SystemC TLM2.0 Modeling of Network-on-Chip Architecture

by

Jyothi Swaroop Arlagadda Narasimharaju

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved March 2012 by the
Graduate Supervisory Committee:

Karamvir Chatha, Chair
Aviral Shrivastava
Arunabha Sen

ARIZONA STATE UNIVERSITY

May 2012

ABSTRACT

Network-on-Chip (NoC) architectures have emerged as the solution to the on-chip communication challenges of multi-core embedded processor architectures. Design space exploration and performance evaluation of a NoC design requires fast simulation infrastructure. Simulation of register transfer level model of NoC is too slow for any meaningful design space exploration. One of the solutions to reduce the speed of simulation is to increase the level of abstraction. SystemC TLM2.0 provides the capability to model hardware design at higher levels of abstraction with trade-off of simulation speed and accuracy. In this thesis, SystemC TLM2.0 models of NoC routers are developed at three levels of abstraction namely loosely-timed, approximately-timed, and cycle accurate. Simulation speed and accuracy of these three models are evaluated by a case study of a 4x4 mesh NoC.

To my parents

ACKNOWLEDGMENTS

I would like to take this opportunity to thank Dr. Karamvir Chatha for providing me an opportunity to work under him and for providing valuable motivation, suggestions and guidance throughout my research. I would also like to thank Dr. Aviral Shrivastava and Dr. Arunabha Sen for kindly agreeing to serve on my dissertation committee.

I am thankful to my lab colleagues, Glenn Leary, Amrit Panda, Weijia Che and Haeseung Lee for helping me when I was facing difficulties in my work.

I am also very grateful to my parents. Without their encouragement it would have been impossible for me to finish this work. I am also thankful to my brother, sister and sister-in-law for all their support.

I would also like to thank my friends, here and in India, and my roommates, who have made these two years a memorable one.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

CHAPTER

1 INTRODUCTION 1

1.1 Network-on-Chip 2

1.2 Previous Work 3

1.3 Contributions of the Thesis 5

1.4 Thesis Organization 5

2 HARDWARE MODELING USING OSCI SYSTEMC TLM2.0 7

2.1 Transaction Level Modeling 9

2.1.1 Different Coding Styles of TLM2.0 10

2.1.2 OSCI TLM2.0 Standard 12

2.1.3 Blocking and Non-Blocking Transport interfaces 15

2.1.4 Basic Protocol 17

3 DESIGN OF NETWORK-ON-CHIP ARCHITECTURE IN SYSTEMC TLM2.0 19

3.1 Router Architecture 19

3.2 Functional Level Description 20

3.2.1 Routing table generation 20

3.2.2 Decoder 21

3.2.3 FIFO Implementation 22

3.2.4 Arbiter 22

3.2.5 Additional Phases 24

CHAPTER Page

3.3 Abstraction levels in modeling NoC 24

3.4 LT router model 25

3.5 AT router model 27

3.6 Cycle Accurate router model 30

3.7 Differences between LT, AT and cycle accurate NoC platform 36

4 EXPERIMENTATION, PERFORMANCE ANALYSIS AND SIMULATION RESULTS 37

4.1 Experimental Setup 37

4.2 Performance Analysis 40

4.3 Simulation Results 42

5 CONCLUSIONS AND FUTURE WORK 50

5.1 Conclusions 50

5.2 Future Work 51

REFERENCES 52

APPENDIX

A PERL SCRIPT- AUTOMATED NOC PLATFORM GENERATION 54

LIST OF TABLES

Table Page

1.1. Generic Payload Attributes 15

3.1. Differences between LT, AT and cycle accurate NoC platform 36

4.1. Speedup Table for Uniform-random traffic at injection rate = 0.1 44

4.2. Speedup Table for Hot-spot traffic at injection rate = 0.1 46

4.3. Speedup Table for Complement traffic at injection rate = 0.1 48

LIST OF FIGURES

Figure Page

1.1. Different NoC Topologies 3

2.1. Comparison of different languages 7

2.2. SystemC layered Architecture 8

2.3. Comparison of RTL and TLM models 10

2.4. Different Abstraction Levels in TLM2.0 10

2.5. TLM2.0 Architecture 13

2.6. Producer Consumer Model 14

2.7. Blocking Transport Function 16

2.8a. Forward Non-Blocking Transport Interface 17

2.8b. Backward Non-Blocking Transport Interface 17

2.9. Basic Protocol Phases 18

3.1. NoC Router implementation 19

3.2. Different levels of abstraction 24

3.3. LT NoC block diagram 25

3.4. LT NoC Model 26

3.5. AT NoC block Diagram 27

3.6. AT NoC flowchart 29

3.7. Arbitration in AT router 30

3.8. Different types of FIFO 32

3.9a. Cycle accurate NoC flow chart- Forward and Backward functions 33

3.9b. Cycle accurate NoC flow chart- Arbitration 34

3.10a. Elements in FIFO at n^{th} cycle 35

3.10b. Elements in FIFO at $(n+1)^{th}$ cycle 35

Figure Page

4.1. 4x4 Regular Mesh NoC 37

4.2. Synthetic Master 39

4.3a. Latency Comparison with Uniform-random traffic 43

4.3b. Error rate in latency with Uniform-random traffic 43

4.3c. Transaction Objects per second with Uniform-random traffic 44

4.4a. Latency Comparison with Hot-spot traffic 45

4.4b. Error rate in latency with Hot-spot traffic 45

4.4c. Transaction Objects per second with Hot-spot traffic 46

4.5a. Latency Comparison with Complement traffic 47

4.5b. Error rate in latency with Complement traffic 47

4.5c. Transaction Objects per second with Complement traffic 48

A.1. NoC platform generation 55

CHAPTER 1

INTRODUCTION

With increasing performance demands of current day applications, multi-core devices have taken an important place in semiconductor technology. One of the daunting challenges of these multi-core devices is the challenges posed by the on-chip communication. When many cores share a bus, global synchronous communication becomes challenging as the master cores compete over the control of the bus. Also there are many more signal integrity issues caused by the increased parasitic effects and cross-coupling. Network-on-Chip (NoC) is fast replacing the traditional bus architectures for on-chip communications.

The principal component of a NoC is a router and it is responsible for sending information from one point to another in a NoC. These routers can operate independently at different frequencies in a NoC. End to end chip communication can be considered to be pipelined through multiple routers along a path. There are many factors affecting the NoC performance such as network topology, flow control, routing, arbitrating algorithm and so on. It is very difficult to determine the optimum circuit structure based on the register transfer level (RTL) design flow. To solve this problem, a new level of abstraction at which designers could explore the design space much faster than RTL has emerged namely Electronic System Level (ESL) design.

ESL design is a design methodology where a system can be designed at different abstraction levels. Each abstraction level differs from the other in the amount of functional details used to describe the system. Iterative redesign becomes extremely expensive especially at RTL level when the time to market is less. When designed at a higher abstraction level, the designer has more time to

explore the design space and can come up with multiple design alternatives. In this thesis, OSCI SystemC TLM2.0 standard is used to model the NoC. The design phase starts with a higher level of abstraction and as it moves down to lower abstraction levels, different alternatives in design are explored which enables the designer to make finer grain design changes.

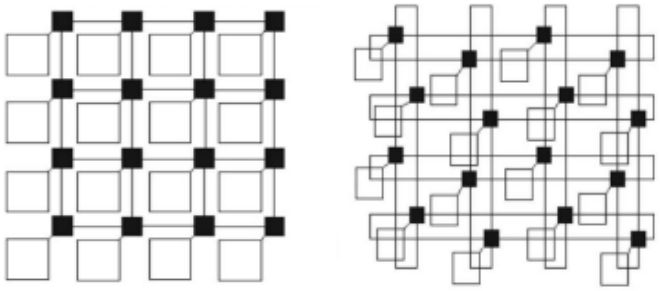
1.1 Network-on-Chip

There are two basic types of on-chip interconnections: buses and NoC. There are several factors that have led to the advent of NoC. With the continuous technology scaling of semiconductor devices, there has been an increase in performance efficiency of the cores. The interconnection between the cores is also required to deliver high communication speed to meet the performance efficiency requirements. Interconnects do not scale at the same rate as devices, and hence delay on communication channels is much larger than clock period. The power required to drive the interconnections becomes significant part of overall chip power thus cutting back the benefit from device scaling. Traditional bus architecture is not efficient since long buses increases both delay and power consumption.

Ideally the design should be completely independent of communication subsystem. The emergence of Globally Asynchronous Locally Synchronous (GALS) design methodology based multi-core devices has also raised the need for global asynchronous communication. GALS design methodology is where a set of local synchronous modules communicate with each other asynchronously. Thus, synchronous on-chip communication as assumed by several bus based architectures is no longer desirable. NoC is considered to be the solution for communication on future generation multi-core devices replacing bus based

architectures. NoC supports asynchronous packet switching based communication. Long signal propagation delays are effectively pipelined by introducing multiple routers along the path [7]. NoC supports high performance concurrent communication as various routers operate in a decentralized manner.

NoC can be classified into two broad categories based on their topologies. Regular topologies such as mesh, torus or hypercube are suitable for processor architectures aimed at general purpose computing. Irregular or custom topologies are suitable for application specific processors such as media-processors where the various cores demonstrate fairly well defined on-chip communication patterns. Irregular NoC have been demonstrated to be superior in router (resource) requirements and power consumption for application specific processors in comparison to regular architectures. Figure 1.1 shows some of the basic shapes of NoC.



Mesh Torus

Figure1.1: Different NoC topologies

1.2 Previous Work

Modeling techniques using SystemC TLM (Transaction Level Modeling) have been studied widely. Shirner et al. [14] created two TLM models for AMBA bus and compared them against synthesizable bus functional model version. More abstract TLM models were four orders of magnitude faster with error up to

45%, and the more accurate TLM reached two order of magnitude speedup with an error of 35% in worst cases. Lehtonen et al. [15] simulated different models of 2D mesh NoC such as 4x4, 6x6 and 8x8. Frequency of 50MHz was used for the NoC and simulations were run for 100ms. When simulated with a 4 word payload, TLM AT models speed up ranged from 13x to 15x when compared to RTL-VHDL models, with error on average latencies to be less than 10%. More abstract TLM LT models were 2x faster than AT models. Hu et al. [12] developed approximately-timed and cycle accurate router models and compared them against RTL router models. Simulations were performed for 4000 cycles with each initiator (two initiators were used) sending 500 transactions, with each transaction having 4 beats. TLM AT router model showed a speed up of 11.7 whereas cycle accurate model showed a speed up of 6.85 compared to RTL model. Kohler et al. [16] describes a method to estimate latencies in one process using the concept of temporal decoupling. Simulations were performed on 8x8 mesh network with uniform-random traffic pattern. Multi-hop model achieved a speedup by a factor of 20 compared to a cycle-approximate hop-by-hop TLM simulation. Estimated error depended linearly on the network utilization. They measured 45% deviation in average latencies for saturation load but on lower loads the estimated error was in acceptable range. Sgroi et al [17] address the SoC communication with a NoC approach. Here the communication is partitioned into layers following the OSI(Open System Interconnection) structure. Software reuse is promoted with an increase of abstraction from the underlying communication. Streubuhr et al. [10] proposes an efficient modeling approach that permits simulation-based performance evaluation of MPSOCs at Electronic System Level (ESL).

All the existing works model the communication architecture at two levels of abstraction, either loosely-timed and approximately-timed or approximately-timed and cycle accurate, and provide a comparison at these abstraction levels. In this thesis, a comprehensive case study with 4x4 mesh NoC at three levels of abstraction namely loosely-timed, approximately-timed and cycle accurate is presented. The loosely-timed and approximately-timed models show better speedup when compared to [12] and [15].

1.3 Contributions of the Thesis

The primary contributions of the thesis are,

- Design of a functional level parameterizable NoC router architecture.
- Implementation of NoC router model at three abstraction levels- Loosely-timed, Approximately-timed and Cycle Accurate.
- Implementation of 4x4 mesh NoC using above mentioned router models for performance analysis and simulation purposes.
- Implementation of framework for performance data collection.
- Comparison of Speed vs Accuracy for NoC models at various abstraction levels.
- Perl Script for automated generation of NoC platform using a set of files describing the characteristics of NoC.

1.4 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 describes design methodologies using SystemC TLM2.0 standards. Design and architecture of NoC router in SystemC TLM2.0 is explained in Chapter 3. It also describes the design of NoC router models in three abstraction levels of loosely-timed,

approximately-timed and cycle accurate. Chapter 4 describes the implementation of 4x4 2D mesh topology NoC for experimentation and performance analysis. It also describes the simulation results comparing the NoC platform at different abstraction levels. Lastly, the concluding remarks and future work is described in Chapter 5.

CHAPTER 2

HARDWARE MODELING USING OSCI SYSTEMC TLM2.0

SystemC is a system design language that has evolved in response to a need for a language that improves overall productivity for designers of electronic systems. SystemC offers real productivity gains by letting engineers design both the hardware and the software components together as they would exist on the final system, but at a higher level of abstraction. [4] This means that it is possible to concentrate on the actual functionality of the system rather than on its implementation details. Moreover, since the detailed implementation is not finalized, it is still possible to perform consistent changes to the system, enabling an effective evaluation of different architectural alternatives (including the partitioning of the functionalities between hardware and software).

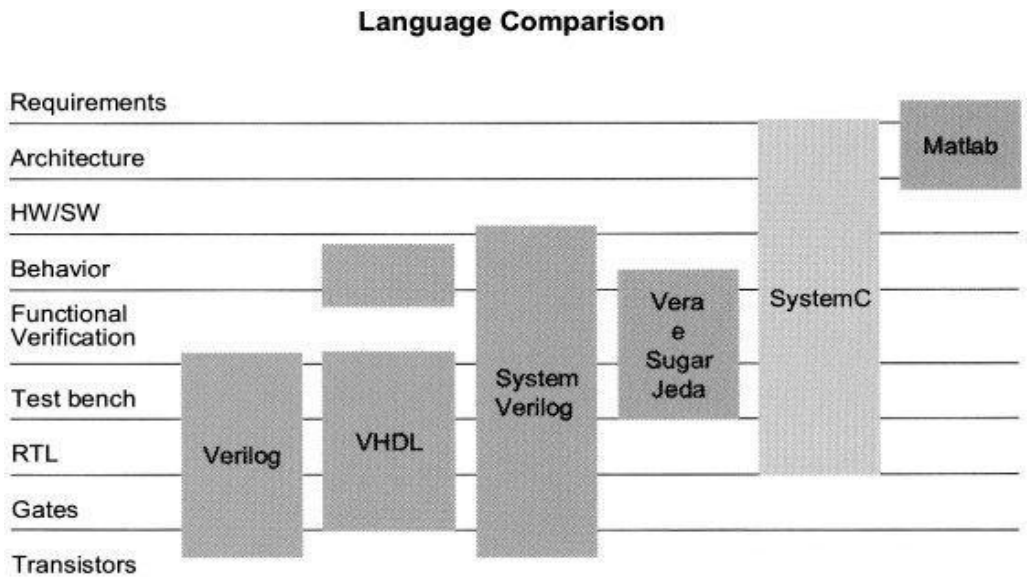


Figure2.1: Comparison of different languages [4]

Figure 2.1 shows a comparison among SystemC and other Hardware Description Languages (HDLs). Although SystemC supports modeling at the register transfer

level (RTL), it is more often used for the description at higher abstraction levels. SystemC is characterized by its higher simulation speed than HDLs; note that this high simulation speed is not only due to the SystemC language itself, but it is mainly caused by the high level system descriptions enabled by the use of SystemC. SystemC is C++ class library which focuses on system level design and verification. It defines a customizable base model of computation with a generalized model for communication through channels and synchronization based on events. SystemC provides different time models such as Untimed, Untimed with discrete ordered events and timed with discrete ordered events. The models of SystemC communicate with each other through channels which can be either SystemC standard channels or customized ones. The following Figure 2.2 shows the architecture of SystemC language.

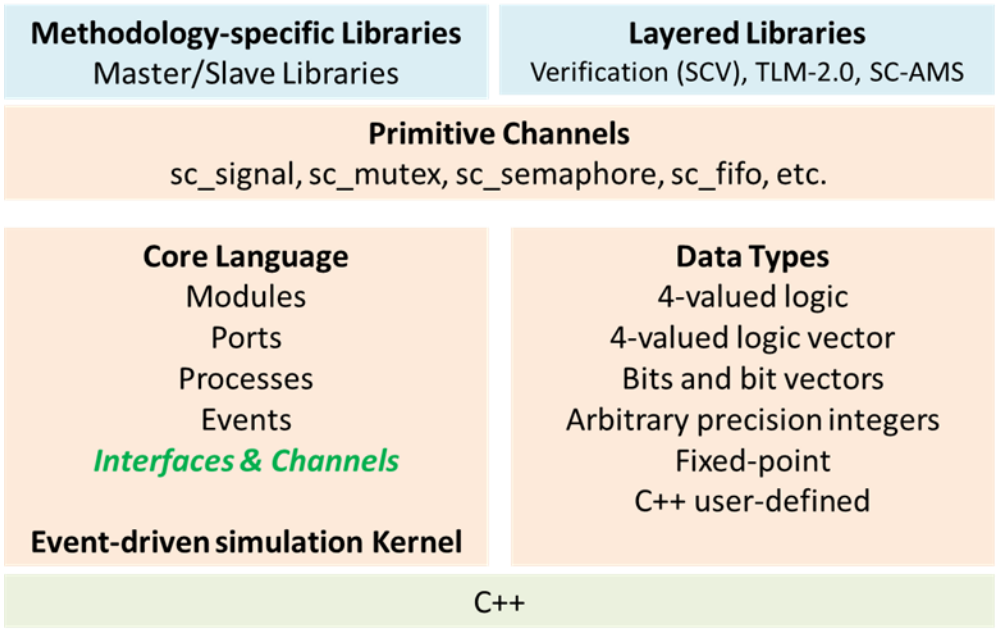
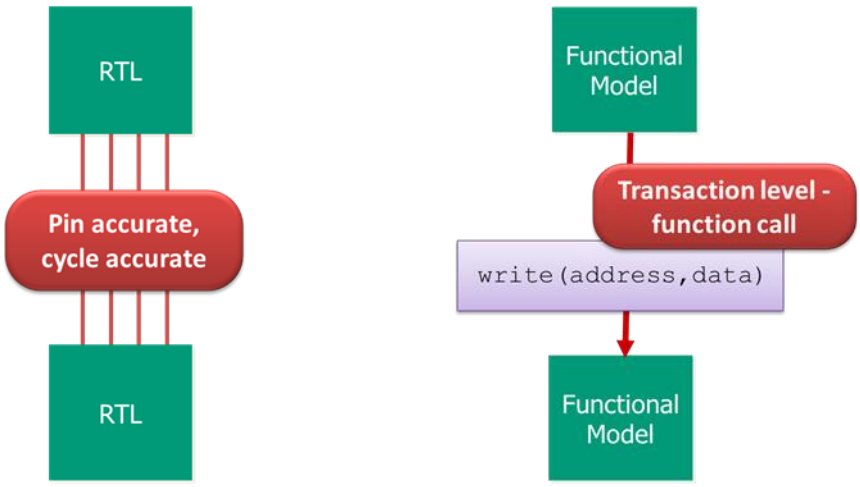


Figure2.2: SystemC layered Architecture [4]

The basic layer of SystemC provides an event-driven simulation kernel. This kernel works with events and processes in an abstract manner, coordinating events and switching between processes, thereby allowing SystemC to simulate the implicitly parallel hardware features. Modules and processes describe the abstraction of structural information, while interfaces and channels represent the abstraction for communications. Data is transferred between modules through interfaces and channels. Since SystemC is implemented on top of C++, all the C++ features can be used to speed up modeling and increase code reusability. [4]

2.1 Transaction level Modeling (TLM)

TLMs are higher level abstraction models compared to RTL models. They separate the communication details from the implementation details of the functional units. In TLM, communication occurs through function calls and more emphasis is given to functionality of transfer than the actual implementation of communication protocol. TLM models consist only of details needed in the earlier stages of design development. By not including pin accurate details like the RTL models, much higher simulation speeds are achieved compared to RTL models. The input and output signals involved are abstracted into transaction objects, which will be discussed in detail later. TLM models can be set up much faster, as they are much simpler when compared to RTL models and also TLM models run much faster than RTL models. TLM models can be used for design as well as functional verification. The following Figure 2.3 illustrates the difference between RTL and TLM models.



Register Transfer Level Model Transaction Level Model

Figure2.3: Comparison of RTL and TLM models [2]

2.1.1 Different Coding Styles of TLM

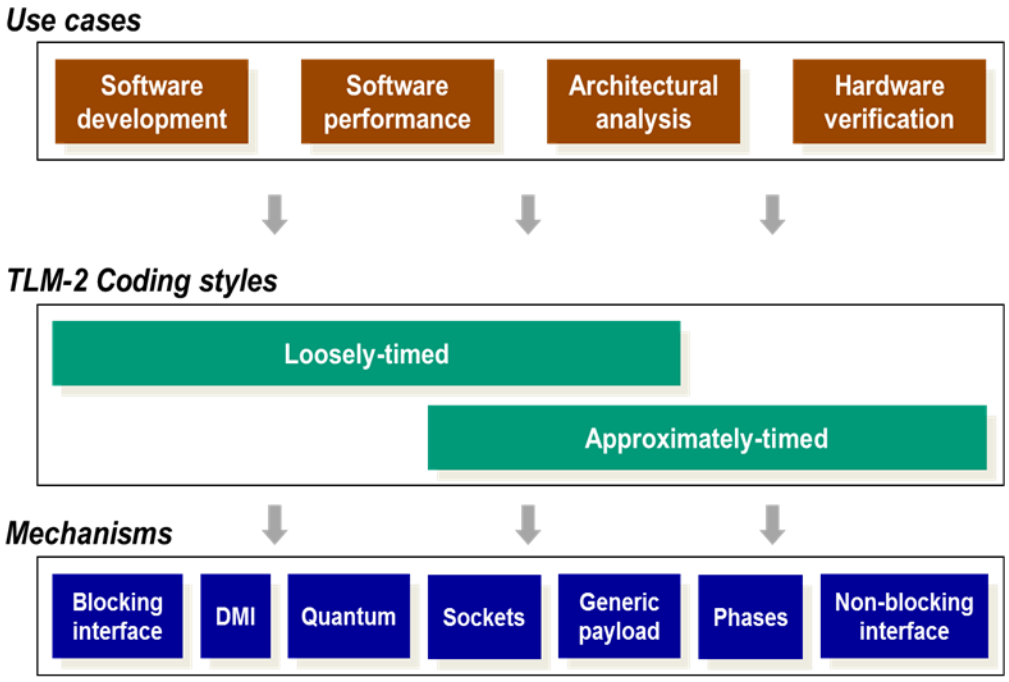


Figure2.4: Different abstraction levels in TLM2.0 [2]

TLM data transfers are modeled as transactions through function calls. These abstraction levels are distinguished by the timing accuracy in which

communication takes place. Design models can be divided into three different categories (or abstraction levels) according to the timing model style: loosely-timed, approximately-timed, and cycle-accurate. Figure 2.4 shows the applications of TLMs with different coding styles. It is clear that for hardware applications, approximately-timed (AT) style is preferred and loosely timed (LT) models are used for software development.

There are two basic coding styles of TLM specified by IEEE standard:

Loosely-timed: The loosely timed models contain less timing details when compared to approximately-timed models and cycle-accurate models. The exact communication details does not greatly influence design decisions at its initial phases, so it can be safely ignored. In loosely-timed models, every communication transfer whether it read transfer from memory or a write transfer to memory can be modeled as a single transaction. The communication in loosely timed models can thus be considered to have exactly two timing points: Begin and End. The loosely-timed coding style is appropriate for software development in an MPSoC environment. This coding style supports modeling of timers and coarse-grained process scheduling, sufficient to boot and run an operating system. The most important aspect of this abstraction level is temporal decoupling, where processes can run ahead of simulation time. This means that the different SystemC models of the architecture do not synchronize with each other at every clock cycle. With Loosely-Timed interfaces, the synchronization mechanisms among the components of a system introduce a continuous trade-off between the amount of temporal decoupling and the simulation speed. It does not make much sense to require an accuracy of 100% at the interface of models

described at this modeling style since, anyway, the timing accuracy of the whole system will be compromised by the temporal decoupling.

Approximately-timed: This coding style has more timing points in a transaction. At this level the number of bus cycles is important: the information that the bus transfers for each clock cycle is grouped in one transaction; this coding style is appropriate for the use case of architectural exploration and performance analysis. At this level a transaction is broken down into multiple phases (corresponding to bus transfer phases), with an explicit synchronization point marking the transition between phases. This coding style does not use temporal decoupling. The processes in this level of abstraction run in lock-step with simulation time. Despite its name, this coding style can accurately model the timing of the communication. This abstraction level does not mean that the model will be described at an RTL level, only that the timing obtained at the interface is correct.

Cycle-accurate style is not listed (in Figure 2.4), since there is no standard for this kind of coding style. In this thesis, the model is built based on approximately-timed style and cycle-accurate features are added. The cycle-accurate model captures the behavior in each clock cycle. There is no need to predict the delay before sending a transaction since a cycle-accurate, clock-triggered module could calculate the delay itself. This is useful when simulating a complex system. However, this kind of model needs more work on modeling and runs slower than the loosely-timed and approximately-timed models.

2.1.2 OSCI TLM2.0 Standard

OSCI (Open SystemC Initiative) have released a new TLM standard in June 2008. It provides a standardized approach for creating models and transaction-

level simulations. Figure 2.5 shows the architecture of TLM-2.0. One contribution of TLM-2.0 is the standard transaction type (generic payload) and related interfaces and socket. To maintain interoperability, TLM-2.0 defines a unified communication mechanism that uses core interfaces, sockets and a basic protocol. An important advantage of TLM2.0 is Interoperability. To maintain interoperability, TLM-2.0 defines a unified communication mechanism that uses sockets, standard transaction type (generic payload), a basic protocol and core interfaces.

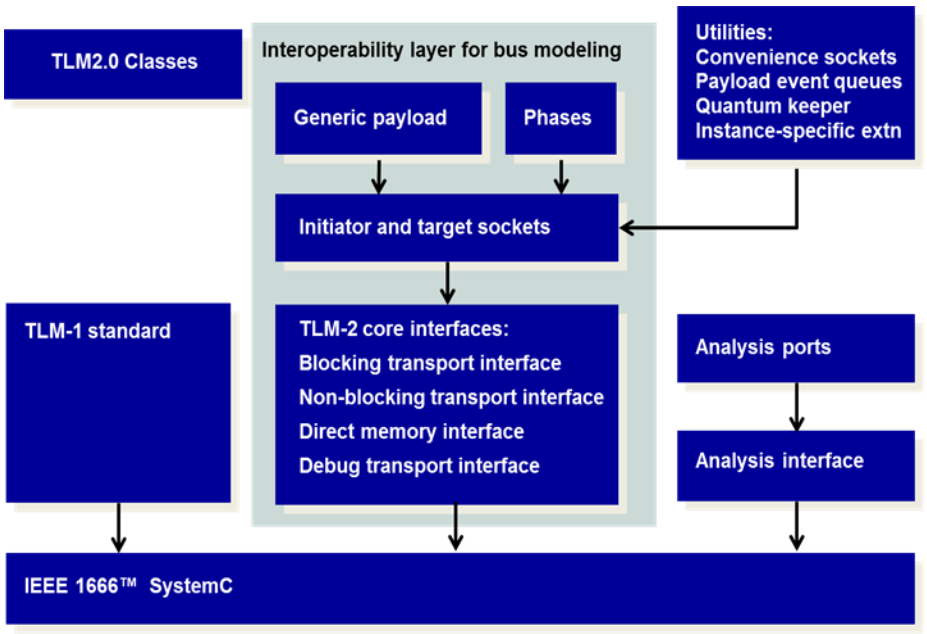


Figure2.5: TLM2.0 Architecture [2]

In TLM-2.0, an initiator is a module that initiates new transactions, and a target is a module that responds to transactions initiated by other modules. A transaction is a data structure (a C++ object) passed between initiators and targets using function calls. The same module can act both as an initiator and as a target, and this would typically be the case for a model of an arbiter, a router, or a

bus. In order to pass transactions between initiators and targets, TLM2.0 uses sockets. An initiator sends transactions out through an initiator socket, and a target receives incoming transactions through a target socket. A module that merely forwards transactions without modifying their content is known as an interconnect component. An interconnect component has both target socket and initiator socket. Figure 2.6 shows the producer consumer model, where the producer is the initiator and consumer is the target. The transaction object shown as a square feature inside initiator block can be sent from initiator to target through interconnect by forward path and the target can respond back by sending the transaction object through backward path.

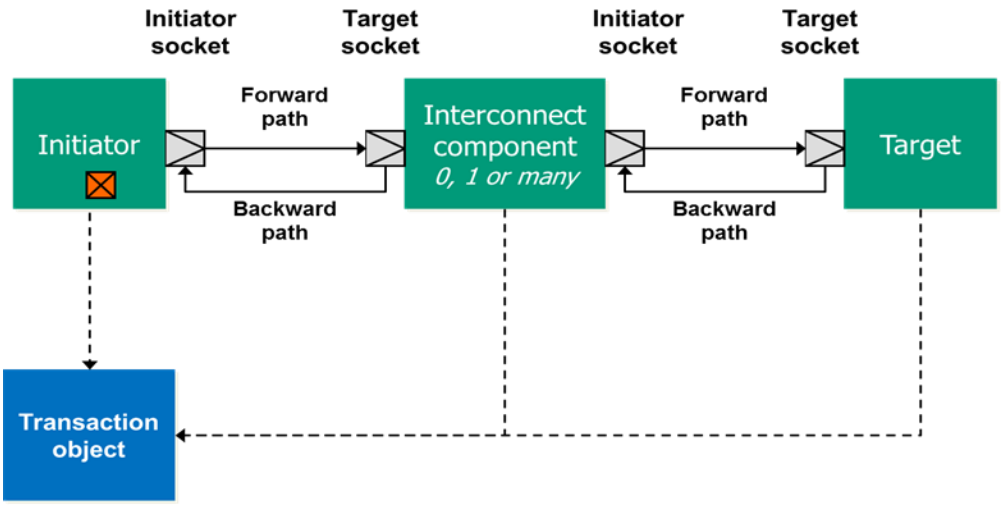


Figure2.6: Producer Consumer model [2]

The generic payload serves two closely-related purposes. It can be used as a general-purpose transaction type for abstract memory-mapped bus modeling when you are not concerned with the exact details of any particular bus protocol, offering immediate interoperability between models off-the-shelf. Alternatively, the generic payload can be used as the basis for modeling a wide range of specific

protocols at a more detailed level. The beauty of this approach being that it is relatively easy to bridge between different protocols when both are built on top of the same generic payload type [15]. Table 2.1 shows the attributes that are generally associated with a generic payload.

Table2.1: Generic Payload attributes

Attributes	**Descriptions**
Command	read or write type of the transaction
Address	read or write address
Data pointer	the pointer pointed to the data array
Data array	A data array, each member is one byte data.
Data length	Number of bytes of the data in a transaction
Byte enable array	Identify which byte lanes are used in data array
Streaming width	Number of bytes transferred on each beat in a transaction
Response status	Status for the response transaction
Extension pointer	Pointer to an user defined extension class

2.1.3 Blocking and Non-Blocking transport interfaces

Blocking and Non-Blocking interfaces are the two basic interfaces of TLM2.0 transport interfaces. The blocking interface uses blocking transport function for communication. This function is called by the initiator thread, received by the target thread, which processes the request and then returns the result. Until the

transaction has been processed and released the initiator thread is blocked. The blocking transport functions and its arguments are shown in below figure.

Figure2.7: Blocking transport function

The principal argument of a blocking transport function is the transaction object handle which is the pointer to the data structure that has different attributes as mentioned above while explaining generic payload. The b_transport call also carries a timing annotation represented by sc_time argument in the Figure2.7, which should be added to the current simulation time to determine the time at which the transaction is to be processed. The timing annotation is active on both the call to and the return from the b_transport method. This kind of interface is usually used in loosely-timed coding style.

The non-blocking transport functions are called by the initiator thread, received by the target thread, which immediately returns, before processing the request. Subsequently the target, having processed the request makes a transport call backwards to the initiator to return the result. In the non-blocking case there are actually two types of transport used. The forwards transport path is used by the initiator to pass the request to the target and the backward transport path is used by the target to return the response. The advantage of the non-blocking transport interface is that the initiator can carry on processing, while the target is processing the request originally made.

Figure2.8a: Forward non-blocking transport interface

Figure2.8b: Backward non-blocking transport interface

These functions have timing annotation as well as phase as arguments along with the transaction object handle. The timing annotation has the same significance as explained above in blocking transport interfaces. The phase can take any value mentioned below in the basic protocol phases.

2.1.4 Basic Protocol

TLM-2.0 defines basic transaction phases to maintain a basic communication protocol. The basic protocol is accurate enough for simple transactions. Users can extend the payload with extra attributes and define new phases to implement a certain protocol. The four important phases of a base protocol are:

BEGIN_REQ (Begin Request)

- Initiator acquires bus
- Connections becomes “busy” and blocks further requests
- Payload becomes “busy”

END_REQ (End Request)

- Target “accepts” request and completes the handshake
- Bus freed to start additional requests

BEGIN_RESP (Begin Response)

- Target acquires bus to provide a response
- Bus becomes “busy”

END_RESP (End Response)

- Initiator acknowledges response to complete it.
- Bus and Payload reference freed up

The following Figure 2.9 shows the phases involved in a base protocol.

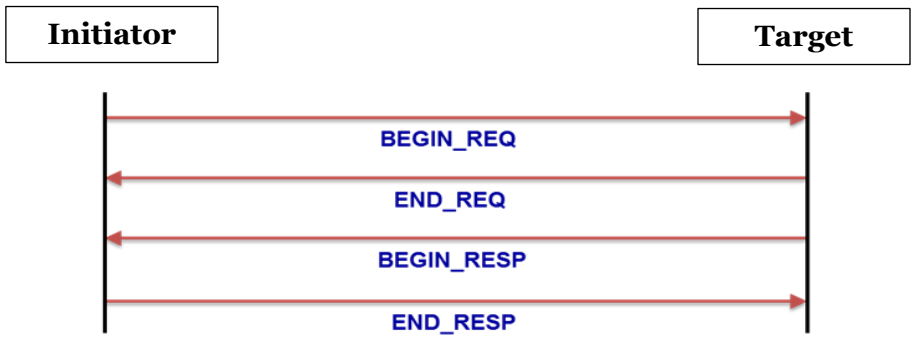


Figure 2.9: Basic Protocol Phases

The return types used for non-blocking function calls are:

TLM_ACCEPTED

- Transaction, phase and timing arguments unmodified (ignored) on return
- Target may respond later (depending on protocol)

TLM_UPDATED

- Transaction, phase and timing arguments updated (used) on return
- Target has advanced the protocol state machine to the next state

TLM_COMPLETED

- Transaction, phase and timing arguments updated (used) on return
- Target has advanced the protocol state machine straight to the final phase

CHAPTER 3

DESIGN OF NETWORK ON CHIP ARCHITECTURE IN SYSTEMC TLM2.0

3.1 Router Architecture

The most important component of a NoC interconnect fabric is a router. The router is responsible for transmitting the packets from one point in the network to another. Multiple routers are connected together in a NoC interconnect fabric. The router may have a variable number of input and output ports. A packet arriving at an input port will be forwarded to one of the output ports. A destination address in the packet header and a routing table will be used to make the output port selection.

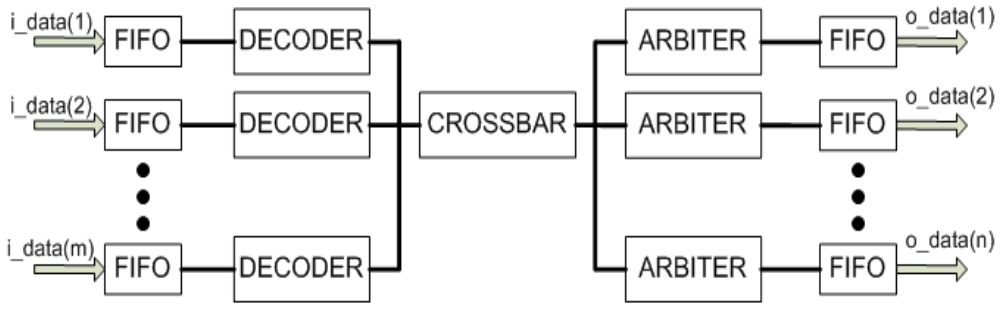


Figure3.1: NOC Router Implementation

The basic components of NoC router are following:

- FIFOs
- Decoder
- Crossbar
- Arbiter

In a NoC, when a packet is being transmitted, few control signals are also used to control point to point transmission on the on chip network. Different functions used to model NoC router are explained in detail in Section 3.2.

3.2 Functional level description

There are various functions used in modeling router at different abstraction level. Different models use few or all of these functions. Detailed description of these functions is explained in this section.

3.2.1 Routing table generation

The router uses a router-table to determine which output socket to send an incoming transaction object. The router-tables are specified to the router at compile-time via routing table file where a table is specified for each router. Each entry in the table contains a 32-bit start address, a 32-bit end address and the output socket number (0, 1, 2, etc). Each entry specifies a range of output addresses to be routed to a specific output socket. As an example, a destination address in the range is 0x00000000 – 0x000000FF should be routed to output socket #2. The corresponding entry in the routing table would be (16#00000000, 16#000000FF, 2). The example below shows three entries in a routing table. The first entry is explained in the example above and the second entry is for address range of 0x00000100-0x000001FF routed to output scoket #1. The third is for a destination address of 0x00000200 routed to output socket #3.

((16#00000000, 16#000000FF, 2),
(16#00000100, 16#000001FF, 1),
(16#00000200, 16#00000200, 3))

Example: Routing table

While the router has output sockets numbered #0, #1, #2, #3, etc., when the address attribute in the transaction object does not matches to a Router Table entry, the transaction object will be dropped. It will be processed normally, but will never leave the router. The router creates the routing table during compile time by reading the router table specific to it (distinguished from other router tables by the router ID), by opening and reading the file named "routing_tables_pkg.vhd". A C++ vector type is declared to store the routing table entries. Each entry in the vector is a struct composed of three elements:

- 32 bit start address
- 32 bit end address
- Output socket (natural) number

3.2.2 Decoder

The decoder uses the X-Y based routing to decide the route of the transaction object. The decoder uses the routing table to determine which output socket the transaction object is to be sent to, for the next level of router. When a transaction objects enters a router, the first function which handles it is the decoder function. The address attribute of the transaction object is used to compare against each entry in the routing table of the corresponding router to check for a match. If a match occurs the corresponding output socket number is returned. If the address is not found in any of the entries of the routing table, a negative one (-1) is returned, which means that the transaction object is illegal or invalid. In such a situation, the transaction object doesn't leave the router and is deleted. Once the output socket number is decided, transaction object is sent to the next router in case of loosely timed. But in case of approximately timed and router cycle

accurate router model, once the address is decoded, it is sent into the corresponding output FIFO associated with the particular output socket.

3.2.3 FIFO implementation

The FIFOs are used in approximately-timed and cycle accurate router models. These are implemented using C++ vector. Each element in the FIFO has two components:

- Socket ID through which transaction object entered the router
- Transaction object handle

Value which stores the socket ID is searched against each entry and when a match occurs, it returns the corresponding transaction handle, which will be used by the arbiter for a particular arbitration scheme. The depth of FIFO is decided by the parameter FIFO_DEPTH which is declared as a macro in the router.

3.2.4 Arbiter

The arbiter is located before every output socket of the router. The basic function of the arbiter is that, it takes in multiple requests and generates grant to a particular request. The arbiter utilizes round robin mechanism arbitration. A fair priority arbiter is made by changing the priority from cycle to cycle. This is used in approximately-timed and cycle accurate models of the router. The arbitration takes place in both the directions i.e output FIFO for processing the requests and input FIFO for processing the responses. The implementation of round robin arbitration is discussed below. The FIFOs contain two elements, the socket ID through which transaction object entered the router and the transaction object handle. Two arrays of integer, with size equal to the number of input and output ports has each element storing the socket ID which has the highest priority in

that clock cycle for the particular input/output port. If the FIFO is empty, arbitration is not performed for that clock cycle and the priority changes by incrementing the ID of the particular socket. When the FIFO is not empty, the FIFO is searched for the ID stored by the priority array. If a match occurs, the corresponding transaction object is selected to be sent through socket. If the FIFO is not empty and match does not occur, the priority ID is incremented until a match occurs and the corresponding transaction object is sent through the socket. This pseudo code for the arbitration scheme used at the FIFOs corresponding to output socket for one clock cycle is shown below.

```
for i=0:Num_outputs-1 do
   if (FIFO_i = empty) then
      current_m(i) = current_m(i) + 1
   elseif (FIFO_i = !empty) then
      for k=0:FIFO_i.size()-1 do
         if (current_m(i) in FIFO_i(k)) then
            nb_transport_fw (corresponding transaction object)
            current_m(i) = current_m(i) + 1
         elseif (current_m(i) is not found in FIFO_i(k)) then
            current_m(i) = current_m(i) + 1 until match occurs
            nb_transport_fw (corresponding transaction object)
         end if
      end for
   end if
end for
```

3.2.5 Additional phases to the base protocol

To make router model cycle and register accurate, two more phases are added to the base protocol. The additional phases are FIFO_FULL and FIFO_AVAILABLE. When a transaction object with BEGIN_REQ phase enters a router, the size of FIFO is checked, if the size is equal to FIFO_DEPTH, the phase FIFO_FULL is sent to all the adjacent routers. If a FIFO_FULL is received by a router, no more transactions are sent through that particular socket until it receives the FIFO_AVAILABLE phase. The router checks for the depth of the FIFO every clock cycle and once it reduces to FIFO_DEPTH- 1, it sends the phase FIFO_AVAILABLE to all the routers to which FIFO_FULL was sent i.e all the adjacent routers.

3.3 Abstraction levels in modeling NoC

Abstraction is a powerful technique for design and implementation of complex System-on-Chips. It allows the designer to tackle complex systems by hiding the low level implementation details. Different amounts of details are visible at different levels of abstraction.

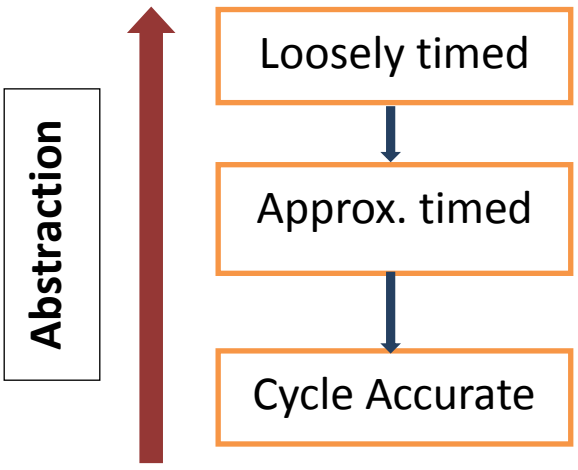


Figure3.2: Different levels of abstraction

In this thesis, three electronic system levels of abstractions are considered for modeling the NoC in SystemC TLM2.0.

- Loosely-timed or LT modeling level
- Approximately-timed or AT modeling level
- Cycle Accurate modeling level

In general, TLMs pose a trade-off between an improvement in simulation speed and a loss in accuracy. The tradeoff essentially allows models at different degrees of accuracy and speed. High simulation speed is traded in for low accuracy, and a high degree of accuracy comes at the price of low speed. [14]

3.4 LT router model

Loosely-timed model uses blocking transport function (explained in Section 2.1.3) for communication. Communication in loosely-timed models allows for exactly two timing points associated with each transaction, call and return of the blocking transport function, respectively. In loosely-timed communication, the transaction is processed within the context of the initiator of the transaction solely; context switches due to multi-hop communication are avoided. In Figure 3.3, this is indicated by a solid arrow for each transaction.

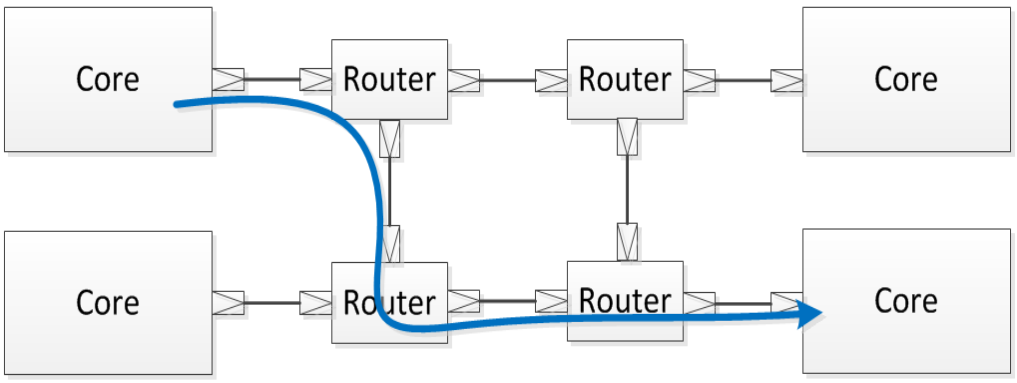


Figure3.3: LT NoC block diagram

The blocking transport function, as the name suggests, blocks other transaction until the current function call is completed. Since all transactions take place sequentially, no congestion takes place in any of the router. Hence there is no need of an arbiter in loosely-timed router model. The LT router model is kept untimed as the interest is in faster simulation at this level of abstraction. The important components of LT router model are:

- Routing table
- Decoder

The detailed functional descriptions of these components are given in Section 2.2.1 and 2.2.2. In case of the loosely timed NoC model, b_transport function in a router decodes the incoming transaction object's address attribute and once the output socket number is decided, it calls the b_transport function in the next router to which transaction object is transmitted to. This serial effect is shown below in Figure 3.4.

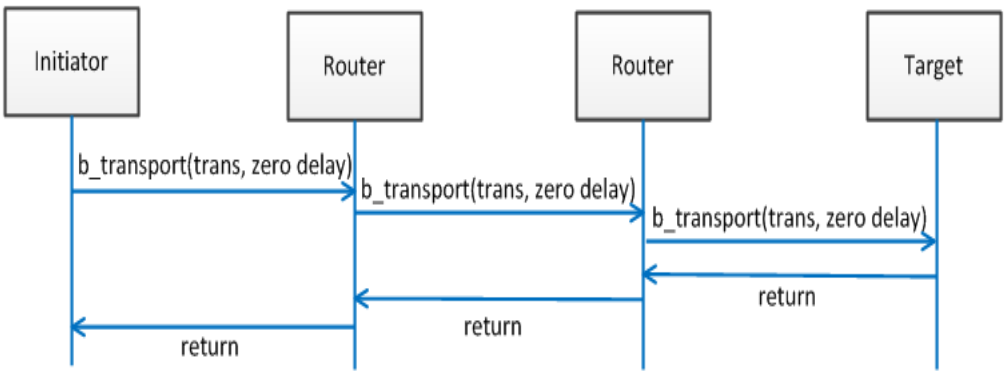


Figure3.4: LT NoC model

The return types used for blocking function calls are:

- TLM_OK_RESPONSE- This is returned when the transaction is complete without any error.
- TLM_INCOMPLETE_RESPONSE- This type is returned when the transaction is yet to be completed.
- TLM_ERROR_RESPONSE- This is returned when the operation of a transaction fails. It can be error due to address or the operation meant to be performed.

3.5 AT router model

For more timing accurate simulation of multi-hop communication, several timing points are needed per transaction. Approximately-timed communication provides more accuracy, and is implemented using non-blocking transport functions. In Figure 3.5, this is indicated by discontinuous lines for a single transaction. This level is needed to simulate congestion on buses, and to experiment with different arbitration strategies. Approximately-timed model uses nb_transport_fw and nb_transport_bw functions (explained in Section 2.1.3) for communication.

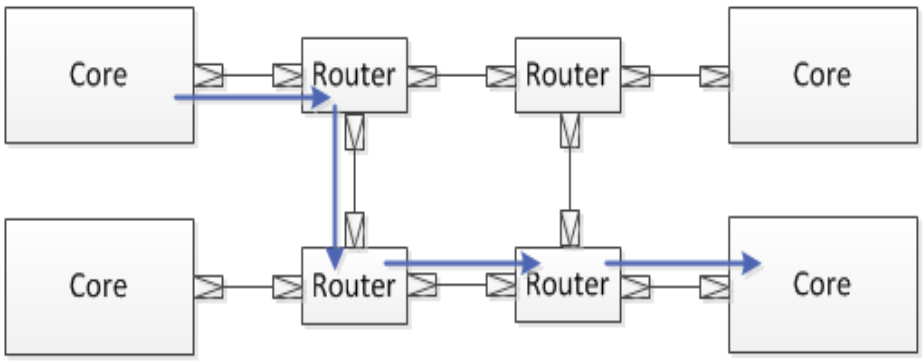


Figure3.5: AT NoC block diagram

At this level of abstraction, while modeling the NoC router, round robin arbitration scheme is used. This is mainly performed to manage congestion when multiple transactions are started simultaneously.

The important components of AT router model are:

- Routing table
- Decoder
- Arbiter
- FIFOs

The transaction object in this abstraction level is first processed by decoder function. Once decoded and output port is selected, it is sent to the output FIFO located at the corresponding output socket. The FIFO consists of mainly two elements, socket ID through which transaction object entered the router and the transaction object handle. The arbiter performs variable priority round robin arbitration based on the IDs stored in the FIFO. Once a match occurs the corresponding transaction object is sent to next router. For faster simulation purposes, depth of the FIFO is not fixed and all the transactions in the FIFO are processed and the FIFO is emptied every clock cycle.

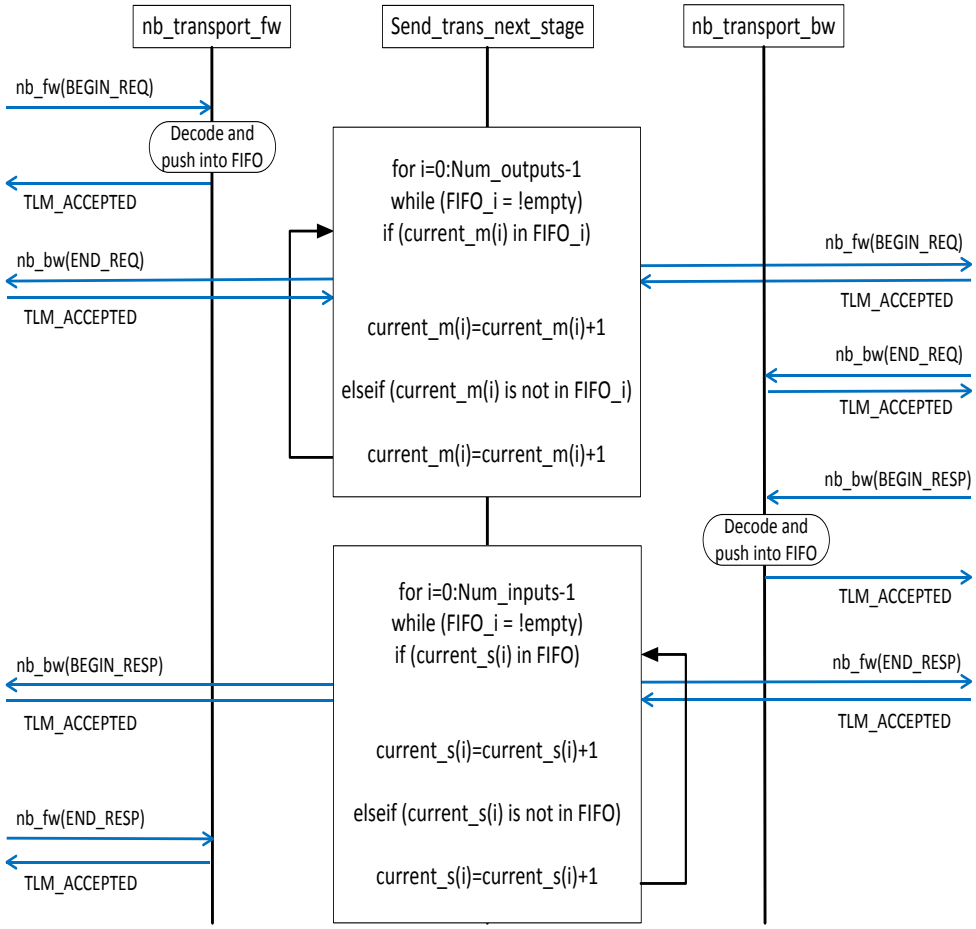


Figure3.6: AT NoC flow chart

Figure 3.6 shows the flowchart while processing a transaction in AT router. During a clock cycle when multiple transactions enters the FIFO, all the transaction are sent through output socket during that clock cycle but the sequence of the transactions being sent follows round robin arbitration policy. The priority is incremented every clock cycle. Since multiple transactions are being sent every clock cycle, latency has to be characterized for each transaction. The latency in approximately timed model is characterized by additional attributes (start time and end time) in addition to the generic payload of a transaction. When a transaction is sent into the NoC, both these attributes record the value of current simulation time. In the router, the i^{th} transaction sent out a

FIFO adds a delay of i times the clock period to the end time attribute of transaction. The difference between the start time and end time attributes of a transaction object are used to calculate the latency of each transaction.

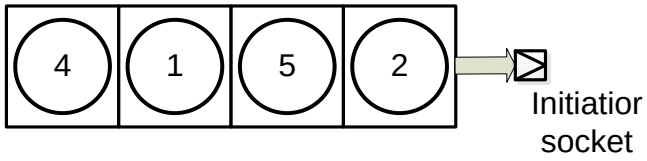


Figure 3.7: Arbitration in AT router

For example say at a particular clock cycle, transactions inside a FIFO are as shown in the above Figure 3.7. The circular features represent the transaction object and the number within it represents the socket ID through which the transaction entered the router. If the current priority for the particular output socket is 2, transaction with ID-2 is sent first through the socket and a delay of one clock cycle is added to the end time attribute and the next transaction that is selected is with ID-4 and a delay of two clock cycles is added. This process continues till the last transaction, which in above example is transaction with ID-1 is sent with a delay of 4 clock cycles.

3.6 Cycle Accurate model

The cycle accurate model differs from approximately timed router model in the following ways:

- Only one transaction object is sent through an output socket every clock cycle
- The model is both cycle accurate and register accurate
- Additional phases to introduce back-pressure

The important components of cycle accurate router model are:

- Routing table
- Decoder
- Arbiter
- FIFOs
- Additional phases: FIFO_FULL, FIFO_AVAILABLE

The operation of decode followed by sending the transaction into the FIFO at the output socket is the same as mentioned in the approximately timed router model. In this model, variable priority round robin arbitration mechanism selects only one transaction object from every FIFO to be sent to the next router or target. Here the depth of the FIFO is limited by the generic FIFO_DEPTH declared as a macro in the cycle accurate router code. The additional phases FIFO_FULL and FIFO_AVAILABLE are used to introduce back-pressure in order to avoid deadlock occurrence as the FIFO depth is limited. Once the FIFO depth increases to its full capacity, FIFO_FULL phase is sent to all adjacent routers since a transaction coming from any of the adjacent router can affect the operation of the FIFO. Once the depth reduces by FIFO_DEPTH-1, FIFO_AVAILABLE phase is sent to all adjacent routers. In this thesis, for analysis and simulation purposes, FIFO_DEPTH is 8.

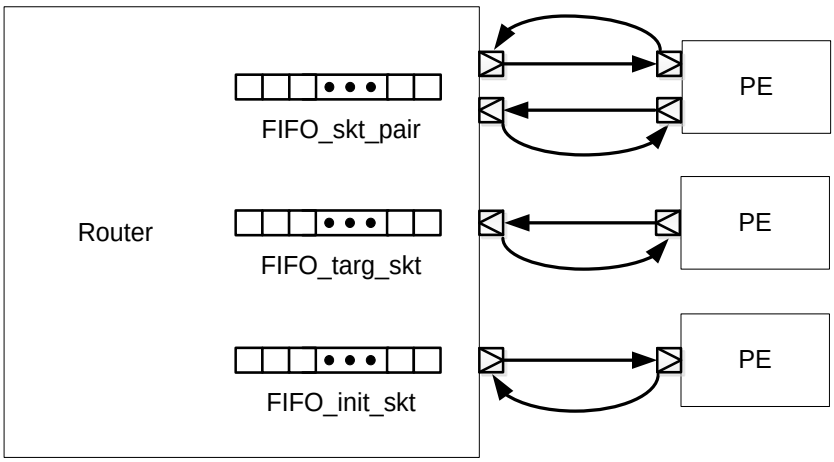


Figure 3.8: Different types of FIFOs

In cases where two adjacent routers or router-processing element are connected by a pair of sockets in either direction, interaction between forward and backward paths in the same direction needs to be considered. In order to maintain cycle accuracy with transactions going through forward and backward paths in same direction so as to allow only one transaction, either a request or a response per clock cycle, three types of FIFOs are used

- FIFO_skt_pair- This types of FIFO is used when there is a pair of initiator and target sockets connected between two routers or between router and processing element. It can store either request or response transactions.
- FIFO_init_skt- This is used only when an initiator socket is connected to a router or processing element. It stores only request type transactions.
- FIFO_targ_skt- This is used only when a target socket is connected to a router or processing element. It stores only response type transactions.

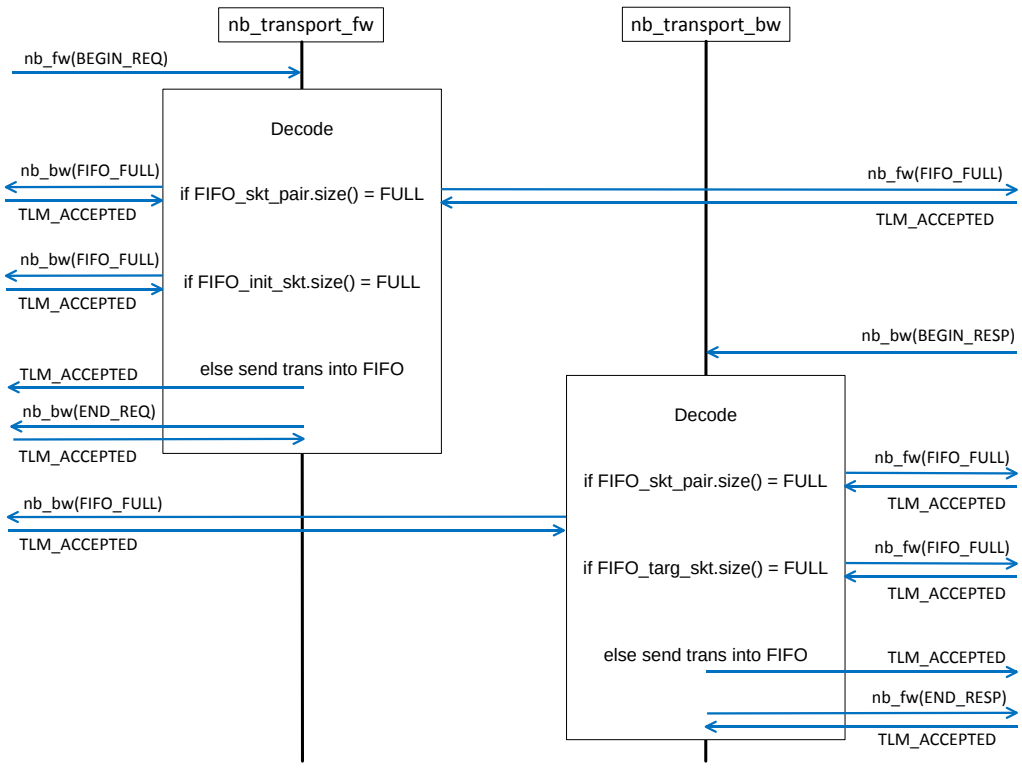


Figure 3.9a: Cycle accurate NoC flow chart- Forward and Backward functions

Once the address is decoded, the type of FIFO is known. If it is of the type FIFO_skt_pair and if it is full all the input ports and output ports are sent with the phase FIFO_FULL through backward and forward paths respectively. This is because both request and responses type transactions can enter this FIFO and all the sockets through which these transactions can enter the router need to be blocked. If FIFO_init_skt or FIFO_targ_skt are full, FIFO_FULL phase is sent through backward and forward paths respectively. If a router receives a FIFO_FULL, it stops from sending transactions through that particular socket until it receives a FIFO_AVAILABLE phase.

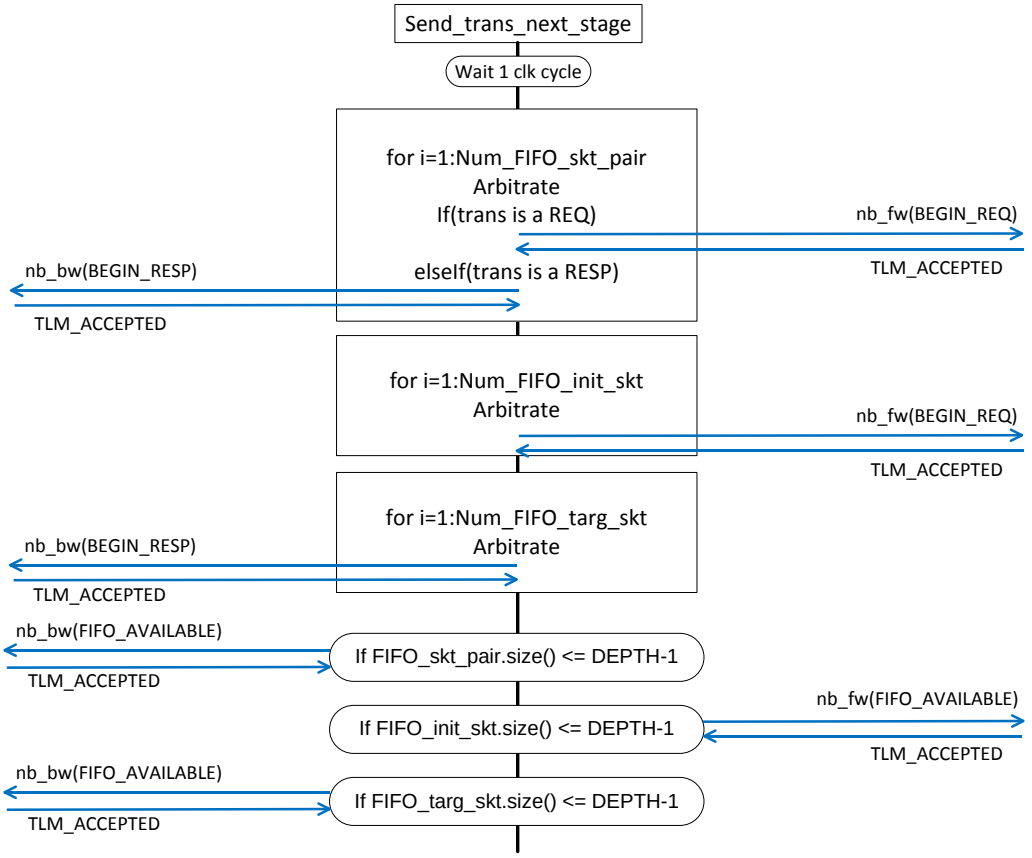


Figure 3.9b: Cycle accurate NoC flow chart- Arbitration

Figure 3.9b shows a part of flowchart of a cycle accurate model. Every clock cycle, depending on the current priority of the FIFO, all the transactions are searched and once a match occurs, it is forwarded to the next stage of router or target. Following this, the priority value is incremented. The size of the FIFOs is checked every clock cycle, and if a FIFO_FULL had been sent by that FIFO and its size reduces to DEPTH-1, depending on the kind of FIFO, FIFO_AVAILABLE phase is sent to the adjacent routers. If it is FIFO_skt_pair, FIFO_AVAILABLE is sent to all input and output ports through backward and forward paths respectively. In case of FIFO_init_skt and FIFO_targ_skt, FIFO_AVAILABLE is sent through all input and output sockets respectively.

There is no need to predict latency before sending transaction out of the router as cycle accurate NoC platform will calculate the delay itself. For example say the elements in FIFO are as shown in Figure 3.10a at the beginning of n^{th} cycle and the current priority for arbitration at the corresponding socket is 2, the transaction with ID-2 is sent through and the priority for next cycle is incremented to 3, and transaction with ID-3 will be sent during $(n+1)^{th}$ cycle.

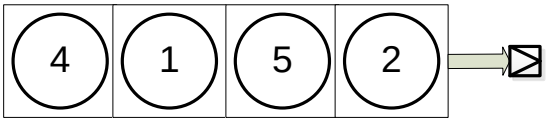


Figure 3.10a: Elements in FIFO at n^{th} cycle

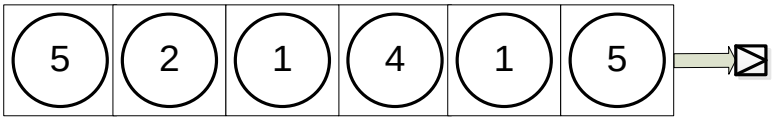


Figure 3.10b: Elements in FIFO at $(n+1)^{th}$ cycle

If transactions have entered the FIFO through sockets 5, 2 and 1, the elements in FIFO appear as shown in Figure 3.10b at the beginning of $(n+1)^{th}$ cycle. Since the priority for the current cycle is 3 and there is no transaction with ID-3, priority is incremented to 4 and transaction with ID-4 is sent through the socket at $(n+1)^{th}$ cycle. This model differs from approximately timed model as only one transaction is sent through a socket every clock cycle and latency is not predicted as the platform can calculate the delay.

3.7 Differences between LT, AT and cycle accurate NoC platform

The differences between different NoC platforms are tabulated below.

Table 3.1: Differences between LT, AT and cycle accurate router models

Feature	**LT**	**AT**	**Cycle-accurate**
Blocking transport	Used	Not used	Not used
Non-blocking transport	Not used	Used	Used
Number of phases used	None	4	6
Synthetic master	Functional	Cycle accurate	Cycle accurate
Synthetic slave	Functional	Functional	Cycle accurate
Arbitration	Not used	Round-robin	Round-robin
Latency modeling	Transaction based	Transaction based	Simulation based

CHAPTER 4

EXPERIMENTATION, PERFORMANCE ANALYSIS AND SIMULATION RESULTS

4.1 Experimental Setup

The NoC architecture provides the communication infrastructure for the cores. In the design, a dead-lock free routing algorithm (X-Y routing) is used. The utilized topology for implementation is a 4×4 regular two dimensional mesh. This topology is shown in Figure 4.1.

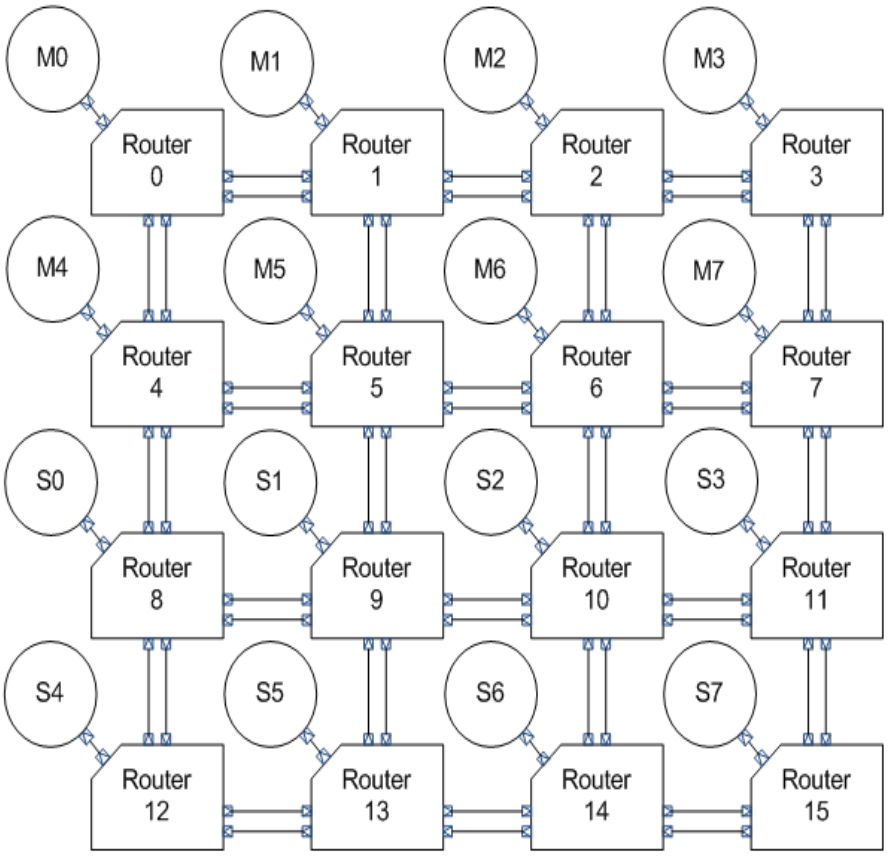


Figure4.1: 4x4 regular mesh NoC

In the above Figure4.1, pentagons represent NoC routers and circles represent the processing elements of this network. It contains 16 routers, and routers are named based on their position in coordinate system. Each router is named as RouterX where X is the ID of the router and the ID is in increasing order from left to right, top to bottom starting from top-left most router. The value of X ranges from 0-15. The processing elements can be either master or slave. The processing elements named as M*i* are synthetic masters where *i* stand for the ID associated with the processing element. And the processing elements named as S*i* are synthetic slaves where *i* stand for the ID associated with the processing element. The value of *i* in both masters and slaves range from 0-7. Each router contains both initiator and target sockets in the direction where a router is connected. And it contains a target socket if connected to a synthetic master core and an initiator socket if connected to a synthetic slave core.

For n-dimensional mesh topologies in NoCs, dimension order routing produces deadlock-free routing algorithms. The X-Y routing is one of the most commonly used algorithms of this kind. The routing algorithm which is used in this design is a version of X-Y algorithm. This algorithm is deterministic algorithm where a transaction object takes routing in one dimension and it continues till this transaction object attains the desired coordinate in that dimension. After that, routing is continued to do the same procedure in the other dimension. This routing algorithm prevents deadlock. According to the position of each router and destination address, routing takes place first in X direction and then in Y direction.

An important characteristic of NoC is Injection rate, which can be defined as number of transaction objects injected by a master core per clock cycle per socket into the network. The injection rate can vary from 0 to 1.

LT master's injection rate is characterized by number of transaction objects sent by a single master within a quantum period. In case of AT NoC model and cycle accurate models, same synthetic master is connected. The block diagram of such a synthetic master is shown in the figure3.2.

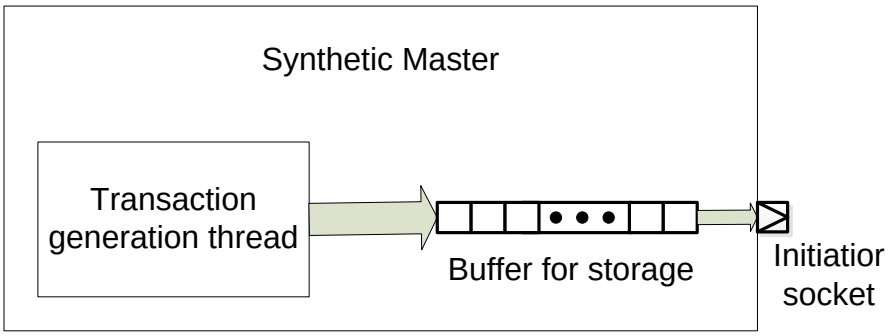


Figure4.2: Synthetic master

The synthetic master contains an infinite sized buffer, which stores the transactions generated by the transaction generation thread. Since at high injection rate, the number of transaction that queue increases due to congestion, an infinite sized buffer is used. The transaction generation thread depending on the injection rate sends transaction objects into the buffer. For example if the injection rate is 0.1, a transaction object is sent into the buffer every 10 clock cycles. And if the injection rate is 0.5 a transaction object is sent into the buffer every two clock cycles. At a particular injection rate, the clock cycle within the injection period, at which the transaction object is sent into the buffer is randomized using uniform random distribution. On the other hand, if the buffer is not empty it sends the transaction object into the network every clock cycle.

The LT and AT slave simply performs memory operation depending on the address attribute of the transaction object and the type of transaction, either read or write. It takes one clock cycle to perform every operation.

Cycle accurate master and slave are similar to approximately timed models but they can also handle additional phases of FIFO_FULL and FIFO_AVAILABLE. When cycle accurate models receive a FIFO_FULL, they do not send a transaction into the platform until it receives a FIFO_AVAILABLE phase.

4.2 Performance Analysis

All measurements were executed on a workstation with INTEL(R) Q9400 2.66GHz quad core processor, 3GB RAM and 32-bit RedHat linux. The platform modeled at the three abstraction levels are connected to synthetic masters and slaves. Synthetic masters generate synthetic traffic load based on following spatial distributions:

- Uniform-random: A master sends traffic to all the slaves with equal probability.
- Hot-Spot: In this distribution, few slaves are selected as hot spots and a certain amount of traffic is sent to these nodes, rest of the traffic is distributed uniformly among all other slaves.
- Complement- A complementary distribution is where destination address is 1's complement of source address. It creates a scenario where master-slave pairs are created. For example, a network has 4 masters and slaves, and they are numbered as 0,1,2,3. Master with ID-0 sends transactions to slave with ID-4 and Master with ID-1 sends transactions to slave with ID-3 and so on.

The performance of SystemC TLM2.0 NoC models are characterized by many metrics. The most important metrics are:

- Latency
- Error rate
- Transaction objects per second
- Simulation speed

One of the most significant metrics being measured is the latency, which represents the delay between the initiation of a transaction object by a master and the receipt of that transaction object by a slave. In practice, this latency is mainly affected by queuing and processing delays. Queuing delay occurs when a router receives multiple transaction objects from different sources heading towards the same destination and needs to queue the transaction objects for transmission. Processing delays are incurred while a router determines what to do with a newly received transaction object. For simulation purposes, the LT slave is modeled with no processing time and merely sending a response back when it receives a transaction object, but the slaves are modeled with one clock cycle latency to process a single operation.

The trade-off between speed and accuracy is studied by comparing the error rate in latency, taking the cycle accurate model as the reference. The error rate is calculated as:

$$\text{Error rate} = \frac{\left|\text{Latency of a model} - \text{Latency of CA model at same injection rate}\right|}{\text{Latency of CA model at same injection rate}} \times 100$$

The other metric used to compare the performance of NoC at different abstraction levels is transaction objects per second, which is defined as number of transactions completed by the NoC platform at a particular injection rate for a running time of 100 seconds. It compares the speed of the models at different abstraction levels. In case of LT NoC platform, a transaction is considered to be complete only when the function is returned to the initiator, whereas in approximately timed and cycle accurate model, the count is incremented only when all the phases of a transaction are completed.

The fourth metric used to characterize the NoC models is the simulation speed. This parameter also shows the comparison in terms of speed of operation to execute a particular number of transaction objects. The figures below in the section of simulation results show the execution time to finish 10^6 transactions completely. Speed up of each model is calculated with RTL-VHDL model as reference.

4.3 Simulation Results

In this section the characteristics of SystemC TLM2.0 NoC models at different abstraction levels will be discussed. Simulation results of these abstraction levels are compared against that of a generic synthesizable RTL-VHDL 4x4 mesh NoC. Different synthetic traffic patterns have been used for evaluating interconnection networks. Uniform-random, Hot-spot and complement are the most widely used traffic models for the analysis of interconnection networks. The 4x4 mesh NoC platform is injected with above mentioned traffic patterns and simulation results are shown below. In case of hotpsot, slaves S0 and S7 are sent traffic with a probability of 30% each and the rest of traffic is uniformly distributed among other slaves.

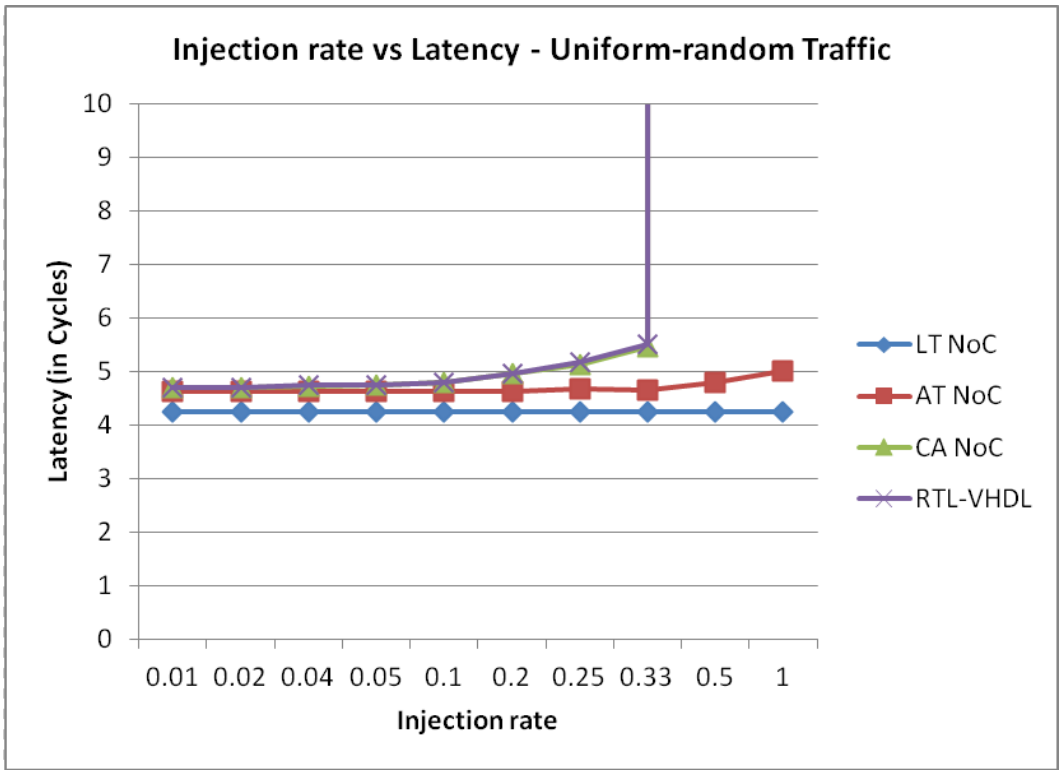


Figure 4.3a: Latency Comparison with Uniform-random traffic

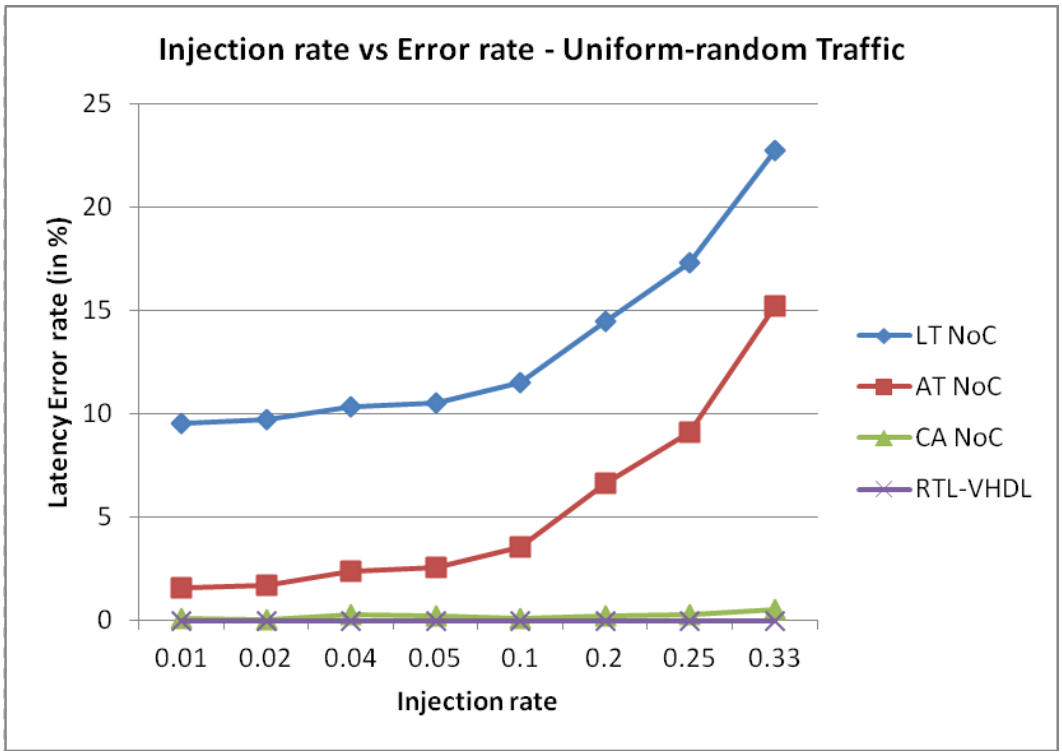


Figure 4.3b: Error rate in Latency with Uniform-random Traffic

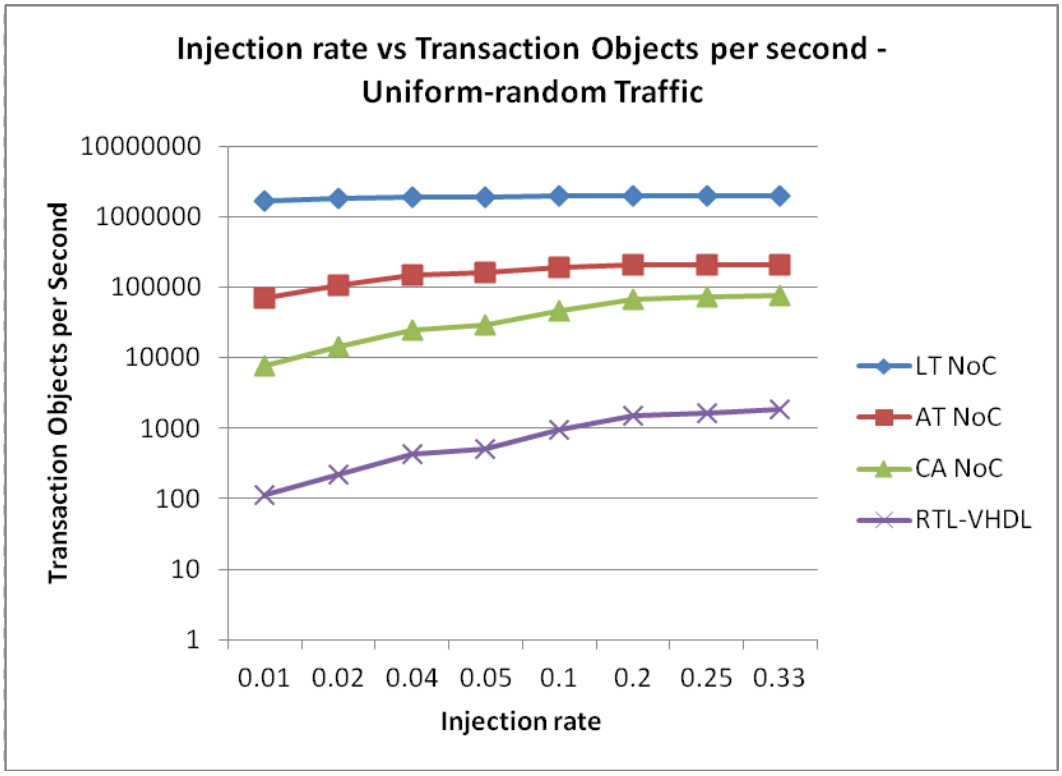


Figure 4.3c: Transaction objects per second with Uniform-random traffic

Table 4.1: Speedup Table for Uniform-random Traffic at injection rate = 0.1

Abstraction Level	**Execution time** (to finish 10^6 trans)	**Speedup**
Loosely timed	0.5 sec	2278.7
Approx. timed	5 sec	227.9
Cycle Accurate	22 sec	51.8
RTL-VHDL	1139.37	1

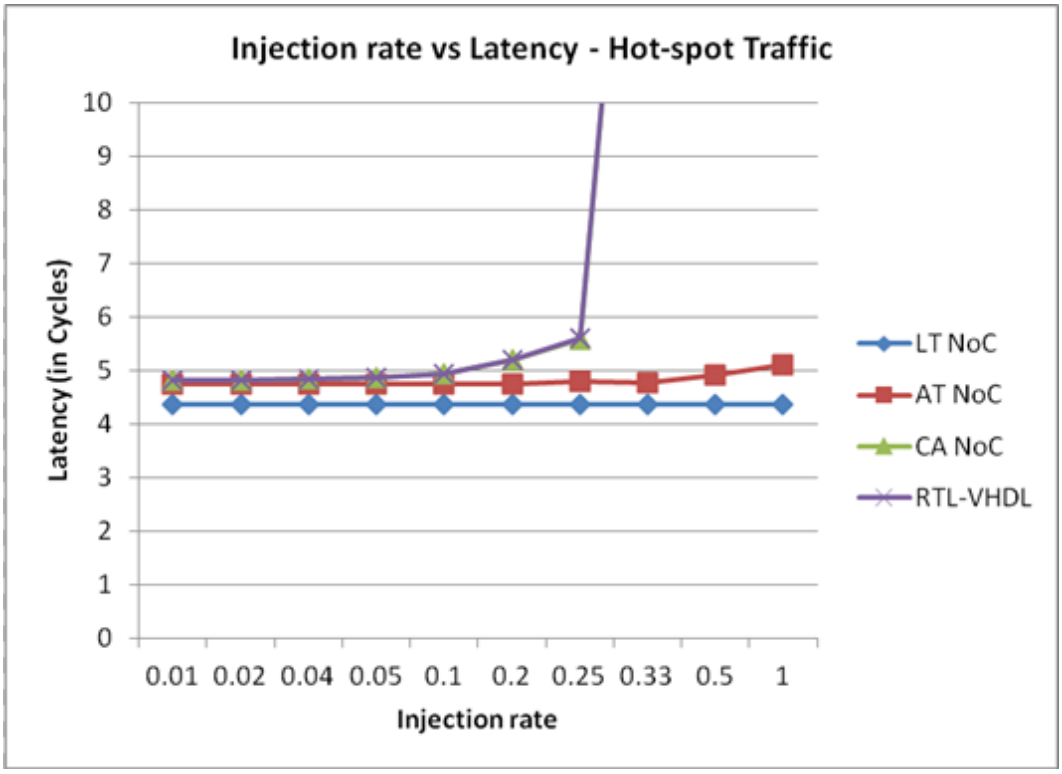


Figure 4.4a: Latency Comparison with Hot-Spot traffic

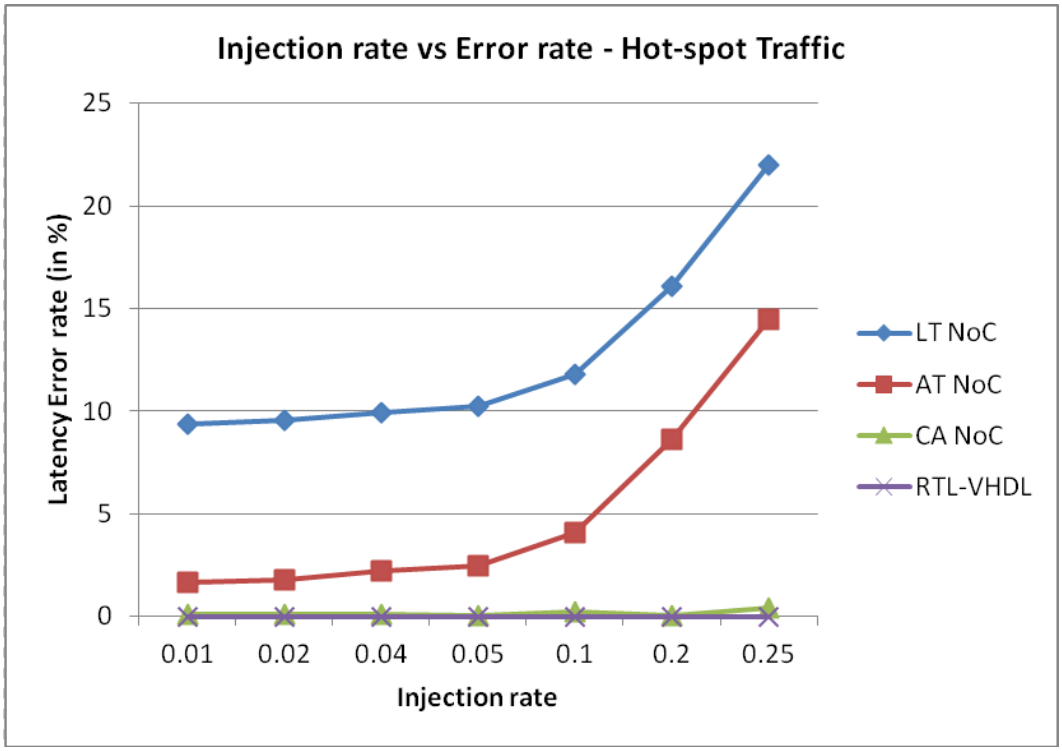


Figure 4.4b: Error rate in Latency with Hot-Spot Traffic

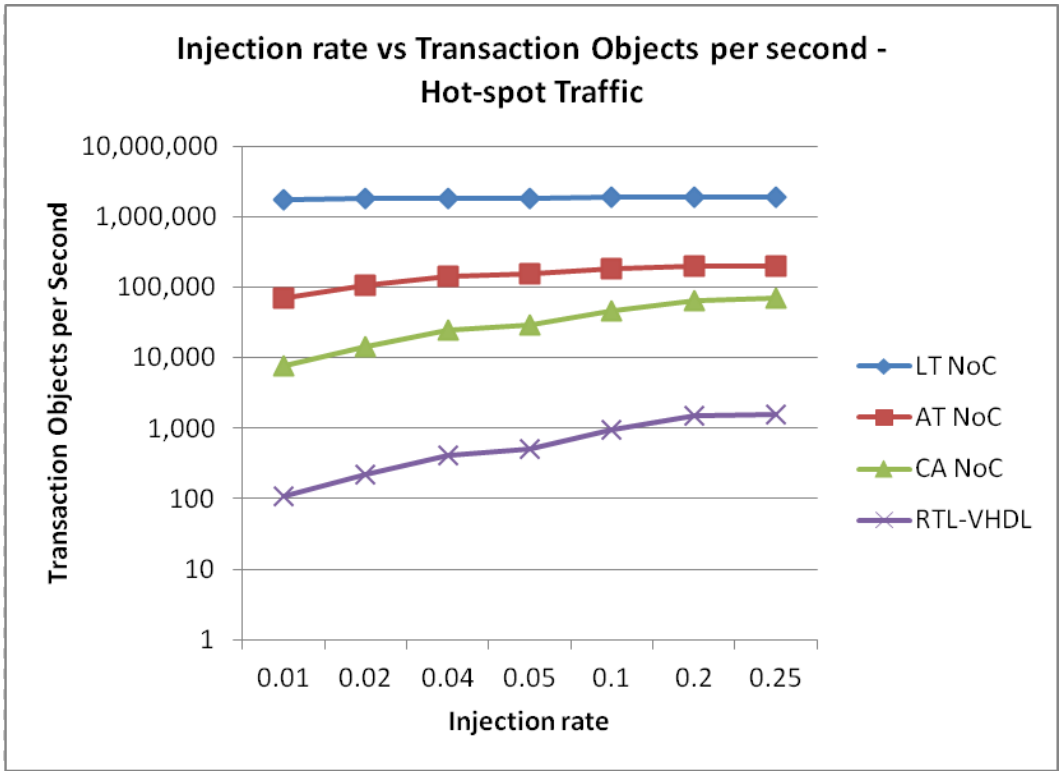


Figure 4.4c: Transaction objects per second with Hot-Spot traffic

Table 4.2: Speedup Table for Hot-spot Traffic at injection rate = 0.1

Abstraction Level	**Execution time** (to finish 10^6 trans)	**Speedup**
Loosely timed	0.5 sec	2349.5
Approx. timed	5 sec	235
Cycle Accurate	22 sec	53.4
RTL-VHDL	1174.75	1

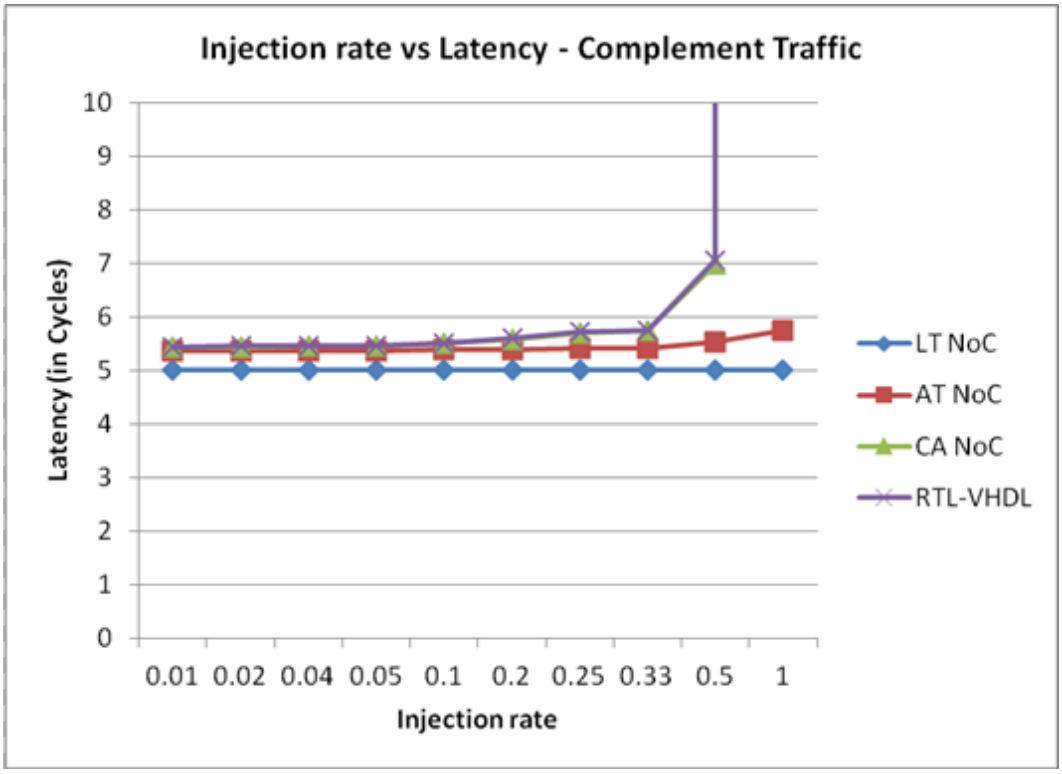


Figure 4.5a: Latency Comparison with Complement traffic

Figure 4.5b: Error rate in Latency with Complement Traffic

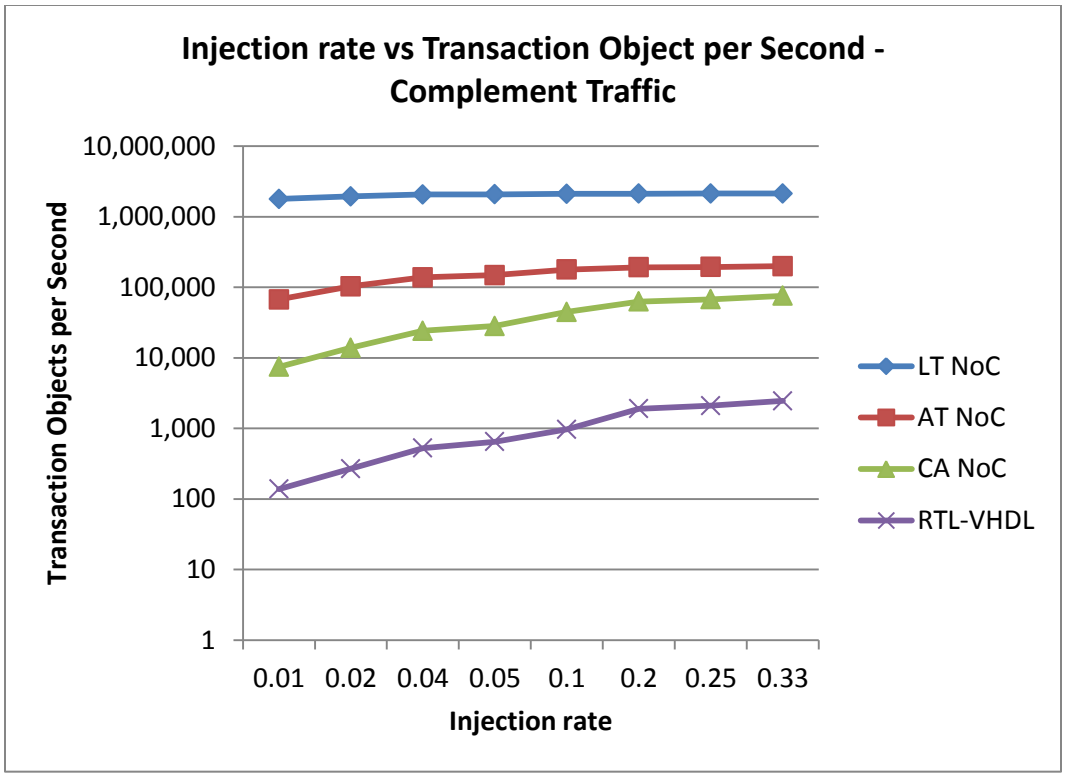


Figure 4.5c: Transaction objects per second with Complement traffic

Table 4.3: Speedup Table for Complement Traffic at injection rate = 0.1

Abstraction Level	**Execution time** (to finish 10^6 trans)	**Speedup**
Loosely timed	0.5 sec	2416.9
Approx. timed	5.5 sec	219.7
Cycle Accurate	23 sec	52.5
RTL-VHDL	1208.47	1

In above plots, it is clear that latency increases with increasing injection rate, this is because of higher congestion taking place at those injection rates. In case of cycle accurate model, it increases exponentially after congestion since the masters are suspended (when FIFO_FULL is sent to master) for many clock cycles as a result of which the transaction are queued in the buffer for many cycles.

The error rate in latency is calculated with cycle accurate model as the reference. Since LT doesn't witness congestion, error rate increases with increasing injection rate, as the latency remains a constant in case of LT-NoC. Since AT model is not cycle accurate and can send multiple transactions in same cycle, error rate is high in case of AT NoC as well, as shown in Figure 3.3b, 3.4b and 3.5b.

The LT-NoC is the fastest as there is very less context switching happening whereas cycle accurate is the slowest among the different models as it involves with huge amount of context switching and also masters are suspended due to back pressure from routers as FIFO depth is limited. Approximately timed models are faster than cycle accurate models since they involve less timing (accuracy) points as the communication protocol in approximately timed is basic protocol and cycle accurate models use two additional phases. Also there is no back pressure in these models.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

In this thesis, OSCI SystemC Transaction-Level models for a NoC interconnect are presented. It describes loosely-timed, approximately-timed and cycle accurate router structures. A simulation performance of 2D 4x4 mesh topology NoC modeled at three abstraction levels was compared against each other. Based on TLM-2.0 standard, these models offer good interoperability.

The models were simulated with synthetic traffic based on uniform-random, hot-spot and complement distribution. Since the LT model doesn't handle multiple transactions on fly, and hence doesn't experience congestion, so the latency of LT models is constant irrespective of the traffic and injection rate. AT model latency is higher than LT model but doesn't increase exponentially after congestion because it's not cycle accurate. Cycle accurate models are closer to RTL level and their results are similar to RTL results. The speed-up of LT model when compared to RTL-VHDL model is around 2200-2450x. But on average the error rate in latency is around 15-18%. AT-models are around 200-250 times faster than RTL-VHDL models with an average error rate of around 5-6%. Depending on the application, the system is modeled at a particular abstraction level. With higher abstraction level a considerable speedup is achieved, although it comes with the trade-off of lower accuracy.

5.2 Future Work

One goal of these models is to provide an approach to explore the design space of NoCs at system level. With these models, designers could develop NoC with different structures and analyze the performance. This model also provides a basic framework for transaction-level model of NoCs. For exploring the design space, these models can used to implement different arbitration schemes (example with QoS), topologies and routing algorithms. Virtual channels can also be used in the NoC to optimize their performance.

REFERENCES

[1] The Open SystemC Initiative (OSCI), "SystemC v2.2 User's Guide," 2007. Available: http://www.accelera.org

[2] The Open SystemC Initiative (OSCI), "Transaction Level Model Standard" v2.0, 2009. Available: http://www.accellera.org

[3] Thorsten Grötker, Stan Liao, Grant Martin, Stuart Swan, "*System Design with SystemC"*, Kluwer Academic Publisher, 2002

[4] David C. Black, Jack Donovan, Bill Bunton, Anna Keist. *SystemC: From the Ground Up. S1:* Kluwer Academic Publisher, 2004

[5] Qualcomm Network on Chip Library, Release 1.6.2

[6] William James Dally and Brian Towles, "*Principles and Practices of Interconnection Networks"*, Morgan Kaufmann Publishers, 2004

[7] Glenn Leary, Krishna Mehta and Karam S. Chatha, "*Performance and Resource Optimization of NoC Router Architecture for Master and Slave IP Cores",* Proceedings of International Conference on Hardware/Software Co-design and System Synthesis (CODES-ISSS), September30-October5, Salzburg, Austria, 2007

[8] Nilanjan Banerjee, Praveen Vellanki, and Karam S. Chatha, "*A Power and Performance Model for Network-on-Chip Architectures*", Proceedings of Design Automation and Test in Europe Conference, Paris, France, February 16-20, 2004.

[9] Hye-On Jang et al., "*High-Level System Modeling and Architecture Exploration with SystemC on a Network SoC: S3C2510 Case Study*", Proceedings of Design, Automation and Test in Europe, vol. 1, pp. 538-543, Feb. 2004.

[10] Martin Streub¨uhr, Jens Gladigau, Christian Haubelt, and J¨urgen Teich, *"Efficient Approximately-Timed Performance Modeling for Architectural Exploration of MPSoCs"*, Forum on specification and Design Languages (FDL 2009), Sophia Antipolis, France, September 22-24, 2009

[11] Seyyed Amir Asghari, Hossein Pedram, Mohammad Khademi, and Pooria Yaghini, *"Designing and Implementation of a Network on Chip Router Based on Handshaking Communication Mechanism"*, World Applied Sciences Journal 6 (1): 88-93, 2009. ISSN 1818-4952

[12] Jianchen Hu, *"Transaction-level Modeling for a Network-on-chip Router in Multiprocessor Syste*m", Master's thesis, Computer Engineering, North Carolina State University, Raleigh, 2009

[13] Luca Fossati, "*Development of the SystemC model of the LEON2/3 Processor",* Contract carried out by Luca Fossati, Politecnico di Milano (Italy)

[14] Gunar Schirner and Rainer D¨omer, "Quantitative Analysis of the Speed/Accuracy Trade-off in Transaction Level Modeling", Transactions on Embedded Computing Systems, 8(1):1 29, 2008.

[15] Lasse Lehtonen, Erno Salminen, and D Hämäläinen, "Analysis of Modeling Styles on Network-on-Chip Simulation", in the proceedings of NORCHIP, 2010

[16] Adan Kohler, Martin Raetzki, "A SystemC TLM2 Model of Communication in Wormhole Switched Netwroks-on-Chip", in the proceedings of Forum on Specification Design Languages, 2009. FDL 2009, pp. 1-4, sep. 2009

[17] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, A. Sangiovanni-Vincentelli, "Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design", in the proceedings of Design Automation Conference 2001, June 18-22, 2001

[18] www.doulos.com/systemc

APPENDIX A

PERL SCRIPT- GENERATION OF NOC PLATFORM

As a part of automated generation of Network on Chip platform, a perl script takes few files characterizing the NoC as input and generates the network on chip platform, provided the SystemC TLM2.0 models of router and processing elements are available. The Figure below shows the files that are read as input and what each file contains and how it is used to generate the platform. The following paragraphs describe each file and its characteristics in detail.

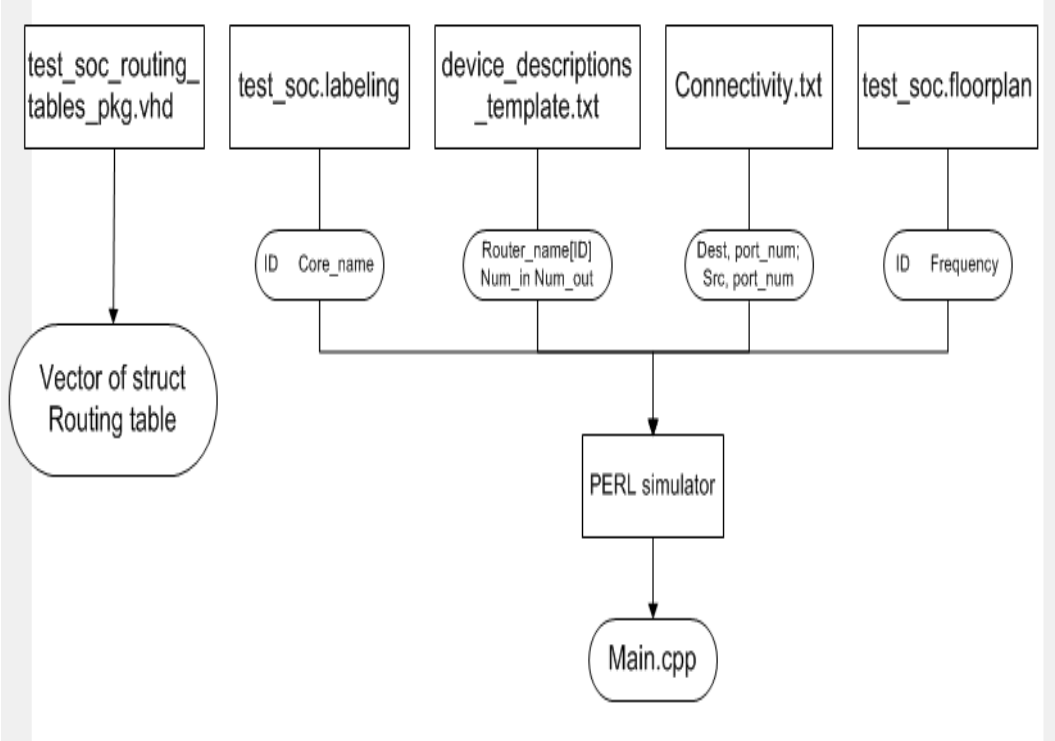


Figure A.1: NoC platform generation

test_soc_routing_tables_pkg.vhd:

In the this thesis, the NoC uses X-Y routing algorithm where a transaction object depending on the present coordinate and destination address keeps transversing in the X direction. Once it reaches the x coordinate of the destination router, transaction object travels in the Y direction till it reaches the destination router. The file *test_soc_routing_tables_pkg.vhd* contains the routing table information for each router. The heading of each routing table is of the form **CONSTANT Rn_ROUTER_TABLE: NOC_ROUTER_TABLE_ARRAY :=**

Where *n* represents the ID of the corresponding router. The following figure shows the routing table of router with ID 0.

```
CONSTANT R0_ROUTER_TABLE: NOC_ROUTER_TABLE_ARRAY :=
                        (
                        ( (16#00000000, 16#FFFF0000, 2) ,
                          (16#00000100, 16#FFFFFFFF, 1),
                         (16#00000200, 16#FFFFFFFF, 3))
                        );
```

Format of contents in *test_soc_routing_tables_pkg.vhd*

The perl script opens the file during compile time and depending on the router being constructed, the corresponding routing table is read which distinguishes from other routing tables depending on the ID of the router. The routing table is C++ vector. Each entry in the vector is a structure with three components: Start Address, Mask and output socket number. For example when the routing table in the above figure is read, first entry would have start address to be 00000000, mask to be FFFF0000 and output socket number as 2.

test_soc.labeling:

This file contains the information of ID associated with the processing elements. The format of the each line in the file is ID followed by the processing element's name. The master cores are named as M*i* and the slave cores are named as S*i*, where *i* is in the range of IDs given to master and slave cores respectively. This ID depends on number of master cores and slave cores. This information is used while creating instances of the Synthetic Processing elements during the creation of NoC platform. The following figure shows a part of *test_soc.labeling* file.

```
0  S0
1  M0
2  M1
3  M2
```

Format of contents in *test_soc.labeling*

device_descriptions_template.txt:

This file contains the number of the input and output ports of each router. The format of each entry is router name, number of input ports, and number of output ports. The following figure shows a part of *device_descriptions_template.txt* file. This information of number of input ports and output ports is used for the creation of target sockets and initiator sockets for each router respectively.

```
R39, 9, 9
R0, 3, 3
R23, 3, 3
```

Format of contents in *device_descriptions_template.txt*

Connectivity.txt:

This file contains the details of connection between routers and routers to processing elements. The format of this file is destination core/router, port number followed by semicolon has the source core/router, port number information. The following figure shows a part of the file *connectivity.txt*. The first entry in the following figure means the output port#1 of router R0 is connected to input port#1 of router R2.

R2,P1;R0,P1
R0,P2;R2,P1
R0,P1;R39,P1
R39,P1;R0,P1

Format of contents in *Connectivity.txt*

test_soc.floorplan:

This file contains the details of frequency of operation of the processing elements. The format of the contents of this file is ID of the processing element followed by the frequency of operation of the corresponding core. The following figure shows a part of the file *test_soc.floorplan*. This information is used while creating the platform using approximately timed and cycle accurate models. The unit of frequency is MHz. The first entry in the following figure means the frequency of operation with ID 0 is 266MHz.

0 266
2 133
4 133

Contents of *test_soc.floorplan*