req[0]])v_arbit=0; // if the requested output
was reserved, go to next input
                                }
                                if(v_arbit!=0){
                                        grant0.write(1);
                                // set grant
                                        v_select.range(2,0) =
v_req[0];
                                        v_free  = v_free &
(~v_arbit);   // inactive the related output
                                        v_connected_input[0]=1; //
input 0 is connected

v_reserved_output[v_req[0]]=1; // output is reserved
                                        if(req0.read()[5]){

v_connected_input[0]=0;v_reserved_output[v_req[0]]=0;} // if it is
tail flit, reset connection and reservation
                                }
                        }
                        if (!req1.read()[4]) //if buffer is not empty
                        {
                                if(!v_connected_input[1]){
                                        header1 =
req1.read().range(1,0);
                                }

                                if(v_id[0] < header1[0]) v_req[0]=3; //
go to east
                                else {
                                        if(v_id[0] >
header1[0])v_req[0]=5; //go to west
                                        else{
                                                if(v_id[1] <
header1[1])v_req[0]=4; // go to south
                                                else{

if(v_id[1] > header1[1])v_req[0]=2; //go to north
                                                        else
v_req[0]=1; // that is the destination
                                                }
                                        }
                                }
                                switch (v_req[1]) {
                                        case 1:   v_arbit=v_free &
1; break;
                                        case 2: v_arbit=v_free &  2;
break;
                                        case 3: v_arbit=v_free &  4;
break;
                                        case 4: v_arbit=v_free &  8;
break;
                                        case 5: v_arbit=v_free &  16;
break;
                                        default: break ;
                                }
                                if(!v_connected_input[1]) // if input
is not connected
                                {
                                        if
(v_reserved_output[v_req[1]])v_arbit=0; // if the requested output
was reserved, go to next input
                                }
                                if(v_arbit!=0){
                                // if there is any free output
                                        grant1.write(1);    // set
grant
                                        v_select.range(5,3) =
v_req[1];
                                        v_free  = v_free &
(~v_arbit);   // inactive the related outputs
                                        v_connected_input[1]=1; //
input 1 is connected

v_reserved_output[v_req[1]]=1; // output is reserved
if(req1.read()[5]){v_connected_input[1]=0;v_reserved_output[v_req[1]]
=0;} // if it is tail flit, reset connection and reservation
                                }
                        }
                        if (!req2.read()[4]) //if buffer is not empty
                        {
                                if(!v_connected_input[2]){
                                        header2 =
req2.read().range(1,0);
                                }

                                if(v_id[0] < header2[0]) v_req[0]=3; //
go to east
                                else {
                                        if(v_id[0] >
header2[0])v_req[0]=5; //go to west
                                        else{
                                                if(v_id[1] <
header2[1])v_req[0]=4; // go to south
                                                else{

if(v_id[1] > header2[1])v_req[0]=2; //go to north
                                                        else
v_req[0]=1; // that is the destination
                                                }
                                        }
                                }
                                switch (v_req[2]) {
                                        case 1:   v_arbit=v_free &
1; break;
                                        case 2: v_arbit=v_free &  2;
break;
                                        case 3: v_arbit=v_free &  4;
break;
                                        case 4: v_arbit=v_free &  8;
break;
                                        case 5: v_arbit=v_free &  16;
break;
                                        default: break ;
                                }
                                if(!v_connected_input[2]) // if input
is not connected
                                {
                                        if
(v_reserved_output[v_req[2]])v_arbit=0; // if the requested output
was reserved, go to next input
                                }
                                if(v_arbit!=0){
                                        grant2.write(1);    // set
grant
                                        v_select.range(8,6) =
v_req[2];
                                        v_free  = v_free &
(~v_arbit);   // inactive the related outputs
                                        v_connected_input[2]=1; //
input 1 is connected

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

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

                        if (!req3.read()[4]) //if buffer is not empty
                        {
                                if(!v_connected_input[3]){
                                        header3 =
req3.read().range(1,0);
                                }

                                if(v_id[0] < header3[0]) v_req[0]=3; //
go to east
                                else {
                                        if(v_id[0] >
header3[0])v_req[0]=5; //go to west
                                        else{
                                                if(v_id[1] <
header3[1])v_req[0]=4; // go to south
                                                else{

if(v_id[1] > header3[1])v_req[0]=2; //go to north
                                                        else
v_req[0]=1; // that is the destination
```

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

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

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

                if (!req4.read()[4]) //if buffer is not empty
                {
                        if(!v_connected_input[4]){
                                header4 =
req4.read().range(1,0);
                        }

                        if(v_id[0] < header4[0]) v_req[4]=3; //
go to east
                        else {
                                if(v_id[0] >
header4[0])v_req[4]=5; //go to west
                                else {
                                        if(v_id[1] <
header4[1])v_req[4]=4; // go to south
                                        else {

if(v_id[1] > header4[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);
        }
}
```

B. 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();
                clk_count++;
                if(!ach_in.read())
                        {
                        //v_packet_out.data = v_packet_out.data
+source_id.read()+ 1  ; // made a desired data
          v_packet_out.data = clk_count;
                        v_packet_out.id = source_id.read();
                        v_packet_out.dest= 1;

                                // assign destination
                        if(v_packet_out.id == 1) 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);

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

C. sink.cpp

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

        packet v_packet;
        if ( sclk.event() ){ack_out.write(false); clk_count++;}
        if (packet_in.event() )
        {
                pkt_recv++ ;
                ack_out.write(true);
                v_packet= packet_in.read();
                cout << "                        New Pkt:  " <<
(int)v_packet.data<< " is received from source: " << (int)v_packet.id
<< " by sink:  " << (int)sink_id.read() << endl;
                cout << "Flint delay: " << clk_count << "-" <<
(int)v_packet.data << "=" << clk_count-(int)v_packet.data << endl;
       sum += clk_count-(int)v_packet.data;
                if(pkt_recv%5 == 0){
```

```
                        cout << "Packet delay: " << sum <<
endl;
                        sum = 0;
                }

        }
}
```