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

COE838 – System-on-Chip Design

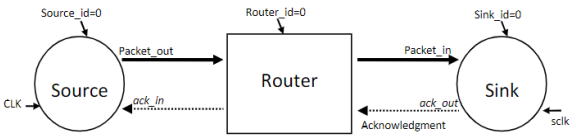
Apurva Patel - 500876938

Abstract

This project focuses on the modelling and simulation of Network-on-Chip (NoC) systems using SystemC, with focusing on interconnection using mesh architecture, 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 as a SystemC and C++ codebase, however, the final objective of this project is to develop and design a 4x4 mesh interconnection architecture. The design comprises three main components, the routers, IP cores, and their interconnections. The project progresses with exploring packet structures, including headers and payloads, essential for data transmission within the NoC. The project's core components include the source module, sink module, and router module, interconnected to facilitate packet routing and communication. 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.

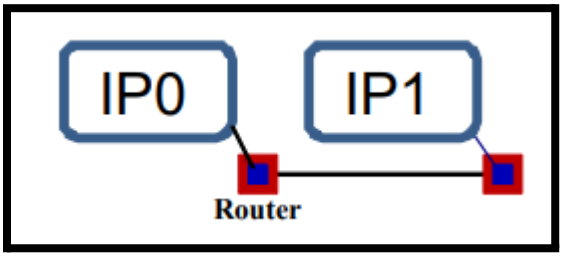
Introduction

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.

Theory

For this project, we will explore NoC architecture focused on mesh topologies, 1×2 and 4×4 configurations to be more specific. 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, as depicted in the project manual, consists of two 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.

Let us discuss the modules used in this project:

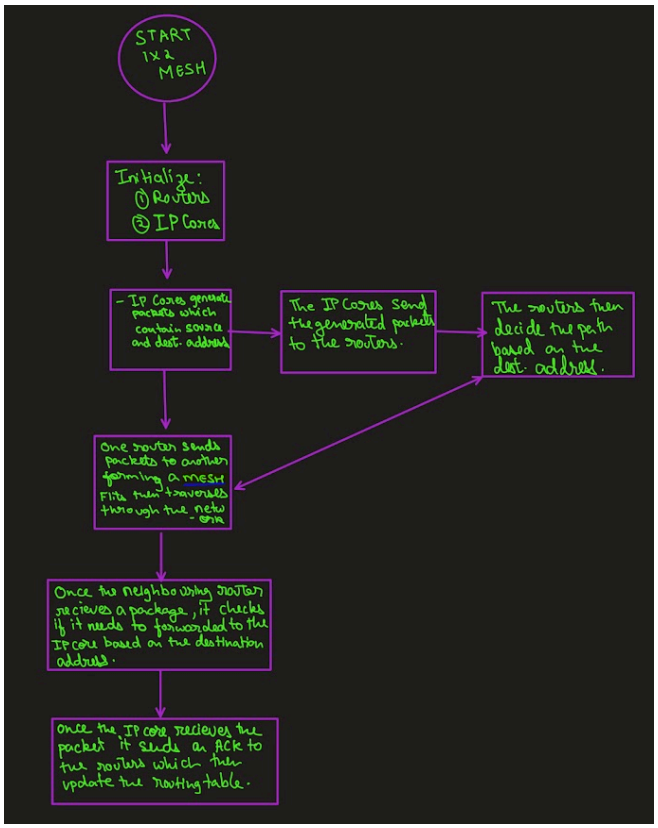
Router Module: This module manages the routing or transmission of data packets between the source and sink modules. It includes components such as arbiter, FIFO buffers and crossbar switches which are used for routing decisions and flow control during transmission of packets from one router to the other depending on the destination address and the path decided.

- Arbiter: An Arbiter in a router, is used for resource allocation, deadlock prevention and priority handling in a NoC.
- Buffer FIFO: As the name goes, these are buffers used to temporarily store data when another transaction is taking place. Manages data flow efficiently.
- Crossbar: This is a type of switch used to connect multiple inputs to multiple outputs, allowing multiple connections to take place instead of a single transaction at a time.

Source module: This module is responsible for generating new packets that consist of the source and destination address.

Sink Module: This module receives the data packets sent from the Source via routers. Once the packets are received, the Sink module sends an Acknowledgment or ACK signal across the network.

Here is a flow chart depicting a summary of the theory and introduction of this project:



Work Progress and Pending

I have split this project into two phases. The first phase involves studying and understanding the project by carefully reading through the project manual provided to us. I have gained proficient knowledge about how a 1x2 mesh topology is implemented, how it works and the use of such a topology. For phase one, the provided source code made it significantly easier for me to understand the code flow and its working.

For phase two, I have implemented the 4x4 mesh topology base network, on which I will be developing the other components needed, based on the knowledge I gained during my analysis of the 1x2 mesh network. The arbiter code required as part of the router has been designed; however, since the entire project is not ready, I was not able to compile it, but I am sure it works as they are based on the provided code. Next step is to implement buffers and crossbar switches as implemented in the 1x2 mesh topology. Along with the above implementations, I also need to design a scheme to handle the multiple requests/connections that will be handled by the network. Once the design is developed, it is time to test and simulate the SystemC code to check for any debugging scenarios and record the results thereafter.

Tentative Plan for pending work:

These are remaining components that need attention for phase two:

- FIFO Buffer
- Crossbar Switch
- Resource allocation and handling scheme for multiple requests or connections.
- Testing and debugging the entire project and recording the results.

Conclusion

Overall, the project is running a bit, behind time due to focus on wrapping up the capstone project and other course work. However, since we are provided with the 1x2 Mesh Network implementation, it should not be an issue to expand it to a 4x4 mesh network. I should be able to work on the project by this upcoming weekend and wrap it up by 26th March 2024.

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
#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
```

```
            {
                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(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[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_connected_input[4]) // if
input is not connected i.e. it is
header
            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);
    }
}
```