SystemC based NoC (Network-on-Chip) Modeling  
Course Project Final 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.

The provided information and code is to be altered in accordance with the manual to meet specifications such as adjusting the size of the NoC to be 4x4 instead of the provided 1x2 router size. The addition of the multiple routers would create a more complex system which would require alterations to be made to many features in order to correspond properly, such as with the sources and sinks, as well as the crossbar module and arbiter, which would thus in turn be required to have much more information readily available.

The purpose of the project is to further acquire knowledge regarding the NoC and its functionality, uses and benefits, while understanding the complexity and being able to weigh them against the positives while comparing it to a more traditional bus system.

The other important aspect of this project is to become better familiarized with SystemC, which is a library of C++ used for verification, testing and simulation purposes.

1. Past Work or Review

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. The project is completed with all functionality meeting standards, and functioning as intended.

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.

The NoC allows packets to be sent across the device is multiple methods. The current method being used for the project is the wormhole. The wormhole involves a header flit creating a path that only its body can follow, until it is completed, and the packet is finished sending, upon which the path is then sealed, allowing other packets to enter and follow it again and use that path for themselves.

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.

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. Methodology

The project was tested based on the requirements of the lab manual. The first most basic test was whether the packets were sent to routers at all. This was the first challenge, as the size of the NoC needed to be increasing, thus exponentially increasing the complexity as well. This was implemented successfully, and the packets were able to be delivered, as the results would indicate.

The next method for determining success was if the time taken for a packet was able to be tracked. This was also successfully implemented. The time was stored from the sink modules and displayed on the command window for easy access.

Note that for our project specifications, we did not test features such as packet loss, and as the bonus was not completed, the design was not altered to a torus design. However, this was potentially able to be accomplished by simply telling the routers on the edges that they can have a connection instead of it being left empty.

The sink keeps track of router times through the packet event and that is how the time is able to be displayed and tested on the terminal window. As the packet is received, on the positive clock edge, it will record time required.

The final test to determine if the project was completed was to test if the information could display if the flit was a header, body, or tail.

From the results, it can be determined that this was accomplished correctly. The header is the flit that stores information such as the source and destination, which can be read, allowing the destination to know that it was a header file. The header also contains the clock signal as well was it tail bit if it is a tail. This is able to differentiate the flits. The data flit contains a clock for synchronization purposes, the tail bit to indicate the end, as well as the actual data itself.

With the ability to recognize all of the information that is required, the project can be called successful.

1. Design

The packets were designed to have a tail, body, and header flits as per the lab manual. The traffic control file is responsible for managing where the packets are sent, with this destination address being stored in the header flit of the packet. From here, when the packet is sent from the source to the buffers, and consequently to the router, this destination information must be read in order to determine the routing. The traffic generator is connected to the inputs of all sources and must assert a signal in order to have a packet sent from a source to a destination. When the traffic generator has completed this, it will de-assert itself, thereby not allowing any additional packets to be sent.

The first step of expanding the NoC was creating multiple routers, and each router was given an ID to keep track. Next, they were mapped, with each router then knowing which router is next to it in each direction, facilitating transfers. This way, each router knows where the packet should be sent in order to reach the destination as it understands which path can be followed to reach which destination.

Once the NoC was expanded with multiple additional routers, the next step was creating proper modules. The source had the traffic generator connected, allowing the source to send packets where they were specified.

An important aspect is that the process is sensitive to multiple events, that which any of which occur, cause the packet to be routed.

The sink module is the final destination of all the packets. It has a single output, which is the acknowledgement that the packet is received. The sink has a process which is invoked any time a new packet enters through the dat in port and a clock edge is positive. When this occurs, the process ceases receiving packets and records the time and number of packets. This is then displayed as a result.

The router is responsible for routing the packets from the proper source to sink. From a buffer, which acts in FIFO methods, the router gains flits from the packet which it must send. It must first send a request with the information from the header of the packet to the arbiter, which must grant access in order for the process to occur. When granted, it is then able to being routing the packet.

The arbiter is responsible for managing whether routers are free and if permission is granted to the packets. It has request ports and grant ports, which must be activated in order to allow the packets to be sent from the routers. When a packet is sent to the router, on the negative clock edge, the arbiter will determine if the port is free or not. If it is allowed to be sent, it will read the destination address and must check if it is free and will not send if the channel is occupied.

Next, the crossbar module was altered to include more inputs and output signals as required, as the size of the NoC was increased.

The si signals were grouped as one in order to facilitate the transfers as they worked in conjunction. Using this all, the project was designed with the aid of the diagram created in Figure 2, and the code was created as such to meet the requirements as well as most basically, be functional and accurately simulate a NoC.

1. Experimental Results

As previously mentioned, the project was successfully implemented. While the bonus was not accomplished, the tasks required were able to be completed, allowing the 4x4 mesh NoC to function properly with features including keeping track of time taken for packets as well as including the routing taken as well as which flit is currently being looked at. The results can be seen in the figures provided below.

A screenshot of a computer

Description automatically generated

Figure 3. Experimental Results.

In Figure 3 above, it can be seen that the output displays all relevant information. The arbiter is noted keeping track of relevant sources. It can be seen that the program successfully is able to note if it is processing a header or body flit. The packet number is also noted, and a message is sent that the packet is successfully delivered from which source to which sink. ID values are also kept track of, as well as the time taken for the process, all of which is displayed on the terminal.

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. The additional, optional portions were not able to be completed, however this can be done at a later time for an individual challenge. The requirements of the project were attained, as the 1x2 NoC was transformed into a 4x4 mesh with additional features such as routers being able to track package times as well as make note of which flit was either a header, body, or a tail. Based on all of this information, it is safe to call the project successful. The project was very useful in gaining better understand of not only SystemC code, but also to gain greater insight into how the NoC functions, as well as the benefits of using it over other topologies that were used in the past.

1. Appendix

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

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer code on a white background

Description automatically generated

A computer screen shot of a message

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer code with black text

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated