



TAMPERE UNIVERSITY OF TECHNOLOGY

IRFAN ULLAH  
SYSTEMC MODEL OF HIERARCHICAL NETWORK-ON-CHIP FOR  
SYSTEM-LEVEL ON-CHIP MULTI-CORE PLATFORM  
Master of Science Thesis

Examiners: Jari Nurmi

Sanna Määttä

Examiners and topic approved  
in the Computing and Electrical  
Engineering Faculty Council  
meeting on 8 September 2010

## **Dedication**

To the Holy Prophet Muhammad (SAW) and his companions who laid the foundation for modern human civilization and paved the way to social, political, moral, ethical, spiritual, cultural, and metaphysics revolution.

## ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Master's Degree Programme in Information Technology

ULLAH, IRFAN: SystemC Model of Hierarchical Network-on-Chip for System-Level On-Chip Multi-Core Platform.

Master of Science Thesis, 47 pages

August 2010

Major: Computer Systems

Examiners: Prof. Jari Nurmi, MSc Sanna Määttä

Supervisor: Prof. Jari Nurmi

Keywords: Network-on-Chip, System-Level Modeling, SystemC

System-Level Modeling is one of the key tools to speed up the process of design space exploration. Open source system level design tool is the solution for SMEs to get maximum benefit out of system level modeling in affordable range.

SystemC is a C++ library extension that is used for open source system level modeling. In this thesis, a NoC based on hierarchical NoC for Ninesilica is modeled using SystemC. The Ninesilica multi-core platform that is developed at Department of Computer System in Tampere University of Technology. The system level NoC model is able to simulate the communication network with several number of nodes and data packets. The modeled NoC is able to give useful information regarding to delay, data packet buffering and number of clock cycles required to transfer all the data packets. The user can also be able to get information about the position of any data packet at any clock cycle in the network.

The behavior of the communication network is analyzed with different number of nodes and several network configurations. The data load is also varied in order to verify that the NoC model is working properly. The NoC model successfully completed all the tests and gives the results as expected. The NoC model is able to buffer, transmit, and receive data packets without any loss of data packets. The NoC model can be configured and re-configured. The simulation results are written to a text file. Several comparisons between different network topologies with variable data load is also made and some conclusions based on those results are made.

## **PREFACE**

The research work in this thesis was done at Department of Computer System in Tampere University of technology (2009-2010). In this thesis system-level model of Network-on-Chip is implemented. The research work in this thesis is supported by an ARTEMIS project called SYSMODEL. The aim of SYSMODEL project is to support SMEs and increase their efficiency using high level modeling tools.

I would like to thank Prof Dr(tech) Jari Nurmi and Roberto Airoidi (MSc) for their continues guidance. I especially thank to Sanna Määttä (MSc) for her help in writing process and Jarno Mannelin (MSc) for his help and guidance in SystemC related issues. I sincerely admit that without the help of the above mentioned people the completion of this work would not be possible. I would like to thank Waqar Hussain (MSc), Muhammad Salman, Omer Anjum (MSc), Mubashir Ali, Hadaith Ullah (MSc), Ali Khan, Wasim Muneer (MSc) and Faraz Amjad (MSc) for their guidance and moral support. I would also like to thank all the members of Pakistani Community here in Tampere that provides very healthy, cooperative and home like environment. I am extremely thankful to my parents and family for their prayers and trust in me.

I am very thankful to Government of Pakistan, Higher Education Commission (Pakistan) and COMSATS Institute of Information Technology for providing me the opportunity for studying abroad (here in Finland). I would also very thankful to Government of Finland for providing us (Students) very good living facilities and free of cost high-tech education.

## Table of Contents

ABSTRACT .....	iii
PREFACE .....	iv
Abbreviations .....	vii
List of Tables.....	viii
Introduction .....	1
System Level Modeling .....	4
2.1 Motivation .....	4
2.2 The Y-Chart.....	5
2.3 System Level Modeling Languages.....	6
2.4 System Level Synthesis Tools.....	7
SystemC .....	11
3.1 SystemC Simulation Kernel .....	13
3.2 SystemC Components .....	14
3.3 SystemC and VHDL .....	15
System-on-Chip and Network-on-Chip .....	16
4.1 System-on-Chip .....	16
4.2 SoC Design Benefits .....	17
4.3 SoC Design Challenges .....	17
4.3.1 System Level Modeling.....	18
4.3.2 Block-Based Modeling.....	18
4.3.3 Platform-Based Design.....	18
4.4 Communication Networks.....	19
4.4.1 OSI Model .....	19
4.4.2 Network Components.....	20
4.4.3 Network Types.....	21
4.5 Network-on-Chip .....	22
4.5.1 NoC Protocol .....	22
Register-Transfer Level Model of a Network-on-Chip .....	24

---

5.1 Synthesizable Switching Logic for NoC .....	25
5.2 NoC Architecture .....	25
System-Level Model of a Network-on-Chip.....	27
6.1 Implementation of the Network Architecture.....	28
6.2 Extract and Decode Data.....	31
6.3 Data Packet Routing.....	33
6.4 Functional Diagram of NoC Model .....	34
6.5 Testing and Results .....	35
6.5.1 Testing All Buffers and Links .....	36
6.5.2 Worst Case Test with Minimum Number of Data Packets .....	37
6.5.3 Network Analysis with one Thousand Data Packets .....	40
Conclusions and Future Work.....	43
REFERENCES .....	44

## Abbreviations

ASIC	Application-Specific Integrated Circuit
COFFEE	Core For FrEE
CPU	Central Processing Unit
DMA	Direct Memory Access
FIFO	First In First Out
IC	Integrated Circuit
IP	Intellectual property
ISS	Instruction Set Simulator
LAN	Local Area Network
MPSoC	Multi-Processor System-on-Chip
NoC	Network-on-Chip
OSCI	Open SystemC Initiative
OSI	Open System Interconnection
PBD	Platform Based Design
QoS	Quality of Service
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
RTOS	Real-Time Operating Systems
SLD	System Level Design
SLM	System Level Modeling
SoC	System-on-Chip
SSDM	System-level Synthesis Data Model
TDMA	Time Division Multiple Access
TUT	Tampere University of Technology

## List of Tables

6.1: Details of Nodes in a 4x4 network configuration	29
6.2: Details of the simulation environment	35

# Chapter 1

---

## Introduction

The recent progress in semiconductor technology enables the manufacturers to integrate multiple processors on a single chip [1]. Latest improvements in silicon manufacturing technology decrease the size of the Integrated Circuit (IC) and power consumption. This increases the overall efficiency of the systems. State of the art systems, for instance digital signal processors, contain application specific accelerators, programmable coprocessors, memory hierarchy and I/O blocks on a single chip [5]. Mapping multiple processing and memory units on a single chip enables us to develop complex system but the cost of research and time to market increases [2]. There are various methods to overcome this problem: Hierarchical approach, divide the system problem into smaller parts. Top down approach, start the design in high level of abstraction and refine into lower levels.

It is very expensive and time consuming to start system design directly from low levels of abstraction (chip fabrication is a very expensive as well as time consuming process). For instance changes in the hardware are expensive in terms of effort and time. Also error detection and correction is a very exhausting task. Most of the time, the problem is irreversible and needs major changes in the hardware architecture. The effort and cost required is almost the same when designing and manufacturing a new chip. The issue is not the availability of the necessary information, but there are so many available design options. That is why we need design exploration to test various options and verify the correct functionality and Quality of Service (QoS) [3].

In order to overcome the problem of selecting a suitable design for hardware architecture, a system level model is required. Basically, a system level model is a behavioral reference of the entire system. High level synthesis tools are often used to convert the system level model representation to a Register Transfer Level (RTL) description. These tools can also automatically allocate the resources for the processes and schedule them.

There are several different languages available to develop a complete system at system level, for instance SpecC, System Verilog, SystemC, and Rosetta. SystemC is a class library extension of C++ that is developed for system level modeling of the hardware software co-design platform. The modification in the architecture at this level is much easier in terms of cost, effort and time. In SystemC, systems can be developed at high abstraction level and then process them downwards to RTL [4].

The work done in this thesis is a part of SYSMODEL project. The main aim of SYSMODEL project is to provide open source system level methods & tools to Small and Medium size Enterprises (SMEs) [11]. These system level tools help in design and implementation of time and power critical heterogeneous systems. In this thesis, a system level model of a Network-on-Chip (NoC) is presented. The model is implemented using SystemC and based on Ninesilica hierarchical NoC. The basic aim of this thesis is to create a system level NoC model that

- Must be implemented in high level language.
- Must be cost effective.
- Easily reconfigurable.
- Can be connected with a system level Instruction Set Simulator (ISS). Together to form a system level model of Ninesilica Multi-Processor System-on-Chip (MPSoC).
- Can be exploited in design space exploration and validate the correct behavior of NineSilica NoC with different numbers of nodes and traffic load on the network.

The rest of this thesis is organized as follows. Chapter 2 describes system level modeling, motivation for using system level modeling, several system level languages and synthesis tools. Chapter 3 describes SystemC in more detail, its simulation kernel and the main components that are used during the implementation of system level NoC. Chapter 4 gives an overview of System-on-Chip (SoC), NoC, Open System Interconnection (OSI) model, benefits of using SoC, and SoC design challenges. Various types of networks, network components, and communication protocols are also discussed. Chapter 5 presents a low

level model of hierarchical NoC that is already implemented at the Department of Computer Systems at Tampere University of Technology. Some of the important features and architecture of the NoC are explained. Chapter 6 gives an overview about the system level NoC model, its implementation and functionality. Some tests were made to validate the proper functionality of the NoC model. Also some comparisons and analysis were made among different network topologies and data load. Finally Chapter 7 concludes the whole thesis and gives a few suggestions for the future work.

# Chapter 2

---

## **System Level Modeling**

The traditional design method to design the system directly at lower levels of abstraction has become unfeasible due to time to market pressure and the increasing complexity of embedded systems. The general rule to address the complexity of the system is to increase the abstraction level to system level. A system level model captures the behaviour of the whole system based on the system specification. System level model represents concurrent and/or sequential processes execution. It starts with the behavioural model and can then be extended further by adding several processing and communication blocks that add important properties to the system such as application algorithms, communication protocols, connectivity and routing to the system. Each model is considered to be the specification to the next level model, in which more implementation details are added after more design decisions are made. Therefore, the methodology is to determine the specifications of the system and then further explore different available possible solutions for the system. Consequently the designer is able to select the best available solution to get the refined version of the complete system [32].

### 2.1 Motivation

The progress in silicon technology enables the system designer to integrate multiple processing cores on a single chip with enhanced functionality and to support numerous applications. For instance a mobile phone is now transformed into a mini computer. State-of-the-art mobile phones are often equipped with a digital camera, MP3 player, Internet access, large memory space, personal digital assistance, and various communication interfaces to communicate with its environment. This trend is due to the increase in the number of features that a single chip is capable to integrate as well as the market demand. The increasing capability of a system on a single chip clearly indicates that the complexity

of the system also increases exponentially. To handle the increasing complexity we need for instance software-enabled solutions that meet time-to-market and flexibility requirements [3].

Design complexity makes design and verification complicated. Due to the complexity, abstraction level shift from the RTL modeling to System Level Modeling (SLM) is inevitable. SLM reduces the complexity of the system because system designer has to handle fewer details and SLM tools take care of these details at the respective design phase. However, some of the details are not considered at this level of abstraction such as power management. The mentioned properties enable the designer to give more attention to the system behavior rather than system architecture. SLM increases the speed of system simulation while being accurate enough [6][32].

Orthogonalization of concerns is also one of the important features of SLM for conquering the complexity by separating parts of the design process [33]. The error domain in SLM is comparatively smaller than RTL that makes it easier and faster to debug the system. SLM is composed of software model that represents the behavior of the hardware. It makes system debugging cost-effective and designer is able to find bugs during the design phase of the system.

## **2.2 The Y-Chart**

Most of the embedded systems are heterogeneous. Due to the heterogeneity, system exploration at low level is a difficult task. System design at high abstraction level is cost effective and possesses more flexibility to explore alternative architectures [10].

System designer starts designing the system using the abstract specifications. In figure 2.1, system behaviour consists of the set of application that are mapped to the system architecture. Different sets of applications are mapped onto several different architectures to get the best possible solution. In order to achieve the best solution after mapping, performance analysis is done. During performance analysis designer can restructure or modify the application to improve the design [10].

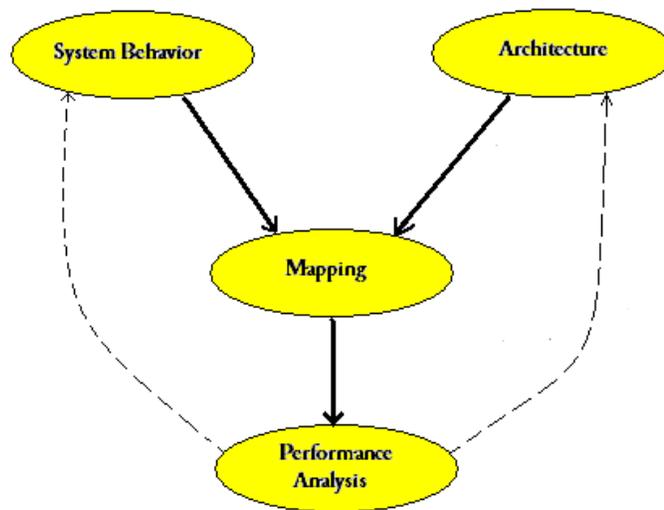


Figure 2.1: Y-Chart [10]

### 2.3 System Level Modeling Languages

SLM tools and languages are needed to model system at high level of abstraction. Various approaches exist to develop a perfect SLD platform but still the SLD languages are going through the development phase. The ultimate goal is to develop a single platform or tool for SLM that is able to complete the whole system design flow. Some SLD languages are briefly introduced below.

**SystemC:** SystemC is an open source C++ class library extension developed by Electronic Design Automation (EDA) companies in 1999. SystemC covers the whole system design flow. It has the properties to model and simulate hardware such as simulating concurrent processes, latencies, delays, and clocks. SystemC library allows the designer to model channels, modules, ports, and processes and to simulate them [11][12]. SystemC is described in more detail in Chapter 3.

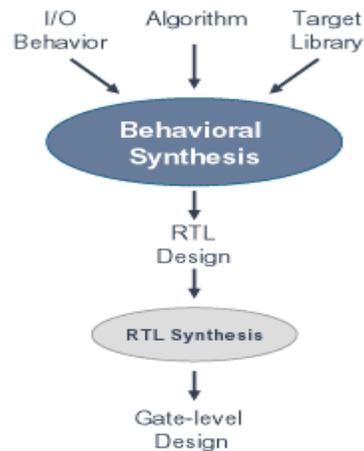
**System Verilog:** System Verilog is a unified hardware description and verification language based on Verilog HDL. It is developed by the Accellera organization. System Verilog adds new features, such as new data types, multidimensional packet arrays, enumerated data types, structures, unions, procedural blocks, and classes to Verilog HDL. System Verilog supports both architectural and behavioural modeling [13].

**SpecC:** SpecC is a system level modeling language. It is the extension of ANSI C language. It was developed by Rainer Dömer and Daniel Gajski in 2001 at the University of California. SpecC uses IP centric co-design methodology for embedded system design, modeling and specification. Although it is not as popular as SystemC, it also contains some important features of SLD language such as concurrency, synchronization, state transitions, and a composite data type [14].

**Rosetta:** Like any other SLD language the main aim of Rosetta is to compose heterogeneous specifications in one semantic environment. The complete system is designed by using components and facets. Facets define the behavior and complete functionality of the system [15] [16].

## 2.4 System Level Synthesis Tools

System level synthesis is the process of converting SLD representation to RTL representation, in order to eventually implement it on FPGA as an Application Specific Integrated Circuit (ASIC). There are various tools available that will automatically make this transformation and optimization of untimed or partially timed system level functional model into fully timed RTL design or model. This will reduce the amount of manual effort. It also decreases the development time and the time to market. System level synthesis tools allow the system designers to concentrate on the behavioral aspects of system design and verification of the modules. Hardware aspects such as process scheduling and resource allocation are taken care by the synthesis tools itself. Some other benefits of system level synthesis tools are the increase in the design productivity, reduction of design errors, and the speed up of the verification process [17][18].

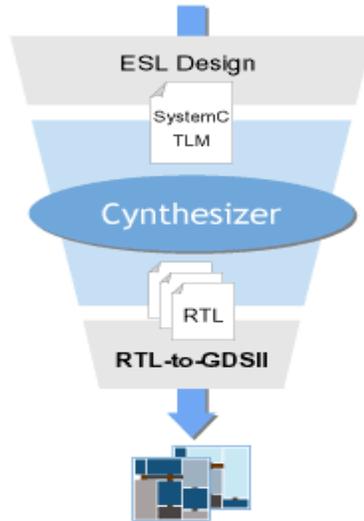


**Figure 2.2:** Generic System Level Synthesis Flow [18] © Forte Design Systems

The general system level synthesis flow is shown in Figure 2.2. The behavioral synthesis tool automatically translates the design from high level of abstraction into RTL design. The behavioral description of modules that includes I/O actions and algorithms for optimizations serves as an input to the behavioral synthesis tool. At this stage the hardware description is analyzed to determine operations and the dataflow dependencies.

The target technology libraries are added as an input to the behavioral synthesis tool for the selected fabrication process. Those libraries include a set of directives that contain information about the timing constraints and a cycle-to-cycle schedule of the operation required by the algorithm. The operations are assigned to the specific processing units such as adders, multipliers or comparators by the allocation and binding algorithms [18].

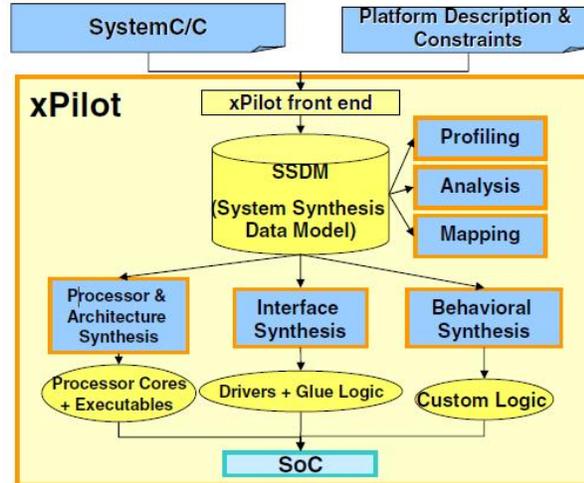
**Cynthesizer:** Cynthesizer is an industrial tool that provides the environment for behavioral synthesis and verification. In addition, it also handles FPGA prototyping and design management. It uses a SystemC model as an input and generates RTL code for mapping on FPGA. The environment provided by Cynthesizer is open source, customizable, and it is based on the C++ standard language. No extra tools are needed because Cynthesizer includes all the tools that are needed for converting system level code to RTL and further to GDSII code (from system design level code to system implementation level) as shown in figure 2.3 [18].



**Figure 2.3:** SLD to RTL using Cynthesizer [18] © Forte Design Systems

**xPilot:** xPilot is a synthesis tools that can automatically convert a C or SystemC description into RTL. xPilot is developed at the University of California, Los Angeles. xPilot can optimize power consumption, complexity of logic, and number of interconnects. xPilot also supports platform-based system level synthesis for both application specific configurable processor and heterogeneous multi-core systems [19].

Figure 2.4 shows design flow of the xPilot. xPilot accepts synthesizable C or SystemC code as an input. The front end compiler optimizes the code or behavioral description. Several high-level programming constructs are constructed from the low level virtual instruction set. After that, elaboration is performed to extract processes, ports, channels, and their interconnection topologies. The internal System-level Synthesis Data Model (SSDM) is constructed. Processes and channels are the basic building blocks of the SSDM. Processes are interacting with each other using ports and channels. Channels implement interfaces to capture a certain communication protocol. Performance analysis, simulation and profiling are performed on SSDM. The system is manually partitioned by the system designer into software and hardware based on the performance analysis [19].



**Figure 2.4:** xPilot system-level synthesis framework [19]. © IEEE

After partitioning the system, behavioral synthesis engine is invoked to compile the hardware part. xPilot generates RTL level VHDL together with the associated constraint files and also generates RTL SystemC code for co-simulation with other software/hardware modules specified in SystemC. An interface synthesis module is being developed to integrate the microprocessors and custom logics together to that generate the software drivers and glue logic [19].

# Chapter 3

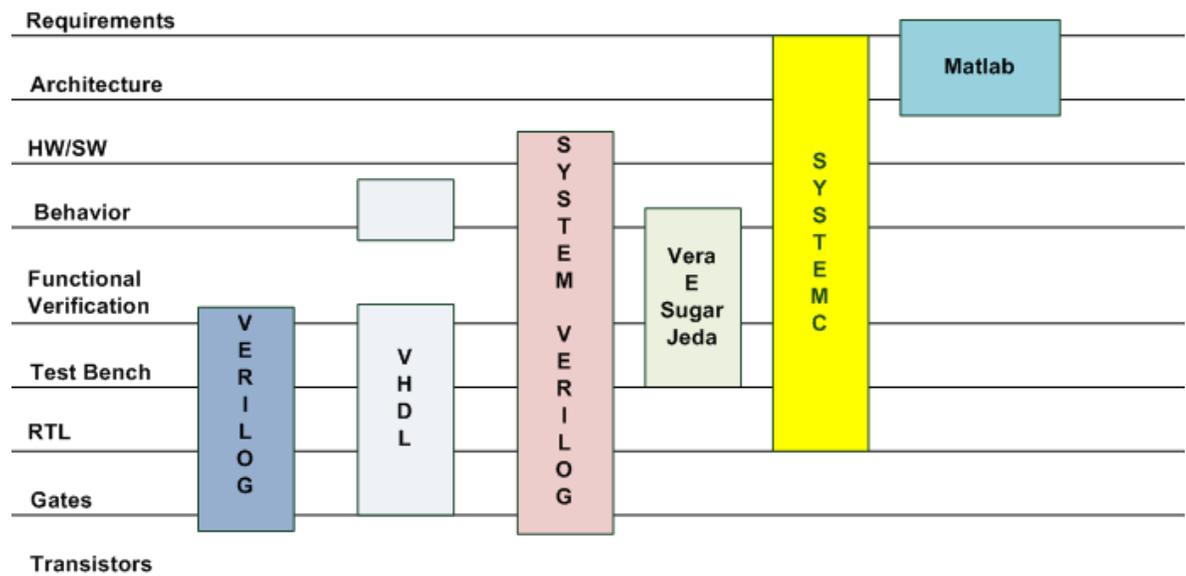
---

## **SystemC**

SystemC class library is developed by open SystemC initiative (OSCI) organization in order to accomplish the growing demand for the fast exploration of software and hardware co-design. It enables the modeling of cycle-accurate software algorithms, concurrent execution of processes, hardware architecture and communication interfaces at high-level of abstraction. In SystemC, it is possible to model the entire system as an executable specification at system level that helps to study the behavior of the complex system without considering the low level implementation details. The system designer is able to modify executable specifications to validate and verify the functionality of the system even before hardware implementation. Different methods such as test benches for functionality verification, automatic synthesis tools for converting from SLD to RTL, and reusing early system models are used to reduce the development and implementation time and to avoid the chances of inconsistency and errors [12][20].

SystemC may not be the best solution to every specific design problem. Compared to other tools and languages available for system simulation and synthesis SystemC covers the vast field in one single environment (Figure 3.1). That is also the main objective of SLM. SystemC is a way of conceptual implementation of the system. The promises of the upcoming version of SystemC (version 3.0) are to model complete operating systems that supports embedded software models, schedulers, preemption, dynamic process and dynamic channels among others [11][20].

One of the main objectives of developing the SystemC library is to handle the increasing complexity of systems. The common practice to address the complexity is to raise the level of abstraction. SystemC addressed the complexity issue with the help of properties such as using the high abstraction level for modeling, reusable designs, and design automation [20].

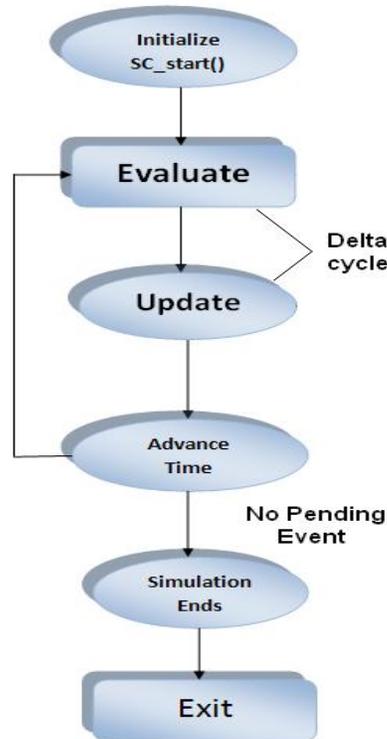


**Figure 3.1:** Comparison of SystemC with other languages and design tools [20]

Most of the developing time for the creation of modern embedded systems is spent on the creation of the hardware architecture, software algorithm and integrating them on one single platform and it never ends here. If the system fails to achieve the expected goals or target then the system designer has to start the development cycle of the system from the beginning again [20].

In SystemC, the behavioral model of the hardware architecture and all the required hardware components are designed and modelled to mimic the actual system hardware architecture. In addition, SystemC provides a single environment for system design, simulation, and verification. This speeds up the system development because the system designer does not have to switch the design environment and perform code transformation for simulation or functional verification. The designer just has to modify the code and simulate. If the output does not meet the requirements, the iteration continues until expected results have been achieved [20].

There are wide ranges of industrial tools that support SystemC. The SystemC is an open source library, to start working with SystemC does not require any costly commercial tools. It can be used with a gcc compiler. That is one of the main reasons that SystemC is popular among academia.



**Figure 3.2:** SystemC Simulation Kernel

### 3.1 SystemC Simulation Kernel

The SystemC includes simulation kernel that is used to simulate system models. The SystemC kernel goes through different phases during simulation as shown in Figure 3.2. In the initialization phase, the processes that need to be executed are selected and executed. In the update phase, all the values of the channels (channels are the communication elements of the SystemC) that are assigned in the previous phase are updated by the kernel. An evaluation followed by the update is known as the delta-cycle. After that the simulation time advances. If there are more pending events, they are selected and executed. If there are no pending events the simulation ends [4].

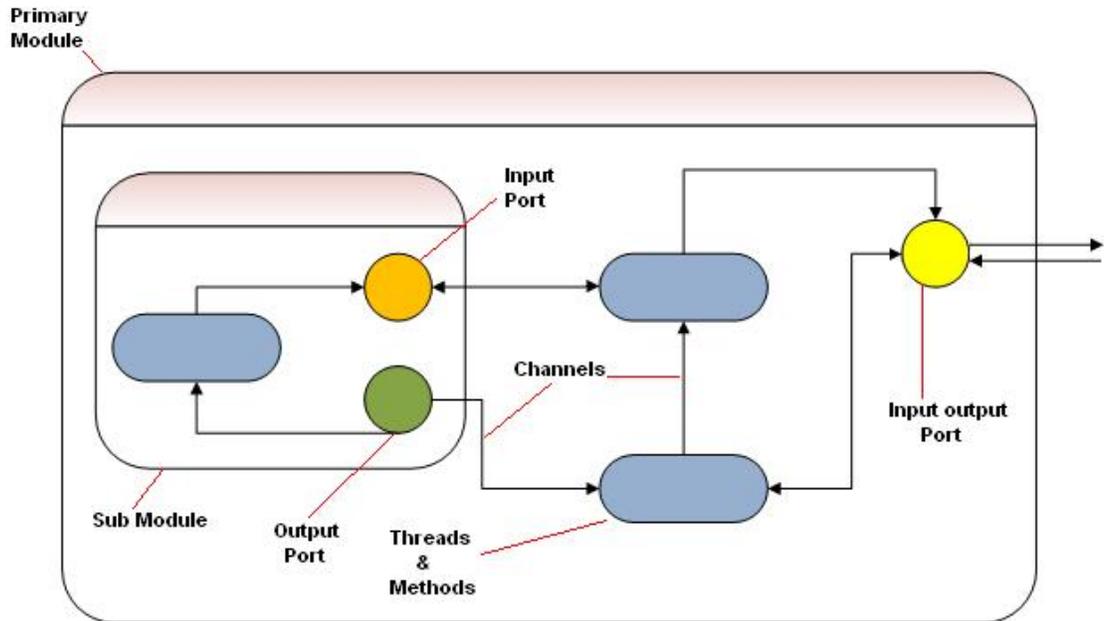
## 3.2 SystemC Components

SystemC uses various components to implement software hardware co-design. They are briefly described in this paragraph. `SC_MODULE` is one of the basic building blocks of SystemC.

A module may contain processes and other modules. Modules receive or transmit data to or from other modules with the help of ports. SystemC has three kinds of ports: input, output, and in-out port. Every module contains at least one process that defines the functionality of the module. There are two types of processes: `SC_METHOD` and `SC_THREAD`. The execution of `SC_METHOD` does not consume time unlike `SC_THREAD` because `SC_THREAD` supports waiting for an event to occur. SystemC methods are generally asynchronous blocks while threads consist of asynchronous processes and also synchronous processes (clocked threads). SystemC processes are concurrent and they are used to describe the functionality of the system. Modules and processes have a functional interface that helps in hiding the implementation details.

In SystemC, signals are just like wires. They connect ports with each other. Signals can be either resolved or unresolved. Unresolved signals have only one single driver (bus) while resolved signals may have more than one driver. Ports of one module are connected to other modules with the help of channels. The difference between signals and channels is that signals are equivalent to simple wire while channels are communication element that may be either a simple wire or consist of complex communication algorithms. In SystemC there are two types of channels: primitive channels and hierarchical channels. A primitive channel is a non abstract class derived from `SC_PRIM_CHANNEL`. A primitive channel is fast, light weight, and allow to call the `REQUEST_UPDATE ()` function. A hierarchical channel is also a non abstract class but is derived from `SC_MODULE ()`. Hierarchical channels are complex and they connect ports and processes.

SystemC has a wide range of data types. All the data types that are supported by C++ are also supported by SystemC. In addition, SystemC also has the user defined data type. Most of these components are illustrated in Figure 3.3. It is not necessary that all of these components are included in the system design [19][20][21].



**Figure 3.3:** SystemC components

### 3.3 SystemC and VHDL

In VHDL, design reuse is attained by using libraries of design units (structural hierarchy). In contrary, SystemC also has several other additional reuse mechanisms based on overloading, templates and inheritance are also available. In VHDL, regular structures (composed of processes) are modeled by using generate statement but in SystemC declaring an array of components is enough. In VHDL, overloading can only be applied to functions and procedures. In SystemC, overloading mechanism can also be applied to constructors of classes that allows dynamic configuration of threads and modules [36].

# Chapter 4

---

## **System-on-Chip and Network-on-Chip**

### **4.1 System-on-Chip**

A SoC consists of hardware blocks such as programmable processors, on-chip memories, accelerators, timers, voltage regulators, interfaces to peripheral devices, and software to manage these hardware blocks, all integrated on a single chip. The presence of all these hardware blocks mentioned above is not necessary but a SoC would contain at least one programmable processor and on-chip memory. All these blocks are connected to each other by a data bus or NoC and the data is routed from the source to the destination by the router with the help of a standard data packet routing protocol. The main objective of SoC design flow is to develop hardware and software in parallel. Most of the hardware blocks used in SoCs are pre-verified and tested. Reusable hardware blocks will accelerate the design process and reduce errors [22][23].

Reusable, predesigned, and verified components that are normally available in the form of synthesizable code or an integrated circuit are called Intellectual Properties (IP). There are three types of IPs depending on their level of flexibility and optimization: soft core, hard core, and firm core. A soft core IP is generally provided in the form of synthesizable RTL code written in VHDL or Verilog. It is the most flexible IP core; the designer is allowed to modify the design according to the system requirements. Firm cores are specified as gate-level netlists and are partially modifiable. They are suitable for placement and routing. A hard IP core does not allow modifications. The designer knows only its output and inputs. Generally hard IPs are in the form of GDSII (a database file format, which is the de facto industry standard for data exchange of integrated circuit or IC layout artwork) that contains layout and timing information [22][23].

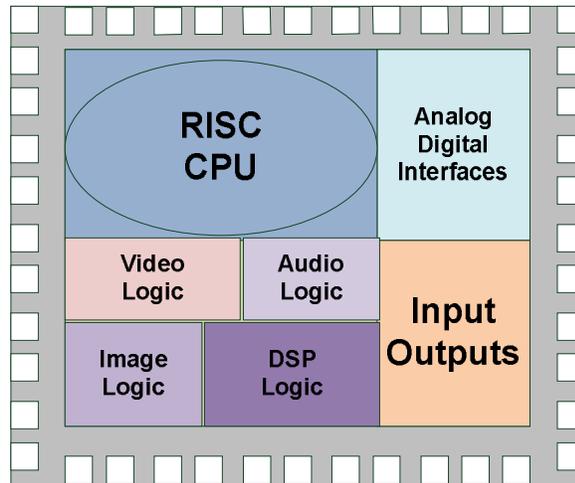


Figure 4-1 An example layout of System-on-Chip [22]

## 4.2 SoC Design Benefits

SoC design methodology reduces the overall cost, design time, and verification effort. SoC design methodology uses IP cores that are pre-verified. This decreases the research cost, which is the major cost of a product. This also saves the cost and time of making test benches and then verifying their functionality. Furthermore SoCs decrease design size and weight, power consumption, and the overall complexity of the system. Reconfiguration can improve the flexibility of the system. It is easy to upgrade and modify the existing system [24]. Moreover on-chip frequency scaling (power management technique that involves dynamically changing the frequency of a processor) is faster and the connections between modules are more reliable than off-chip frequency scaling.

## 4.3 SoC Design Challenges

SoC design continues to evolve fast but is still facing challenges. Systems designed on a single chip are mostly heterogeneous. Due to the design complexity and heterogeneity it is very difficult to make design space exploration quickly. Generally, the system designer has to optimize the system and make tradeoffs. The designer has to design the system in such a way that it is flexible enough to modify and within a short time [25][26]. Different approaches to face the design challenges are discussed below.

### **4.3.1 System Level Modeling**

The increase in the capacity of a single chip makes SoCs capable of having more functional blocks on a single chip and also increases the complexity and time-to-market of system design. System level modeling is one of the solutions that the SoC designers use to handle the design complexity as well as to perform fast design space exploration [26]. System level modeling was discussed in more detail in Chapter 2.

### **4.3.2 Block-Based Modeling**

Block-Based Modeling (BBM) is another approach to handle the design complexity and decrease time-to-market. The basic idea is to divide the system into smaller sub-systems. Once all the sub-systems are completed they will be integrated into a single system. Although BBM seems to be a very effective method, it still has some drawbacks such as dividing the system into multiple independent sub-systems and after completion of sub-system integrates into single system [22].

### **4.3.3 Platform-Based Design**

A platform is an abstraction that covers several lower level refinements. Platform Based Design (PBD) is a hierarchical design methodology. PBD is a combination of hardware platform and software platform. A hardware platform consists of micro-architecture of the system. A software platform consists of the software layer that is implemented on the top of the hardware platform for example Real-Time Operating Systems (RTOS), device drivers, and network communication subsystems. Issues from a software platform domain perspective, for a particular application processor must achieve the minimum speed criteria and system memory must achieve the minimum size criteria. From hardware platform perspective, architecture should achieve the flexibility criteria [33]. In PBD, high productivity is achieved through extensive, planned design reuse. Instead of developing a system from scratch, it can be modified or updated version of the existing system hardware or software [25] [26].

## **4.4 Communication Networks**

In both off and on-chip systems, communication is one of the main bottlenecks in the performance and optimization of the system. There are various methods used to achieve high speed and faultless communication and still the research is going on in this field [27].

The software part is the set of rules for communication, called a protocol, and it is known by all the devices that are taking part or want to take part in the communication. Communication protocols are used for data representation, error detection and correction, indicate initialization, and finishing the communication among the network. They are also used for data encryption and decryption in order to make sure that the data is received by the destination securely. Generally network protocols are divided into different types based on the OSI model layers. OSI is an industrial effort to standardize communication networks. OSI model is discussed more in detail in the following paragraph [34].

### **4.4.1 OSI Model**

The Open Systems Interconnection (OSI) model is developed by the International Standard Organization (ISO) in 1970. The objective of the OSI reference model is to identify and regulate the data among the communicating parties. The OSI model consists of seven different layers, each layer having its own unique role in communication. These seven layers are the application layer, presentation layer, session layer, transport layer, network layer, data link layer, and physical layer [27].

The application layer serves as an interface between the user and the communication system. It provides end-user service such as virtual terminal access and handling, electronic message services, file and job transfer among others. The responsibilities of the application layer include reliable (error free) data transfer, terminal independency, successful message delivery, and file management. The presentation layer (also known as the syntax layer) defines syntax (that is set of rules) and semantics for communication. This layer also provides security and increases reliability by providing for instance the data en- and decryption and data compression. The session layer is responsible for establishing connections between the communication nodes. Message delivery, fragmentation, and

reassembling frames and packets are the responsibilities of the transport layer. Selecting and establishing the path among the nodes for communication is done by the network layer. The data link layer extracts data from the physical layer and organizes it as frames. The responsibilities of this layer include error checking and correction, flow control, and acknowledgment of frames that are received by the destination node. Bit-level transmission is done by the physical layer. It defines electrical and physical specifications such as connector type, cable type, network adapters, voltages, and input/output pins [27].

#### **4.4.2 Network Components**

Primarily a network consists of two types of components: nodes and communication links. Nodes contain the routing algorithm and protocols. Communication links (or channels) transmit data from the source node to the destination node. Most of the time there are more than one path from the source node to the destination node but usually the network selects the shortest path, a path that contains minimum numbers of hops or nodes. This can be achieved by implementing efficient routing protocols. Single or multiple electronic devices can be connected to one node, but each device must have its own unique address. The network can differentiate between different devices based on this address [27].

The example of a communication network is shown in Figure 4.2. The network consists of five nodes. The connecting lines indicate the links among the nodes. Five devices are connected to the communication network. If device 1 wants to communicate with device 5, there are two different paths available that are via Node2, 3, 4 then 5 or Node3, 4 and 5. The efficient choice is to select the second path because it is shortest. The first path is used only if there is a problem in the second path such as buffer overflow, a broken link or data congestion.

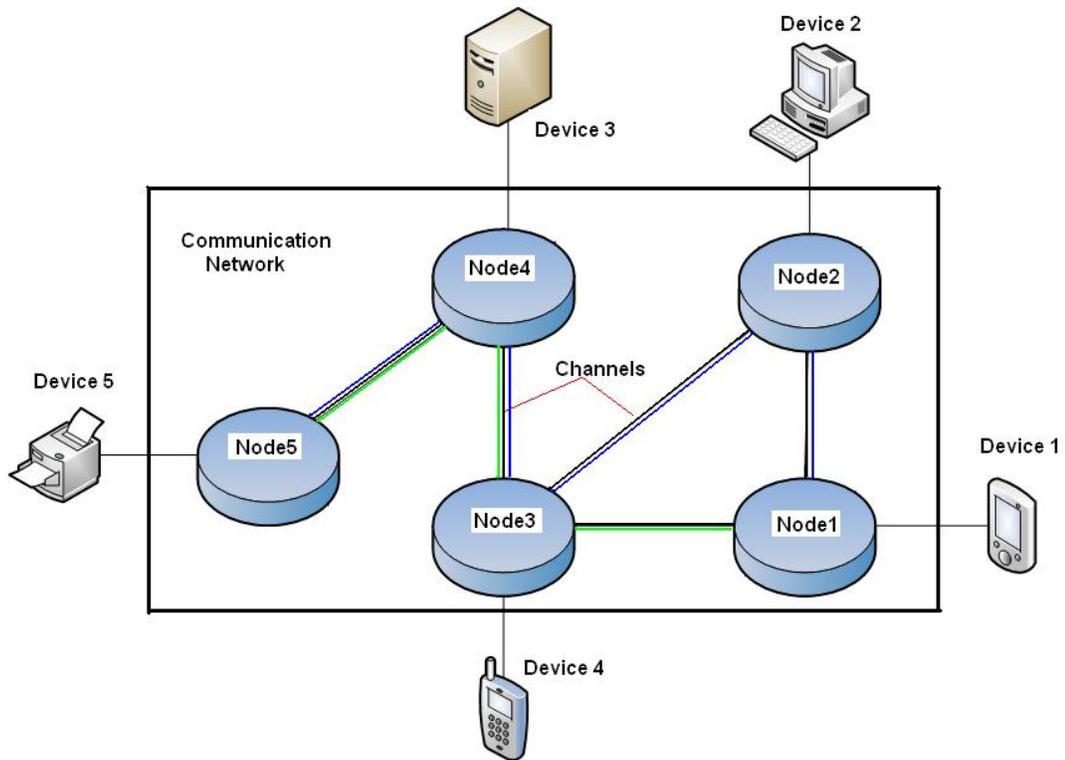


Figure 4-2 Model of Communication Network

### 4.4.3 Network Types

The categorization of networks depends on for instance the communication model implemented by the network, such as bus or star. Packet switching networks and circuit switching networks are two different networks based on the method of transferring data between the nodes [27].

In a shared-medium network, all devices connected to the network are sharing one common communication medium. Only one device is able to use the transmission line at a time. Direct and indirect network consist of the network interface block called a router. The router directly connects a node with its adjacent nodes and indirectly (through its neighbors) to all the nodes in the network. The direct and indirect networks are scalable thus used to build large networks [30].

## 4.5 Network-on-Chip

Due to the increase in number of processing and computing components on a single chip, the communication between them is the major performance bottleneck. Shared buses are overloaded with data requests which exceed their capacity of transferring data [28]. This causes major decrease in performance and poor resource utilization. Network-on-Chip (NoC) is the solution to many problems including the problems mentioned above. NoC is a dedicated subsystem that consists of physically optimized transmission channels and data packet routing algorithms for communication. Optimized transmission channel resolve physical problems such as power consumption, communication link reliability, and performance [29]. In addition to the benefits of NoC there are also several disadvantages such as chip area and transfer latency [3].

Communication topology and physical organization are specified by the NoC architecture. Interconnecting nodes are designed in such a way that they support parallel data packet transmission and minimum distance between the nodes. This increases the communication speed and reliability [30].

### 4.5.1 NoC Protocol

Network resources are managed by a network protocol. It consists of routing algorithms that dynamically manage the resources. The NoC protocols generally use five OSI layers, application layer, physical layer, data link layer, network layer, and transport layer, shown in Figure 4.3. The data link layer fetches data directly from the physical layer. The network layer establishes the connection between the communicating nodes. The transport layer receives data from the network layer in form of data packets. It extracts the actual message from the data packet. It is also responsible for assembling data received from the upper layer into the data packet and forwarding it to the network layer. The physical layer consists of interconnecting wires that connect the node to adjacent nodes [30].

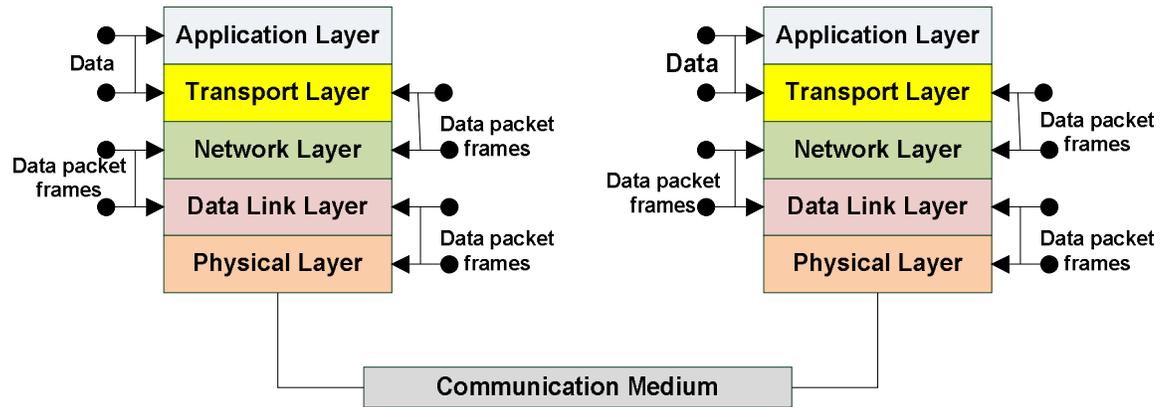


Figure 4.3 OSI Model for NoC [27]

# Chapter 5

---

## **Register-Transfer Level Model of a Network-on-Chip**

A hierarchical NoC was developed in the Department of Computer System at Tampere University of Technology (TUT). A multiprocessor system was built that uses packet switching split-transaction hierarchical NoC for global communication while leaving the local buses intact. The hierarchical NoC reduces traffic load on the local buses by sharing slave devices between multiple masters. The multiprocessor system has two instances of Core For FrEE (COFFEE) Reduce Instruction Set Computer (RISC) processor along with the MILK floating-point coprocessor that is attached through the coprocessor bus. The COFFEE RISC core was developed at Department of Computer System at TUT. The NoC contains shared memory that is attached through the target interface and a network node reserved for accelerator extension. The hierarchical NoC node serves as a bypass traffic only if the accelerator is not attached. This multiprocessor system is implemented on Altera Stratix II FPGA [35].

The communication between the nodes of the network is through interfaces. There are two types of interfaces in each node, initiator and target. The initiator initiates the transmission of data packets. Target interface responds to the outside requests. This kind of interface causes unnecessary waste of resources and slower operation. Incoming data is received through the local shared data bus using Direct Memory Access (DMA) mechanism. The outgoing data is transmitted to global NoC through a co-processor bus. A network node is responsible of converting the data into packets and reconvert. A FIFO buffer synchronizes NoC with a bus cluster. From the results it was proved that the resource utilization of the processor to NoC interface became higher than the resource utilization of the network node [35].

## 5.1 Synthesizable Switching Logic for NoC

Switching logic is the most important part of NoC. The data path is controlled by the arbitration logic. The goal is to achieve switch structure for the hierarchical NoC. The switch structure should be portable between different technology platforms and it must ensure the reusability without any significant area or performance penalties [35].

## 5.2 NoC Architecture

Figure 5.2 shows a hierarchical 3X3 mesh topology NoC. The NoC consists of nine nodes. The communication between the nodes is memory-mapped communication that supports single-cell packets. The NoC contains two levels of hierarchies. At the global level of hierarchy all local clusters are connected to the global network and they initiate data requests as well as respond to them. At local level of hierarchy the devices can act either as an initiator or receiver [35].

Each node consists of a bridge interface that enables the node to communicate in both directions (initiator as well as receiver). An input and output link pair is used to communicate with a global network switch. Communication requests originated from local nodes use the target side of the bridge and incoming communication requests are handled by the initiator side of the bridge. Each incoming data packet contains the address of the sender node. Sender address is stored by the initiator at the time of serving that particular data packet. After serving the data packet the stored address from an initiator is attached with the data packet response and initiator passes it to the target side of the bridge. The target side of the bridge interface generates routing information using routing tables for the data packet response and passes the data packet to a global network [35].

Global level switches are symmetric. The arbiters are shown in Figure 5.2. The global routing switch nodes make changes to the routing information and make it equivalent to the routing code expected at node input. The local switch consists of two parts that are request part and response part. Local switching scheme is designed to replace shared buses [35].

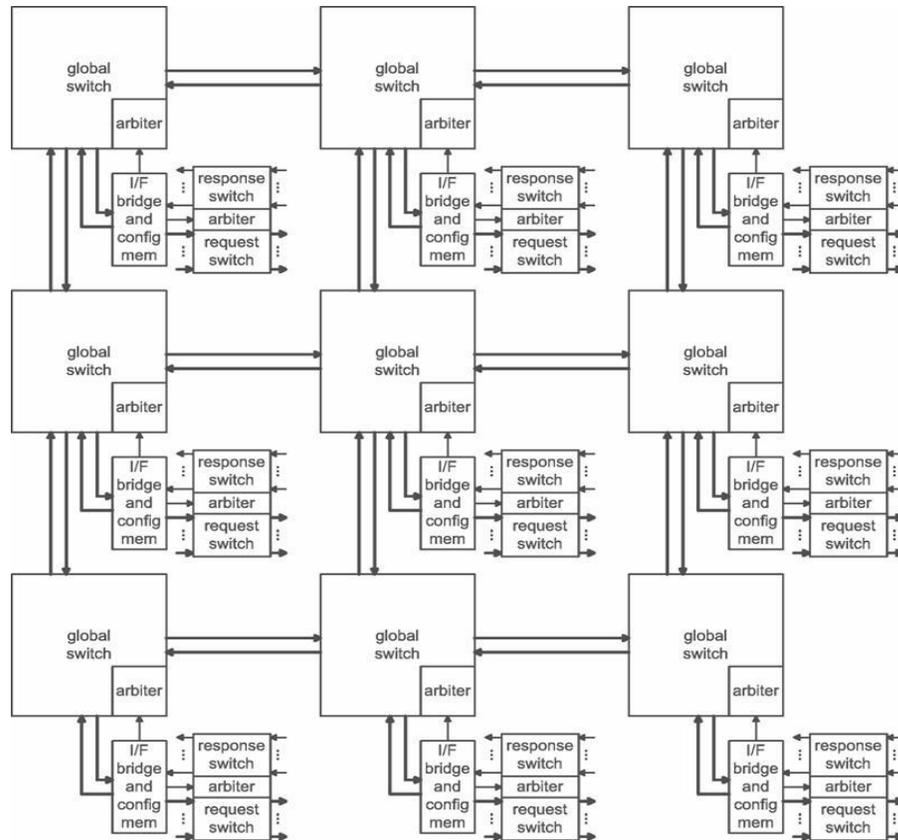


Figure 5.2 A hierarchical 3x3 mesh topology NoC [35] © IEEE

Arbitration scheme is implemented at the local as well as global level of hierarchy. The FIFO based data packet communication is used in the Round-robin fashion. In addition, the highest priority can be used for Time Division Multiple Access (TDMA) arbitration and lower priority configuration determines the sharing of remaining bandwidth [35].

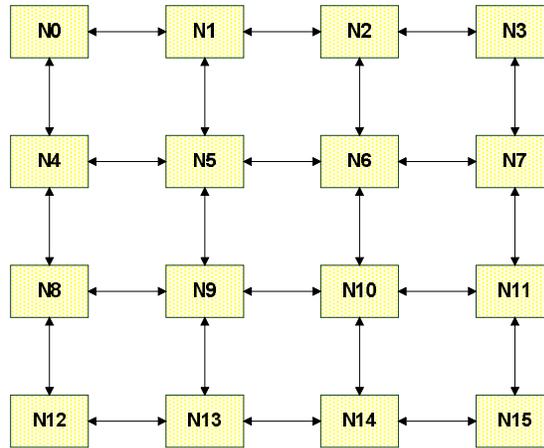
# Chapter 6

---

## **System-Level Model of a Network-on-Chip**

NoC design is a complex and time consuming process because of the increasing complexity of routing algorithms and at the same time designers need to deal with hardware hazards such as traffic congestion, clock cycle delay or data buffering. Hardware description languages are generally used to describe the system from hardware point of view while the routing algorithms are programmed in high level languages such as C++ or Java. It is necessary to create compatible tools. These tools add greater ease to the programmer by decreasing the error domain and thus speeding up the process of design space exploration. In this perspective, SystemC is the tool that is capable of capturing the system functionality, communication, software algorithms, and behavioral model of hardware at system level.

System level model of the NoC that is presented in this thesis is based on the hierarchical NoC architecture explained in Chapter 5. The NoC model is designed and implemented in SystemC. The main objective behind the implementation of the NoC model is to increase the flexibility of the network as well as design space exploration. Using the system level NoC model it is possible to simulate any network from 2x2 nodes to any number of nodes. The NoC model consists of a mechanism for transmitting data packets that is self-configurable according to the number nodes and network configuration. The connections between the nodes, size of data packets and communication mechanism resemble the low level hierarchical NoC. The number of clock cycles consumed by every single hop of data packet is same as in the low level hierarchical NoC. The advantage of using this system level NoC model is that it allows the programmer to completely focus on the software aspects of system that are algorithms and protocols instead of dealing with the hardware problems such as power consumption and resource management.



**Figure 6.1:** A 4X4 network

## 6.1 Implementation of the Network Architecture

The main goal of the system level NoC model is to speed up of design space exploration. Therefore, the implementation of the network architecture is flexible. The NoC model uses mesh topology. User defines the total number of nodes and the configuration of the network that is to be modeled. The NoC model divides the nodes into nine different groups according to their position in the network. At the first step, four corner nodes are selected and put in to the groups named South East, South West, North East, and North West. The positions of the nodes are assigned according to the size of the network. If user wants to create for example a network having 16 nodes and with 4x4 configuration (4 rows and 4 columns) as shown in figure 6.1 then the North West node has the position is (1,1) where the first number represents X-axis and the second number represent Y-axis in the network. North East node would be (4,1), South East node would be (4,4) and South West Node would be (1,4). After assigning the positions and respective groups to the corner nodes, next step is to assign positions and groups to the remaining nodes in the corner rows and columns. Besides the corner nodes in the first row, the remaining two nodes are put into North group and their position would be (2,1) and (3,1). The same would be the case with the last row, first column, and last column.

Table 6.1: Details of the each Node in a 4x4 network configuration.

Group	Position	Name	Adjacent Nodes	Output Buffers
North West	1,1	N0	N1,N4	2
North East	4,1	N3	N2,N7	2
South West	1,4	N12	N8,N13	2
South East	4,4	N15	N11,N14	2
North	2,1	N1	N0,N2,N5	3
North	3,1	N2	N1,N3,N6	3
West	1,2	N4	N0,N5,N8	3
West	1,3	N8	N4,N9,N12	3
East	4,2	N7	N3,N6,N11	3
East	4,3	N11	N7,N10,N15	3
South	2,4	N13	N9,N12,N14	3
South	3,4	N14	N10,N13,N15	3
Center	2,2	N5	N1,N4,N6,N9	4
Center	3,2	N6	N2,N5,N7,N10	4
Center	2,3	N9	N5,N8,N10,N13	4
Center	3,3	N10	N6,N9,N11,N14	4

After assigning all nodes in corner rows and corner column, in the next step all remaining nodes are put into the group of Center nodes and positions are assigned to them. In the Figure 6.1 N stands for node and the number represents logical position of the node in the network. The division of the nodes into the nine groups is based on the output and input ports each group has and the ability to communicate with their adjacent nodes. For example the node in the North West group always has two input and two output ports. This enables the node to communicate with only two of its adjacent nodes: one from the North group and the other from the West group. These four ports also limit the communication of the node to only two full duplex communication links. Two output buffers are added to the node in case there are more than two data packets to be sent from the node at the same clock cycle. The details of the network for the rest of the nodes are shown in Table 6.1 and Figure 6.2.

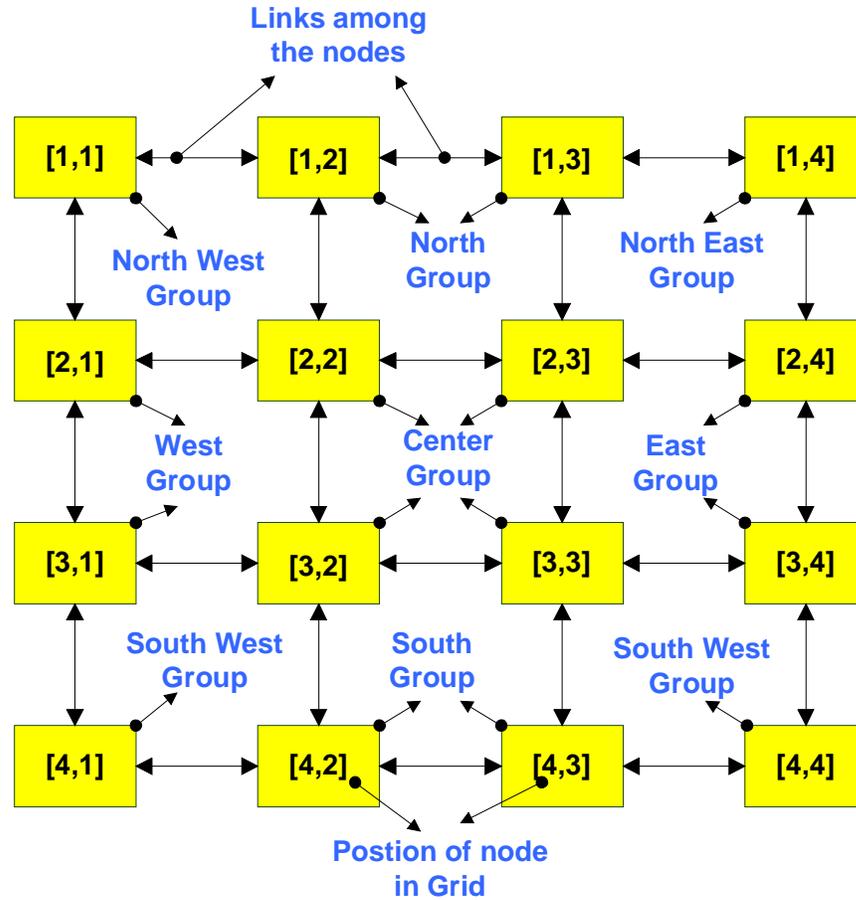


Figure 6.2: A 4x4 configured network according to node's position and group name

In Table 6.1, The group column represents the name of the group for each node. The position column represents the position of the node on horizontal and vertical axis in the network. Logical name is the name given to each node in the network that is used to calculate the logical data path. The adjacent nodes column shows the logical names of the nodes with which the respective node is able to communicate. The output buffers column shows the number of buffers each node contains. The details of the logical data path are explained in the next section.

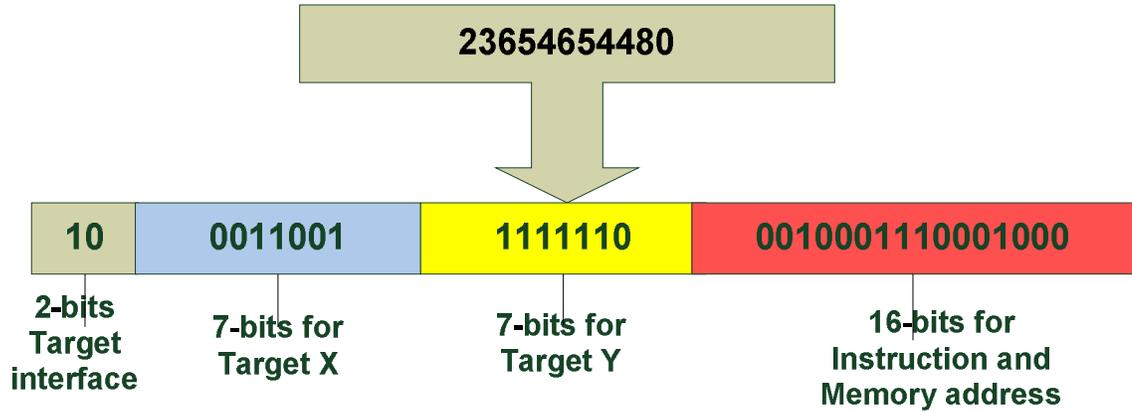


Figure 6.3: The 32-bits payload

## 6.2 Extract and Decode Data

The data to be sent is read from an input text file. The input file consists of the name of the source node and the destination node is encoded in the payload. The size of the payload is 32 bits. It consists of 14-bit for the destination node along with a 16-bit memory address and 2 bits to encode the target slave as shown in Figure 6.3. The logical name of the destination node is extracted from the payload by using the equation below.

$$\mathbf{DstNode} = [ \{ \mathbf{TnRows} - (\mathbf{Diff}) \} * (\mathbf{TrY} - 1) + (\mathbf{TrX} - 1) ]$$

DstNode is the logical name of the destination node. TnRows is the total number of rows in the network. Diff is calculated by subtracting the total number of columns from the total number of rows in the network. TrY is the decimal number of the 7-bit target Y coordinate and TrX is decimal of the 7-bit target X coordinate.

Logical path from the source node to the destination node is created using XY-routing algorithm. The data packet is the payload along with the routing information. The data packet is first sent along X-axis and when destination column is reached then data packet is sent along Y-axis until the destination node is reached. An example of two data packets that are sent in 4x4 network is shown in Figure 6.4. The source node for the first data packet is (1,4) and the destination node is (3,1). First, the packet takes three hops on X-axis until it reaches the destination column and after that it takes two hops on Y-axis to reach the destination row and destination node (3,1).



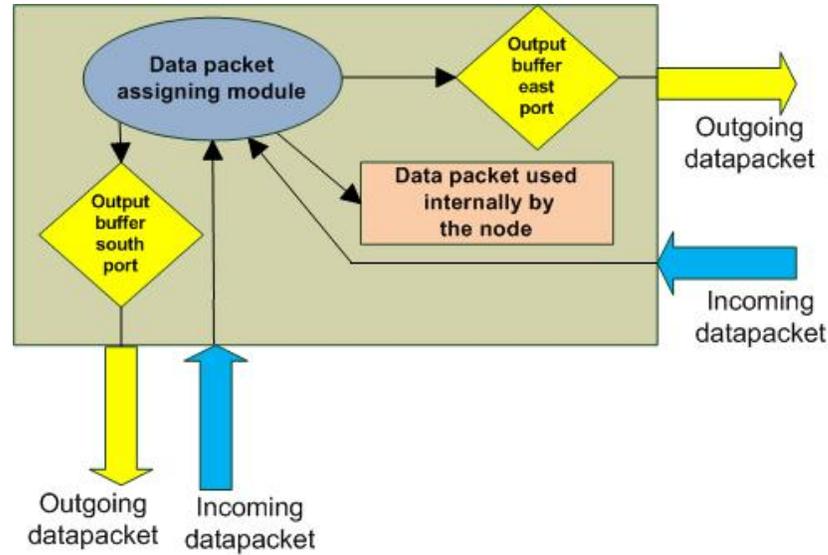


Figure 6.5: Internal architecture of North West node

### 6.3 Data Packet Routing

The data to be sent in the network is stored and copied from the text file. The sources and destinations for all the data packets are decoded and put into the buffers of the respective node in the network. After that, nodes transmit data in FIFO fashion. Each new data packet received at the input of a node is added to the buffer only if it is to be redirected. Nodes are able to transmit a single data packet at one clock cycle. There is one module in each node that checks the physical routing bit of the incoming data packet and takes action accordingly. This module either uses the data packet if it finds out that the particular node is the destination node or sends it to output buffer of the particular port. That particular port is responsible of correctly directing the data packet. The responsibilities of the node include sending, receiving and redirecting data packets. Each node knows about its position in the network, all of its adjacent nodes, and the ports that are attached to it.

Internal architecture of one of the corner nodes is shown in Figure 6.5. The big outside rectangle represents a node. It consists of two buffers that are illustrated with diamond shape boxes. Buffer is linked with output data that is shown with arrow directing outward from the node and the other side is attached with the data packet assigning module that is shown by the oval shaped circle.

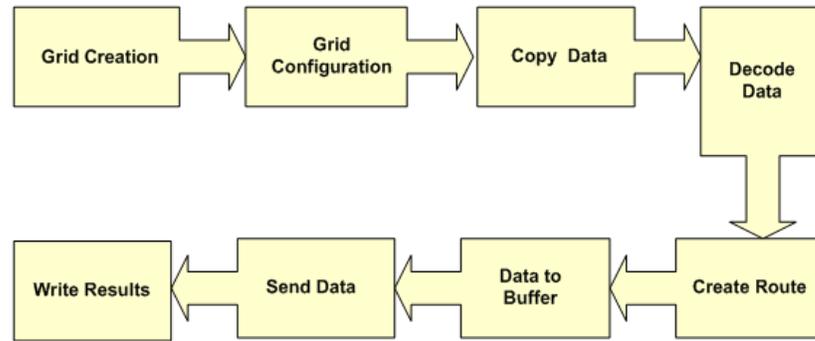


Figure 6.6: Functional Diagram of NoC System Level Model

Assigning module is responsible of receiving data packets from the input ports (shown by the arrows directing towards to node). After receiving a data packet, this module decides whether to send the data packet to output buffer or to use internally.

#### 6.4 Functional Diagram of NoC Model

Figure 6.6 shows the functional diagram of the system. First the user-defined network is created. Each node is configured by assigning ports and connecting them with ports of adjacent nodes.

Data from the input file is read by the “Copy Data” module. The destination nodes are decoded by “Decode Data” module. All the source nodes and destination nodes are verified by “Copy Data” module and “Decode Data” module. If the nodes are valid, the routes are created and data packets are sent to the output buffers of the source nodes. When all the data packets read from the input file are sent to the output buffers then transmission among the nodes starts. The data packets are transmitted from the output buffers in FIFO fashion. While transmitting data packet, each node is also receiving data packets from the adjacent nodes. The data packets received are either used by that node or redirected it to the respective output buffers. If the data packet is to be used by the node then the node just prints out on the monitor screen the name of the port data packet is received, name of the source node, name of destination node and the payload.

Table 6.2: Details of the simulation environment

Operating system	Window XP
Processor	Pentium 4 (3.4 GHz)
Ram	2 GB
Software to compile	Visual studio .net
Tool used	SystemC, C++

At the same time “Write Results” module writes to the output file the total number of clock cycles taken by the data packet, expected number of clock cycles (it is the number clock cycles that a data packet take to reach destination node without any delay), total number of hops it took to reach the destination, payload, logical name of the source node and logical name of the destination node.

## 6.5 Testing and Results

The NoC model is tested and results are evaluated under different conditions. Low level hierarchical NoC takes four clock cycles for a single hop. In order to imitate the functionality of low level NoC the same convention is used and each hop takes four clock cycles in system level NoC.

Table 6.2 shows the simulation environment. The operating system used for the simulation is windows XP. The hardware details of the computer are processor with clock speed of 3.4 GHz, Random access memory 2 GB, Hard disk memory space is 120 GB and equipped with Visual Studio .Net along with SystemC library. However, the NoC model can also work with Linux having C++ compiler and SystemC library.

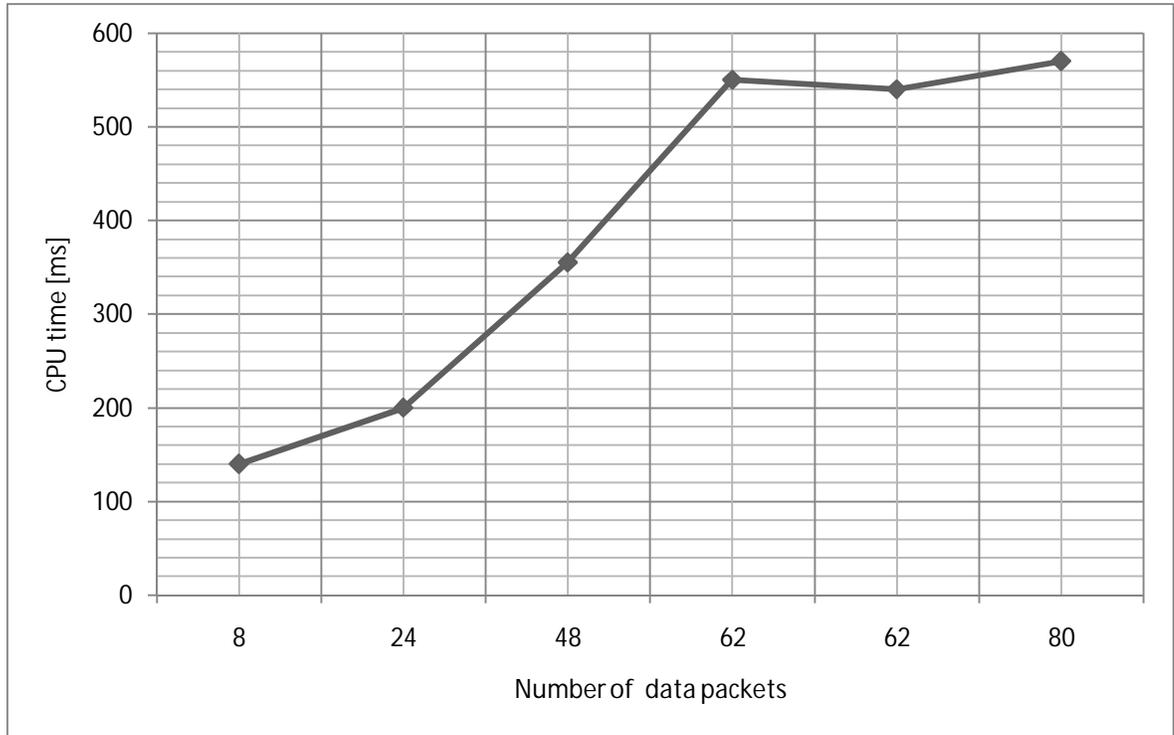


Figure 6.7: Time taken by the CPU to transmit data packets.

### 6.5.1 Testing All Buffers and Links

Six different network configurations are tested. The configurations are 2x2, 3x3, 4x4, 4x5, 5x4 and 5x5. In all analysis every buffer in each node has one outgoing data packet. If after four clock cycles each node receives a data packet on each and every input port then it means that the particular configuration of NoC model is working without any errors in communication channels or buffers.

In all tests, the main goal is to verify that every node, port and buffer in the network is tested and verify that it is working properly. In order to achieve this goal there are a specific number of data packets to be sent that depends on the number of nodes and their position in the network. Accordingly, the network configurations 2x2, 3x3, 4x4, 4x5, 5x4, and 5x5 are sending 8, 24, 48, 62, 62 and 80 data packets respectively. All the network configurations successfully transmitted and received the data packets in four clock cycles (after a single hop) as expected. However, the simulation time of the Central Processing Unit (CPU) varies almost linearly as the number of nodes in the network increases as

shown in Figure 6.7. This test also shows that the efficiency of the network model increases in terms of clock cycles because in all the network configurations it takes the same number of clock cycles but the number of data packets varies. In the above cases, the 5x5 network successfully transmitted eighty data packets while 2x2 network sends eight data packets in the number of clock cycles. The main difference is the time taken by CPU. The difference is in milliseconds but still it shows that more processing power and time is needed if the number of nodes in the model increases.

### **6.5.2 Worst Case Test with Minimum Number of Data Packets**

In this analysis, every port of the node in the network has to transmit three data packets. Two data packets are maximum distance data packets and one is to be sent to the nearest port of the adjacent node. The maximum distance data packet is a data packet that takes maximum number of hops to reach the destination node. The data packets are subjected to intentional delay by the transmission order of the data packet. First the maximum length data packet is sent then a single hop data packet and at the end again a maximum length data packet is sent from the same port. This analysis tests the functionality of the output buffers and the FIFO algorithm. In this case, there are several chances for loss of data packets due to the delay and buffering of data.

Figure 6.8 shows the worst case test for 4x4 configured NoC model. Total number of data packets sent in the network is 144. All the data packets successfully reached their destination nodes. 22.23% of the data packets reached in time without suffering from any delay due to buffering. The rest of the data packets were delayed. The smallest delay was 4 clock cycles and the largest delay is 60 clock cycles. All the data packets reached their destinations within 80 clock cycles.

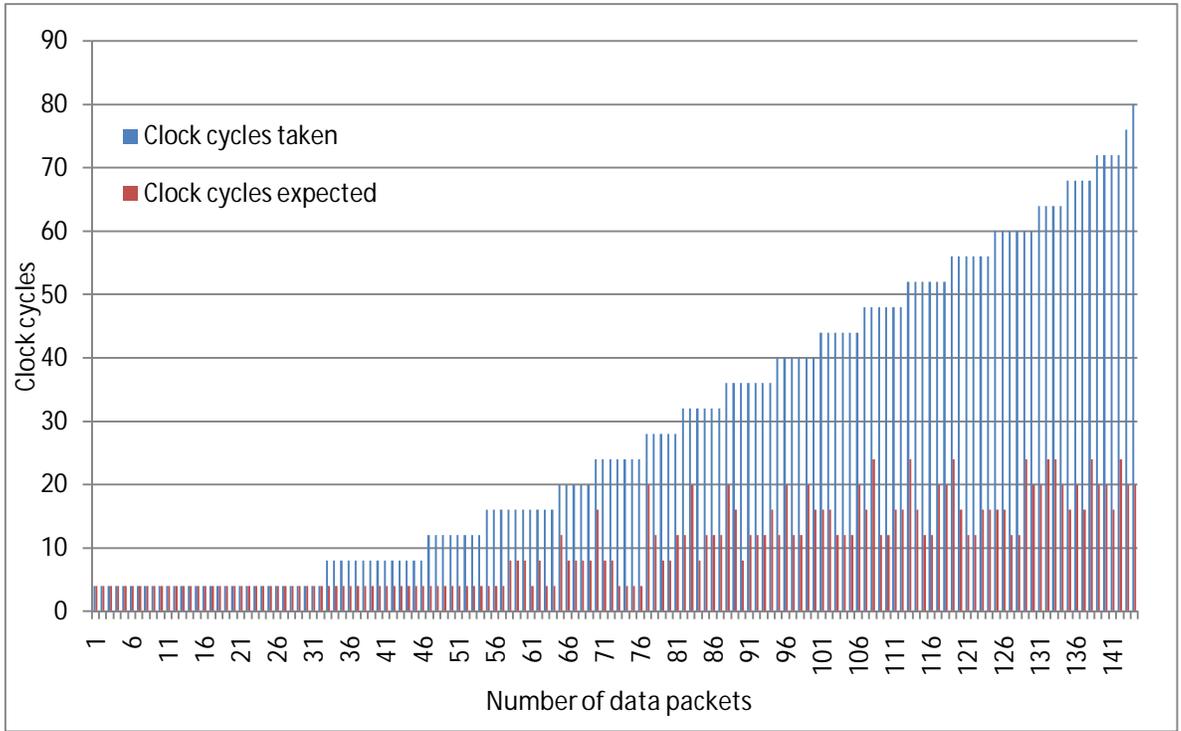


Figure 6.8: Worst case test for 4x4 Network.

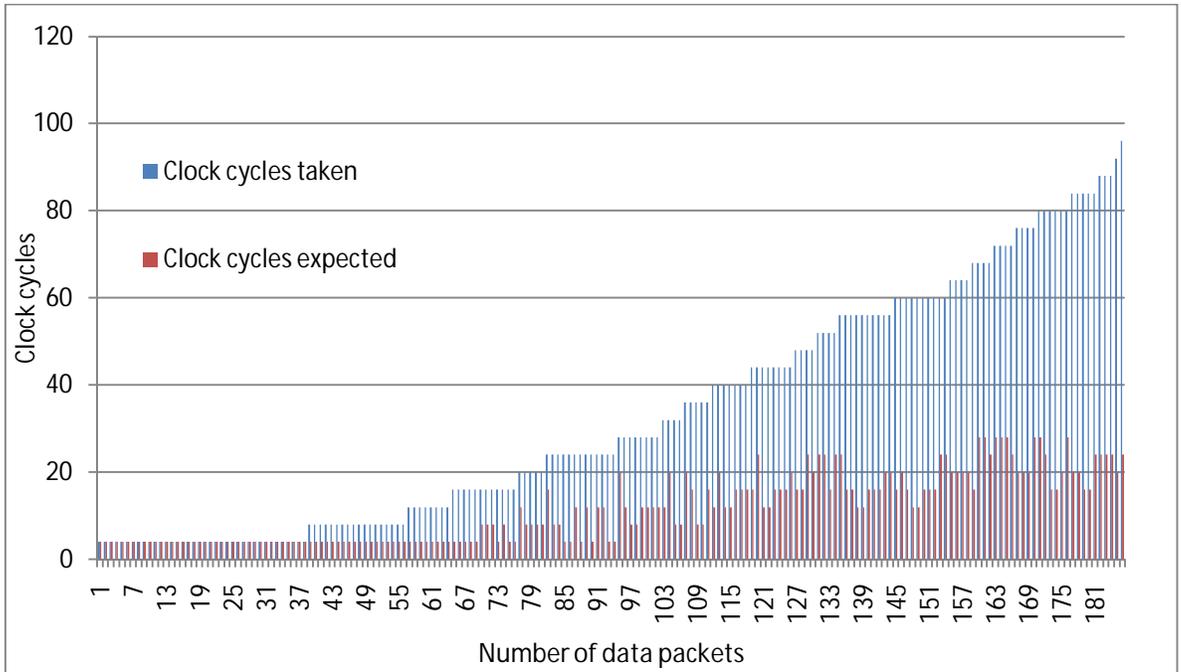


Figure 6.9: Worst case test 4x5 network.

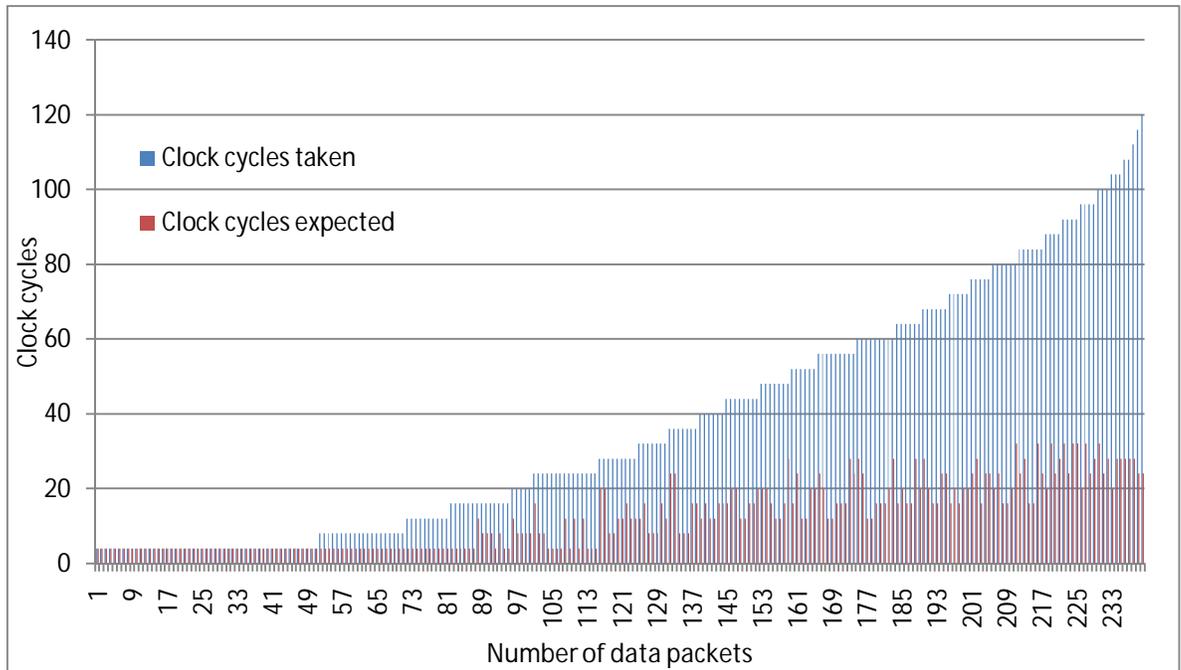


Figure 6.10: Worst case analysis 5x5 network.

The worst case analysis of the 4x5 configured NoC model is shown in Figure 6.9. 186 data packets were sent and all of them reached successfully their destination node. 20.9% of the data packets reached the destination without any additional delay. The smallest delay was 4 clock cycles and the largest delay is 72 clock cycles. The total time taken by the simulation was 96 clock cycles.

The worst case analysis of the 5x5 NoC model is shown Figure 6.10. 240 data packets were sent. All data packets reached their destinations within 120 clock cycles. 21.25 % of the data packets reached their destination within the expected number of clock cycles. The maximum delay for any packet in the network was 96 clock cycles and minimum was 4 clock cycles.

Comparing the three Figures, 6.8, 6.9, and 6.10 it is concluded that

- As the number of nodes and data packets in the network increases the maximum delay for the single data packet increases.

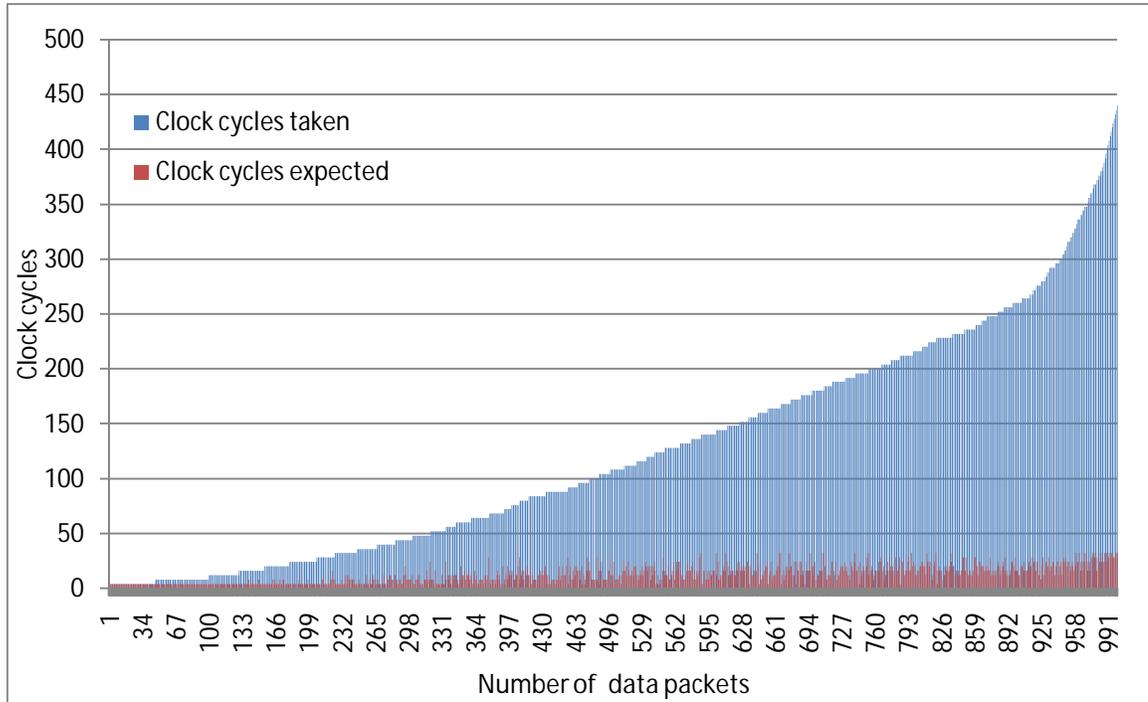


Figure 6.11: 1000 data packets in 5x5 network.

- Minimum delay for any data packet in any type of network configuration is always 4 clock cycles.
- The network having an equal number of nodes in rows and columns are more efficient because their percentage for the data packets that reached in expected number of clock cycles are better than the network having different number of nodes in rows and columns.

### 6.5.3 Network Analysis with one Thousand Data Packets

Figure 6.11, Figure 6.12 and Figure 6.13 show the results of transmitting 1000 data packets in network having configuration 5x5, 4x4, and 3x3 respectively. These 1000 data packets have random sources and destinations. All the data packets reached their destinations without any loss during communication. In 5x5 NoC, all the data packets reached their destinations within 440 clock cycles, in 4x4 and 3x3 NoC they reached within 448 clock cycles. Simulation time taken by the 3x3 is 7625 milliseconds, 4x4 is 9203 milliseconds,

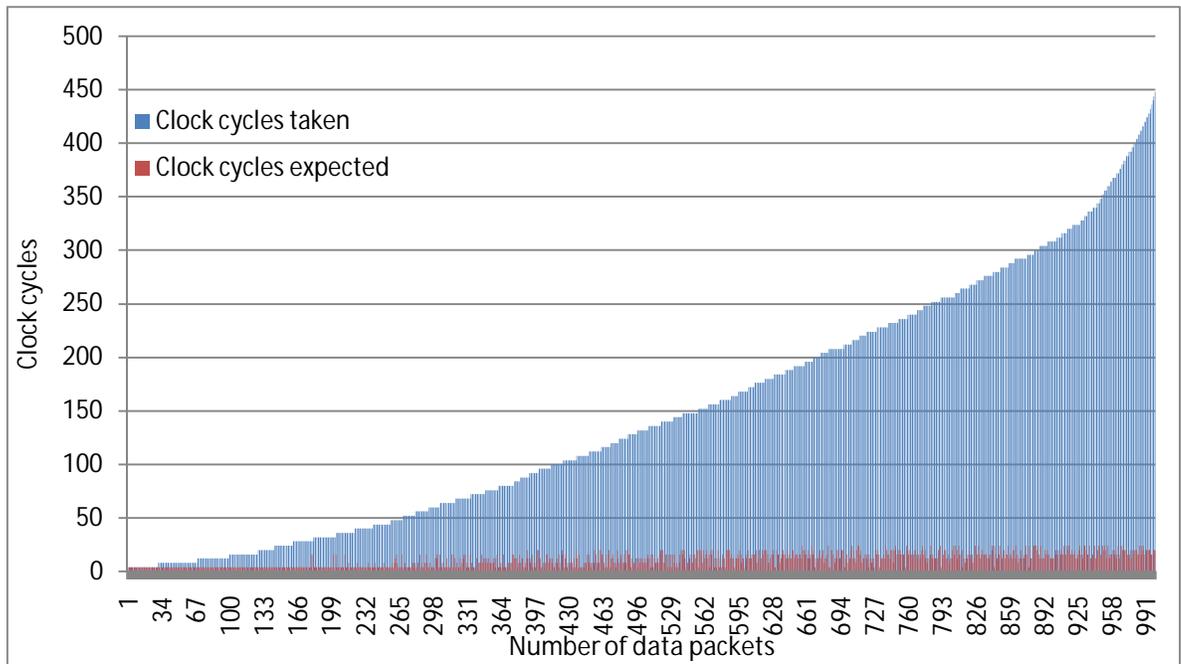


Figure 6.12: 1000 data packets in 4x4 network.

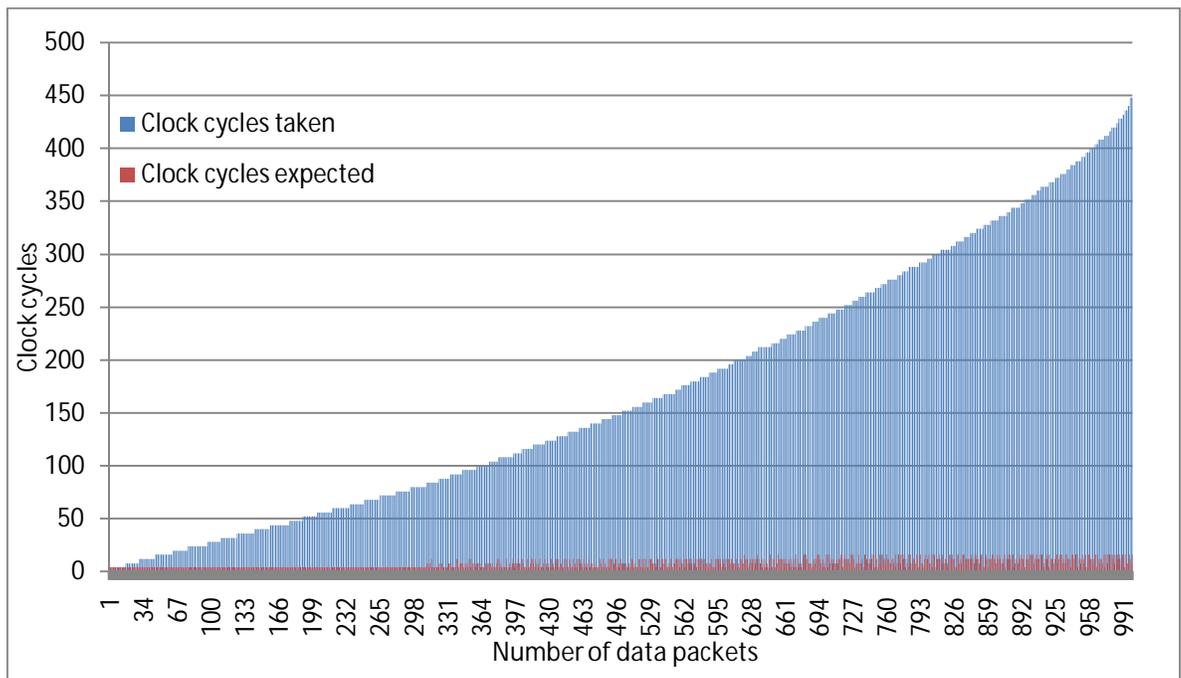


Figure 6.13: 1000 data packets in 3x3 network.

and 5x5 take 7969 milliseconds. As there are no criteria or pattern for the selection of sources and destinations therefore we do not predict the efficiency of one configuration over another. This test just showed that the NoC model is working properly at high traffic load with any configuration and number of nodes. At high level there is no packet loss. Maybe this model needs some modification when synthesizing the model on actual hardware (i.e. FPGA or ASIC).

# Chapter 7

---

## Conclusions and Future Work

System level model describes the behavior of the entire system using high level languages such as C++, Java, SystemC. System level modeling is evolving into a set of complementary methodologies that enable system designer to design, test, debug and verify the entire embedded system in a single platform. Due to increase in the number of computing and processing components on a single chip, the communication among them is a major performance bottle neck. NoC is one of the solutions to deal with this issue. NoC is a dedicated subsystem that consists of physically optimized transmission channels and data packet routing algorithms for communication. The main aim of this thesis was to construct an open source system level model of a NoC. The NoC model is used to speed up the process of design space exploration.

The created NoC model is used to analyze the behavior of the network with different number of nodes, network configuration, and traffic load. In this thesis several different tests were made in order to verify that under different situation the NoC model is working properly. The NoC model successfully completed all the tests. The NoC model is able to buffer, transmit, and receive data packets without any loss of data packets. The NoC model can be configured and re-configured. Several comparisons were also made between different NoC topologies with different traffic loads.

This NoC model may be used to test and compare the efficiency of different routing algorithms. The NoC model can be connected with a system level ISS together to form a system level model of Multi Processor System on Chip (MPSoC). The system level MPSoC formed may be used to analyze the behavior of Ninesilica. Ninesilica is a MPSoC that consists of nine COFFEE RISC core and mesh topology based hierarchical NoC. Ninesilica was developed at the Department of Computer System at TUT.

## REFERENCES

1. W. Wolf, "Multiprocessor Systems-on-Chips", in Proc. *IEEE Computer Society Annual Symposium on Emerging Very Large Scale Integration (VLSI) Technologies and Architectures*, March 2006, pp. 4.
2. K. Keutzer, S. Malik, R. Newton, J. Rabaey and A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform-Based Design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, 2000. pp. 1523 – 1543.
3. T. Kogel, R. Leupers, H. Meyr, "*Integrated System-Level Modeling of Network-on-Chip enabled Multi-Processor Platforms*", Springer, 2006.
4. T. Grötter, S. Liao, G. Martin, S. Swan, "*System Design with SystemC*", Springer, 2002.
5. I. Oussorov, W. Raab, U. Hachmann, A. Kravtsov, "Integration of Instruction Set Simulators into SystemC High Level Models", in Proc. *Euromicro Symposium on Digital System Design*, 2002, pp.126 - 129.
6. A system-level design community technical report available online <http://chipdesignmag.com/sld/>
7. M. Paul, P. Andrews, S. Cassidy, E. Thomas, "System-Level Modeling of a Network Switch SoC", in Proc. *15th International Symposium on System Synthesis, 2002*, pp. 62-67.
8. A. Pelkonen, K. Masselos, M. Cupak, "System-level Modeling of Dynamically Reconfigurable Hardware with SystemC", in Proc. *Parallel and Distributed Processing Symposium*, April 2003, pp. 8.

9. Lecture notes from Lund University Department of Computer Science available online <http://www.cs.lth.se/EDA380/>
10. P. Lieverse, P. Van der wolf, E. Deprettere, K. Vissers, "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems", in Proc. *IEEE Workshop on Signal Processing Systems (SiPS)*, 1999, pp. 181- 190.
11. Technical report by Open SystemC Initiative (OSCI). Available online <http://www.SystemC.org>
12. J. Bhasker "A *SystemC Premier*" Star Galaxy Pub, June 2002.
13. Website of System Verilog <http://www.systemverilog.org/>
14. D.D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, "*SpecC: Specification Language and Methodology*", Springer, March 2000.
15. Website of Rosetta <http://www.ittc.ku.edu/Projects/rosetta/>
16. S.E. Schulz, "An introduction to SLDL and Rosetta", in Proc. *Asia and South Pacific Design Automation Conference*, Jan 2000, pp. 571 – 572.
17. [http://www.mentor.com/products/esl/high\\_level\\_synthesis/](http://www.mentor.com/products/esl/high_level_synthesis/)
18. <http://www.forteds.com/BehavioralSynthesis/index.asp>
19. J. Cong, Y. Fan, G. Han, W. Jiang, Z. Zhang "Platform-Based Behavior-Level and System-Level Synthesis", in Proc. *SoC IEEE International Conference*, 2006, pp. 199-202.
20. C.B. David, J. Donovan, B. Bunton, A. Keist "*SystemC from the Ground up*" Springer, May 2004.
21. <http://www.asic-world.com/>
22. H. Chang, L.R. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, "*Surviving the SoC revolution: a guide to platform-based design*", Springer, November 1999.

23. R. K. Gupta, Y. Zorian “Introducing Core-Based System Design”, in Proc. *IEEE Design and Test of Computers*, vol. 14, no. 4, October 1997, pp. 15-25.
24. T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, P.Y.K. Cheung, “Reconfigurable Computing: Architectures and Design Methods”, *IET Proceedings on Computers and Digital Techniques* Volume 152, Number:2, 2005, pp. 193 – 207.
25. H. Chang, L. Todd, A. McNelly, G. Martin, M. Hunt, L. Cooke “*Surviving the SoC Revolution: A Guide to Platform Based Design*”, Springer. 1999.
26. W. Badawy, G. Jullien “*System-on-Chip for real-time applications*”, Kluwer Academic Pub. 2003.
27. Technical report by S. Hekmat title “*Communication Networks*” available online <http://www.pragsoft.com/books/CommNetwork.pdf>
28. B.J. Ho, L.S. Eun, B. Nader, “On Design and Analysis of a Feasible Network-on-Chip (NoC) Architecture”, in Proc. *International Conference on Information Technology*, April 2007, pp. 1033 – 1038.
29. W.J. Dally, B. Towles, "Route Packets, not Wires: On-Chip Interconnection Networks", in Proc. *Design Automation Conference*, 2001, pp. 684 – 689.
30. L. Benini, G. De Micheli, “Networks-on-Chips: A New SoC Paradigm” , in Proc. *Design, Automation and Test in Europe*, March 2002, pp. 70 – 78.
31. <http://www.mentor.com/>
32. D.D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner, “*Embedded System Design; Modeling, Synthesis and Verification*” Springer 2009.
33. K. Keutzer, S. Malik, A.R. Newton, J.M. Rabaey, A. Sangiovanni-Vincentelli, “System Level Design: Orthogonalization of Concerns and Platform-Based Design”, in Proc. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, December 2000, pp. 1523–1543.

34. Cisco Systems title "*Internetworking Technologies Handbook*" available at [www.cisco.com/en/US/docs/internetworking/technology/handbook/ito\\_doc.html](http://www.cisco.com/en/US/docs/internetworking/technology/handbook/ito_doc.html)
35. T. Ahonen, "*Designing Network-Based single-Chip system Architectures*", Ph.D. dissertation, Tampere University of Technology, 2006.
36. L. Charset, E.M. Aboulhamid, "*A VHDL/SystemC Comparison in Handling Design Reuse*" A report available at [www.iro.umontreal.ca/~chareslu/IWSOC2002.pdf](http://www.iro.umontreal.ca/~chareslu/IWSOC2002.pdf), University of Montreal, 2010.