SystemC based NoC (Network-on-Chip) Modeling  
Course Project Interim Report

Danilo Zelenovic, 501032542

Department of Computer and Electrical Engineering

Toronto Metropolitan University

Toronto, Canada

1. Abstract

The purpose of this project is to simulate a NoC architecture with the aid of SystemC. Focus is placed on the routing specifications as well as evaluation of performance. NoC architecture is crucial to the communication between cores within a SoC design. The simulation specified for this project utilizes and implements routers, arbiters, buffers, and a crossbar switch. While the base information is provided, modifications must be made in order to record packet transmission and times of packet reception for use in analysis of the performance.

1. Introduction

NoC is a communication system between elements in SoCs. It is the connection system widely used in most parts in the modern day due to its superior performance compared to typical bus system architectures. It differs from the bus interconnects as NoC is scalable and is more efficient. As per most hardware, the efficiency is dependent on algorithms from the software portion and more. SystemC is used in the project with the goal of implementing various functions such as routers and buffers to simulate NoC packet transmission. Metrics that will be evaluated include latency, throughput, and delay.

1. Theoretical Explanation and Related Works

The theoretical background of the project requires an understanding of NoC architectures. These architectures utilize routers connected by various topologies. Each router has input and output ports in order to communicate with following routers or cores.

The arbitration mechanisms are responsible for managing shared resources and providing access. There are many various arbitration schemes that can be used. The selected scheme is done in a way that will balance overall performance and fairness of resources within the NoC.

Buffer Management is essential when working with temporary storage such as the case in FIFO buffers. They temporarily store flits while waiting or figuring out where to send the packet. Buffer utilization must be efficient in an NoC in order to properly utilize all resources and to prevent deadlock situations.

1. Work Progress

Currently, the work completed includes an understanding of the project requirements. The knowledge required to attempt the project has also been reviewed. The required questions have also been solved as required for this reporting period. The questions and solutions can be found as follows (questions are numerated):

1. Explain the architecture of source module. How the source module creates data for different sources? How a packet is made at the source module (core) level?

The source module typically represents various sources (IP cores) within the system. Each source module has an associated source which will generate packets. The data for different sources is made by the source file, which creates a packet variable and assigns it a location. The source module has 3 input ports. One for the source id, another for acknowledgements, allowing the next information to be sent, as well as clock signal for synchronization. It also contains one output port, to send out packets to the FIFO buffers in the router. The source module is sensitive to the positive clock edge. At the core level, the source module creates the packets in terms of flits. Each packet is comprised of multiple flits, 5 in this case. Each packet consists of a header and body flits. The header contains information such as destination ID, packet length as well as type. The body flits contain the actual data to be transmitted. The packets are generated by a trigger event such as a request based on a timer. Once the event occurs, the header flit is created for the packet, and all relevant information as previously mentioned is stored. The source encapsulates the header and the body flits into a packet and forwards them to the router.

2. Draw the architecture of router. (Figure 8 [1] should be amended and changed).

A diagram of a computer

Description automatically generated

Figure 1. Original Generic Router

A diagram of a computer

Description automatically generated

Figure 2. Project Router.

3. Set the clock time of source modules clk\_s equal to the router modules clk\_r and execute NoC simulation. Then explain the simulation results on the monitor in terms of receiving data by the sink module of IP1.

This addition can be located within the appendix section.

4. Add a variable in each source module and sink module and record the sending time and receiving time of flits and then output on the monitor the average packet delay in the NoC.

In the appendix, it can be seen that a variable is added to the source as well as the sink module. The goal of this portion is to determine the send and receive times of flits and then displaying the average delay of the packet. A packet is a group of flits, meaning each flit delay must be added and then averaged to determine the entire packet average delay. The changes applied to the code can be seen in the appendix section.

5. The processes in the arbiter module manage wormhole (flow control) communication in the NoC. However, each body flit should have a destination ID (similar to header flit) that is not necessary. Change the codes of arbiter in which after receiving the header flit, it does not need any information from the body flit except the tail bit (the last bit of each flit).

The changes done to accomplish this can be found in the appendix portion of the report. Wormhole communication involves having the header of the packet create a sort of virtual path for the body of the packet to follow. It is responsible for “tunneling” a path that is closed to every other flit, except for its own, allowing seamless flow for the packet. The “tunnel” is then opened for all communication once the packet sending is complete.

1. Tentative Plan for Pending Works

The plan for the pending works is as follows: Verification will be done of work up to this moment. Next, using the modifications we have created, a 4x4 mesh NoC will be designed. This can be done by altering the code provided to add additional routing options. The topology will increase in size, making it 2-dimensional. Different communication patterns will be produced from the source to the sink. The algorithms will all be implemented in the SystemC code. Schematics will be created that showcase the design chosen. Additional work will be completed in order to determine optimal implementation methods. Also to be attempted is the task of transforming the mesh topology into a torus topology, which will result in the top and bottom of the router interconnects as well as the left and right edges to be connected to one another, allowing for new paths to be taken by the data.

1. Conclusion

To conclude, the goal of the project is to simulate a NoC architecture with the aid of SystemC in order to further examine the routing mechanisms and performance analysis. Many key components will be used such as the FIFO buffer and routers and arbiters, as the purpose is to understand the ideas behind NoC design. Additionally, modifications were made to record packet transmission times and more. With this project, the aim is to further understand NoC design concepts for use in future SoC designs. With the following weeks, the NoC will be increased and potentially even have the topology changed in order to test the possibility of a torus system and determine if the benefits are great.

1. Appendix

Relevant source codes that have been modified are included in the appendix. See below.

A screenshot of a computer code

Description automatically generated

Figure 3. Crossbar.cpp

A computer code on a white background

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Figure 4. Main.cpp

A screen shot of a computer code

Description automatically generated

Figure 5. Sink.h

A white background with black text

Description automatically generated

Figure 6. Source.h