Communication Centric Platforms for Future High Data Intensive Applications

Balal Ahmad

Thesis submitted for the degree of Doctor of Philosophy.

The University of Edinburgh

March 2009

<u>Declaration Of Originality</u>

I declare that this thesis is my original work except where stated

Student Name: Balal Ahmad

Signed: ______________________

Date: _________________________

ACKNOWLEDGMENTS

Pursuing a PhD has been a long and rather unique journey in my life. I would like to take this opportunity to express my gratitude to those who helped me through this journey.

The first person I would like to thank is my supervisor Professor Tughrul Arslan. I have been very fortunate to have Prof. Arslan as my supervisor. I am very grateful to him for giving me a chance to pursue my PhD in Edinburgh University. Without his valuable guidance and support this work would not have been possible. Many thanks Prof. Arslan for everything.

Having had a chance to work on SOCCAD project also brought me in contact with Dr Ali Ahmadinia. I have thoroughly enjoyed working with him. His technical advice and precise suggestions during my PhD are whole heartily appreciated.

I would also like to thank Dr. Ahmet Erdogan who always was available when I needed his advice. I extend a sincere thanks to all members of System Level Integration Group at Edinburgh University for their support and motivation.

I would especially like to thank my cousin Major Majid for his helpful suggestions throughout my University life. I won’t be where I am now without his helpful advice and support.

Living at Edinburgh has been a pleasant experience and would not have been possible without the moments I shared with Wahiba, Sadie, Moynaa, Mikhail, Alyas and Omar. Thanks for helping me keep my sanity.

Very special thanks to my parents and their prayers which have supported me throughout my life.

"Tell your heart that the fear of suffering is worse than the suffering itself. And that no heart has ever suffered when it goes in search of its dreams."

___*The Alchemist* by Paulo Coelho

Publications

Refereed Conferences

On Dynamically Reconfigurable Network on Chip

"Dynamically Reconfigurable NoC with Bus Based Interface for Ease of Integration and Reduced Design Time" – Balal Ahmad, Ali Ahmadinia, Tughrul Arslan. In Proceedings of IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2008), 22-25 June 2008. European Space Agency, Noordwijk, Netherlands. Pages: 309-314

"Architecture of a Dynamically Reconfigurable NoC for Adaptive Reconfigurable MPSoC" – Balal Ahmad, A. T. Erdogan, S. Khawam. In Proceedings of IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2006), 15-18 June 2006. Istanbul, Turkey. Pages: 405- 411.

"Dynamically Reconfigurable NoC for Reconfigurable MPSoC" - Balal Ahmad, Tughrul Arslan. In the proceedings of IEEE 2005 Custom Integrated Circuits (CICC 2005), 18-21 September 2005. San Jose, USA. Pages: 277-280.

On Communication Centric Platforms and systemC modelling

"SystemC-based Reconfigurable IP Modelling for System-on-Chip Design" – Ali Ahmadinia, Balal Ahmad, A T Erdogan, Ahmet T, Tughrul Arslan. In Proceedings of IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2008) ,22-25 June 2008. European Space Agency, Noordwijk, Netherlands. Pages: 362-367

"SystemC-based Custom Reconfigurable IP Cores for Wireless Applications" - Ali Ahmadinia, Balal Ahmad, Ahmet Erdogan, Tughrul Arslan. In the Proceedings of Engineering of Reconfigurable Systems and Algorithms (ERSA 2008), 14-17 July 2008. Las Vegas , USA.

"High-Level Power Estimation for Multi-Standard Wireless Systems."- Ali Ahmadinia, Balal Ahmad, Tughrul Arslan. In Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2008), 7-9 April, 2008. Montpellier, France, pp. 275-280.

"Communication Centric Modelling of System on Chip Devices Targeting Multi-Standard Telecommunication Applications" - Ali Ahmadinia, Balal Ahmad, and Tughrul Arslan. In Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 7-9 April, 2008. Montpellier, France, pp. 209-214.

"Communication Centric Platform Based Designs for Future MPSoC" – Balal Ahmad, Ali Ahmadinia, Tughrul Arslan. 6th International Bhurban Conference on Applied Sciences and Technology, 2008. January 2009, CESAT, Pakistan.

"Hybrid Communication Media for Adaptive SoC Architectures" - Balal Ahmad, Ali Ahmadinia, and Tughrul Arslan. In Proceedings of NASA/ESA Conference on

Adaptive Hardware and Systems (AHS 2007), 5-8 August 2007. Edinburgh, U. K., pp. 373-378,.

“System Level Reconfigurable FFT Architecture for System-on-Chip Design.” - Ali Ahmadinia, Balal Ahmad, and Tughrul Arslan. In Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), 5-8 August 2007. Edinburgh, U. K., pp. 169-175.

“System-level Modelling and Analysis of Embedded Reconfigurable Cores for Wireless Systems.” – Ali Ahmadnia, Balal Ahmad, Tughrul Arslan. - In Proceedings of International Conference on Field-Programmable Logic and Applications (FPL 2007). 27-29 August 2007. Amsterdam, Netherlands, pp. 757-760.

Abstract

The notion of platform based design is considered as a viable solution to boost the design productivity by favouring reuse design methodology. With the scaling down of device feature size and scaling up of design complexity, throughput limitations, signal integrity and signal latency are becoming a bottleneck in future communication centric System-on-Chip (SoC) design. This has given birth to communication centric platform based designs.

Development of heterogeneous multi-core architectures has caused the on-chip communication medium tailored for a specific application domain to deal with multi-domain traffic patterns. This makes the current application specific communication centric platforms unsuitable for future SoC architectures.

The work presented in this thesis, endeavours to explore the current communication media to establish the expectations from future on-chip interconnects. A novel communication centric platform based design flow is proposed, which consists of four communication centric platforms that are based on shared global bus, hierarchical bus, crossbars and a novel hybrid communication medium. Developed with a smart platform controller, the platforms support Open Core Protocol (OCP) socket standard, allowing cores to integrate in a plug and play fashion without the need to reprogram the pre-verified platforms. This drastically reduces the design time of SoC architectures. Each communication centric platform has different throughput, area and power characteristics, thus, depending on the design constraints, processing cores can be integrated to the most appropriate communication platform to realise the desired SoC architecture.

A novel hybrid communication medium is also developed in this thesis, which combines the advantages of two different types of communication media in a single SoC architecture. The hybrid communication medium consists of crossbar matrix and shared bus medium . Simulation results and implementation of WiMAX receiver as a real-life example shows a 65% increase in data throughput than shared bus based communication medium, 13% decrease in area and 11% decrease in power than crossbar based communication medium.

In order to automate the generation of SoC architectures with optimised communication architectures, a tool called SOCCAD (SoC Communication architecture development) is developed. Components needed for the realisation of the given application can be selected from the tool’s in-built library. Offering an optimised communication centric placement, the tool generates the complete SystemC code for the system with different interconnect architectures, along with its power and area characteristics. The generated SystemC code can be used for quick simulation and coupled with efficient test benches can be used for quick verification.

Network-on-Chip (NoC) is considered as a solution to the communication bottleneck in future SoC architectures with data throughput requirements of over 10GB/s. It aims to provide low power, efficient link utilisation, reduced data contention and reduced area on silicon. Current on-chip networks, developed with fixed architectural parameters, do not utilise the available resources efficiently. To increase this efficiency, a novel dynamically reconfigurable NoC (drNoC) is developed in this thesis. The proposed drNoC reconfigures itself in terms of switching, routing and packet size with the changing communication requirements of the system at run time, thus utilising the maximum available channel bandwidth. In order to increase the applicability of drNoC, the network interface is designed to support OCP socket standard. This makes drNoC a highly re-useable communication framework, qualifying it as a communication centric platform for high data intensive SoC architectures. Simulation results show a 32% increase in data throughput and 22-35% decrease in network delay when compared with a traditional NoC with fixed parameters.

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

APB	Advanced Peripheral Bus
ASB	Advanced System Bus
ASIC	Application Specific Integrated Circuit
CPU	Central Processing Unit
CTG	Communication Task Graph
DCR	Device Control Register
DMA	Direct Memory Controller
DRAM	Dynamic Random Access Memory
drNoC	Dynamically Reconfigurable NoC
EDA	Electronic Design Automation
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GALS	Globally Asynchronous Locally Synchronous
GPIO	General Purpose Input Output
HW	Hardware
NoC	Network-on-Chip
OCP	Open Core Protocol
OPB	On-Chip Peripheral Bus
PCB	Printed Circuit Board
PIO	Parallel Input Output
PLB	Processor Local Bus
QoS	Quality of Service
SJF	Shortest Job First
SoC	System-on-Chip
SW	Software
TDMA	Division Multiple Access
TLM	Transaction Level Modelling
UART	Universal Asynchronous Receiver/Transmitter
VC	Virtual Component
VLIW	Very Long Instruction Word
VSIA	Virtual Socket Interface Alliance

Chapter 1
INTRODUCTION

1.1 Motivation

According to International Technology Roadmap for Semiconductors, by the end of this decade, System-on-Chips (SoCs), using 50-nm will grow to 4 billion transistors running at 10GHz [ITR–001]. Such growth will be driven by design methodologies supporting component reuse in plug and play fashion. Most quality of service (QoS) matrices will revolve around performance and reliability measures and on-chip interconnections will be the limiting factor for performance and energy consumption [BEN-001].

In today’s electronic industry, time to market is one factor that greatly determines the success of the developed product. Re-use design methodologies and ease of integration are seen as the way forward to reduce this time to market constraint. Reconfiguration has not only emerged as an efficient way of development of low power architectures for multi-standard applications but also plays an important part in the reuse design paradigm.

The definition of a "system" in "system on a chip" has expanded to cover multiple processors, embedded DRAM, flash memory, application specific hardware accelerators and (Radio Frequency) RF components. This has created a situation where there is a mixture of traffic types utilising the on-chip communication media. The communication network that was traditionally handcrafted to a particular traffic type and constraints has to cope with additional burden now and proves to be a limiting factor in achieving the required performance from the present SoC. This is shifting the design paradigm from computation centric to communication centric design flow.

Due to escalating costs associated with design, verification, manufacture and testing of deep sub-micron chips, it is becoming economically infeasible to build highly customised application specific systems. The notion of platform based design is seen as a way to tackle this problem. Platform refers to systems consisting of largely pre-designed and verified hardware and software components that can be targeted towards multiple applications, in order to amortise the high cost of platform development over larger markets. Different platform based designs have emerged in the last few years, however, most of the commercially developed platforms support companies' own products and thus, do not provide the facility to easily integrate third party components into the pre-verified platform.

System modelling and electronic design automation (EDA) tools go hand in hand in today's semiconductor industry. With Multi-core SoC being a major part of electronics used in safety critical systems, quick system simulation for verification purposes poses a new challenge for the system designers. Current application specific integrated circuit (ASIC) or conventional field programmable gate array (FPGA) centric design methodologies are unable to cope with the development requirement of present day sophisticated SoC architectures. Especially with the added reconfigurable component integration in system, there is an urgent need for design methodologies and tools that are able to deal with different types of reconfigurable SoC fabrics targeting future reconfigurable SoC architectures.

1.2 Author's contribution

The aim of this thesis is to propose an on-chip communication framework for future high throughput intensive applications and to develop a design methodology for rapid SoC development. The work presented in this thesis targets three crucial areas of SoC design methodology: development of communication media for future heterogeneous multi-core systems, development of communication centric platforms for plug and play integration of components (reconfigurable, fixed and processor based) to facilitate ease of design reuse, and thirdly, development of a tool to automate the generation of SoC architectures with optimised communication media, focussing on custom reconfigurable cores.

Work carried out in the field of on-chip communication varies from bus based systems to complex Network-on-Chip (NoC) architectures. Along with the advantages of a certain communication medium are its drawbacks; no work is done where two different communication structures with different power and throughput characteristics co-exist in single SoC. In this thesis, a novel hybrid communication medium is presented that combines the advantages of a bus based communication system with a crossbar based communication system and thus both the bus and crossbar co-exist in the same SoC [AHM-001].

NoC is considered to be a solution to the communication demands of future multi-core systems. NoC is generated by choosing a network topology, one of the routing and switching schemes and finding an optimal packet size. However, along with the advantages of any communication parameter in NoC, follows its disadvantages and a compromise has to be reached when choosing these network parameters. This thesis presents a dynamically reconfigurable NoC (drNoC) that eliminates this compromised choice of network parameters and dynamically chooses the best parameter in terms of routing, switching and packet size for the optimised network performance. The proposed drNoC incorporates an Open Core Protocol (OCP) based interface allowing components to be integrated in a plug and play fashion [AHM-002] [AHM-003] [AHM-004].

As mentioned above, platform based design has emerged as an efficient way for rapid development of SoC architectures. This thesis also presents a novel communication centric platform based design flow for future SoC architectures. The proposed platforms have built in controllers, thus making the heterogeneous cores (reconfigurable, fixed, processor based) integrated in a plug and play fashion. This eliminates the need of the dedicated system controller to be integrated or programmed when integrating processing cores to the platform. Four platforms have been developed with different communication media including the hybrid concept mentioned above. Unlike traditional platform based designs, the novelty of the proposed platform lie in its ability to produce SoC architecture in truly plug and play fashion with an optimised communication media [AHM-005].

Lastly, bringing the above mentioned work together, a tool called SOCCAD is developed to automate the development of SoC architectures. The novelty of the SOCCAD tool lies in its ability to automate development of SoC architectures integrating custom reconfigurable cores and with optimised communication media. SOCCAD tool has a built-in component library that not only holds SystemC models of communication media but also includes custom reconfigurable cores, processor cores and other peripheral components required to generate complete system architecture for quick simulation and verification. [AHM-006] [AHM-007] [AHM-008] [AHM-009] [AHM-010] [AHM-011].

1.3 Roadmap of thesis

This remaining of this thesis is divided into six chapters and is organised as follows:

- **Chapter 2** gives an overview of traditional communication media and communication centric platforms. It introduces the concept of shared communication media and gives an overview of commercially available shared and hierarchical buses. The communication centric platform based approach is then described and performance matrices related to on-chip communication media is listed to conclude the chapter.

- **Chapter 3** details the concepts of NoC communication and how it tackles the drawbacks in traditional bus based communication media. This chapter also identifies the expectations from future packet based on-chip communication network and lists performance matrices relating to NoC.

- **Chapter 4** presents two important things, firstly, a hybrid communication medium is presented and secondly the proposed communication centric platforms have been discussed. For the purpose of effective on-chip communication, four platforms have been developed. This chapter also details the systemC based communication centric modelling of the proposed platforms and discusses their simulation results. The developed platforms are also evaluated with the help of real life example.

- **Chapter 5** presents the developed SOCCAD tool, highlights the communication centric placement and explains the automated generation of communication centric SoC architectures utilising the developed communication centric platforms.

- **Chapter 6** presents drNoC, which is developed for SoC architectures with communication requirements of over 10GB/s and explains its architecture and implementation. Simulation results are then listed to demonstrate the effectiveness of proposed drNoC in high data throughput applications.

- **Chapter 7** presents a summary of all the previous chapters reiterating the main contributions by the author. It also identifies topics for future research and development.

1.4 Summary

On-chip communication is a crucial research area and with the SoC design methodology shifting to communication centric design flow, there is a need for development of new communication architectures to cater for the increasing throughput requirements while providing the required QoS in a reduced design time. This thesis presents two novel communication media, a hybrid communication architecture and a dynamically reconfigurable NoC for future high data intensive multi-core architectures. This thesis also presents novel communication centric platform based approach and a tool for automated generation of SoC architectures that aims at rapid development of SoC architectures. The next chapter gives an overview of traditional communication media and commercially developed communication centric platforms.

Chapter 2

LITERATURE REVIEW

Communication Centric Platform Based Designs

2.1 Introduction

This chapter gives an insight of communication centric platform based design. It starts with the definition of term platform and the origin of platform based design. Current on-chip communication media are discussed followed by literature review of communication centric platforms.

During recent years, platform based design, also known as the configure and execute approach is emerging as a powerful SoC design methodology for rapid development of SoC devices. The term “platform” is defined differently by researchers, semiconductor industries and tool vendors.

Sangiovanni-Vincentelli [SAN-001] defines an SoC platform as a "layer of abstraction with two views." The upper view allows an application to be developed without referring to the lower levels of abstraction. Meanwhile, the lower view is a set of rules that classify a set of components belonging to the platform.

The Virtual Socket Interface Alliance's (VSIA) [VIS-001] platform-based design development group defines an SoC platform as "a library of virtual components and an architectural framework, consisting of a set of integrated and pre-qualified software and hardware virtual components (VCs), models, EDA and software tools, libraries and methodology, to support rapid product development through architectural exploration, integration and verification."

Automotive industry pioneered the platform concept in the early 1980s by standardising a common platform across several makes and models. By developing several platforms and sub-platforms, for example, the chassis platform, the interior platform, and the electrical platform etc., the manufacturer was able to outsource significant segments of automotive hardware, thus reducing production cost and design cycle and improving design quality.

The semiconductor industry embraced the concept of platform based design to cope with the rising pressure of time to market, design and manufacture costs. Platform based designs have been classified into different kinds.

The first kind is called a “full application platform” in which users design full applications on top of hardware and software architectures. Such platforms usually are accompanied by software to support compilation, debugging, simulation and emulation. This speeds up the time to market and gives the designer an advantage of having a working chip, running at real speeds and executing in real environment.

Some Industrial examples of full application platform include Philip’s Nexperia including a 32 bit MIPS RISC CPU, a 32 bit VLIW Trimedia processor and three levels of internal buses [PHI-001], Texas Instrument’s OMAP multimedia platform including a DSP, an enhanced ARM processor and an inter-processor communication mechanism [TEX -001], and ARM’s PrimeXsys wireless platform including ARM 926EJ-S processor core, java acceleration, a multi-layer AMBA bus and a selection of peripherals [ARM-002].

The second type of platform, which is not very common in academic based research, is a “processor centric platform”. As the name suggests, in processor based platforms only the processor is reconfigurable but it does not make the whole system. Industrial examples include Improv Systems [IMP-001], ARC [ARC-001] and Tensilica's Xtensa [TEN-001].

The third type is a “communication centric platform” which provides an interconnect architecture but does not provide a processor or a full application. Example includes IBM CoreConnect [IBM-001], ARM’s AMBA [ARM-001] Sonic’s silicon backplane [SON-001] and Palmchip’s CoreFrame architecture [PAL-001].

Finally there is a “fully programmable platform” consisting typically of FPGA logic and a processor core. Examples from industry are Altera’s Excalibur [ALT-001], Xilinx Virtex Pro [XIL-001] and Quick Logic’s PolarPro [QUI-001].

Due to the importance of communication in the future SoC designs, this chapter focuses on the communication centric platform based designs. In the following sections, the basics of on-chip communication is discussed followed by a literature review of traditional SoC communication including bus based, crossbars, hierarchical bus and finally it moves towards the development of communication centric platforms.

2.2 Basics of On-chip Communication

As mentioned earlier, performance of a multi-core SoC is not only determined by the capacity of the processing elements (e.g. CPU speed, cache size, etc.), but it is also limited by the interconnect network. Design and optimization of such interconnect network are critical for system performance. In designing communication architecture for SoC, certain factors have to be considered [LAH-001]:

Appropriate Topology: The topology refers to the way SoC components are connected. It can be in the form of single shared architecture, dedicated communication channels or more complex architectures such as hierarchical buses, token ring or crossbars.

Communication Protocols: The protocol specifies the manner in which communication across the channel takes place. Communication protocols deal with the different types of resource management algorithms used for determination of access right to the shared communication channels. Static-priority, time division multiple access (TDMA), token passing, lottery and code division multiple access (CDMA) are some of the existing communication protocols employed in on-chip communication.

Architectural Parameters: Parameters like bus widths, burst transfer size, priorities etc are also to be defined for the communication channels and associated protocols. Performance of communication media depends highly on these parameters.

Clocking: Clocking can be synchronous or asynchronous. If a single clock is used for the communication medium and its connected cores, the system is referred to as a synchronous system. Asynchronous system in contrast, has no global clock and timing is managed locally. Communication medium synchronisation occurs with the help of handshaking protocol that uses request-acknowledgement signals to ensure that data transfer is completed successfully. Handshaking is also used in synchronous systems for a data transaction consisting of several data transfers. To favour low power, globally asynchronous, locally synchronous approach is used [BAI-001].

Asynchronous buses are typically slower than synchronous buses because of the additional overhead of the handshaking protocol. However, in high data throughput devices, the delays introduced by handshaking are negligible. When slower devices take part in transfers, a wait signal is used to stretch the transfer for several clock cycles. Split transfer is used in slower devices where a read operation is split into two read operations and the bus is released in the middle. Thus, bus is not reserved for the whole operation and can be utilised by other devices for communication. Pipelined bus transfers are also used to obtain higher clock frequencies. In pipelined transfer, address is send on the first cycle for it to be decoded following by the data on following cycles. It is also possible to have the data of last transfer interleaved with the address of next transfer.

Interconnect interfacing: For rapid creation and integration of interoperable Virtual components (VCs), different interfacing standards have been developed for industry. OCP and VCI are the two most recognised standards in the semiconductor industry and research community.

OCP: OCP [OCP-001] is a complete socket standard that facilitates component reusability by providing a configurable interface to on-chip communication sub system. A socket is universal and is targeted for use in virtually any application, while an interface is targeted at a single unique application, where all of the arbitration logic and interface circuitry is defined for that particular application.

SoC designer can select signals from the OCP configurations needed for data, control and test requirements. Defining a core interface using the OCP provides a complete description for system integration. Basic OCP includes only data flow signals and is based on simple request and acknowledge protocol. However, the optional extensions support more functionality in control, verification and testing. Beside the basic OCP version, there are four extensions: simple extension, complex extension, sideband extension and debug and test interface extension. Simple extension and complex extension support burst transactions and pipelined write operations. Sideband extension supports user-defined signals and asynchronous reset.

VCI: The Virtual Component Interface (VCI) [VIS-001] specifies a request-response protocol for on-chip communication. As an interface, the VCI can be used as a point-to-point connection between two units called the initiator and the target (Master and Slave in case of OCP), where the initiator issues a request and the target responds. VCI defines this protocol for the transfer of requests, responses, contents and for coding of these requests and responses.

VCI has three complexity levels: peripheral VCI (PVCI), basic VCI (BVCI), and advanced VCI (AVCI). PVCI provides a simple interface for applications that do not need all the functionality of BVCI. Two main signals are defined in PVCI for handshaking: VAL (signal sent by initiator to inform the presence of valid values in its interface) and ACK (target response signal indicating the end of successful communication).

BVCI has a powerful set of rules. Communication in BVCI interface happens between the Initiator and the target. Contents are transferred separately under the control of a handshake protocol i.e. the request and response messages are completely independent. The use of two communication channels is defined in the BVCI standard.

The AVCI is a superset of the BVCI and adds more sophisticated features such as threads to support high-performance applications. In the AVCI, requests may be tagged with identifiers, which allow such requests and request threads to be interleaved and responses to arrive in a different order.

In a paper published by Porto et al. [POR-001] comparing PVCI, BVCI and OCP interfaces, it is concluded that BVCI results in the highest area utilisation, while PVCI interfaces occupies the lowest area. OCP interface area lies in between the PVCI and BVCI. When analysing the operation frequency, PVCI and OCP exhibit very similar results, however, due to a complex communication protocol, BVCI interface has a lower performance. Thus, for low area and high performance, OCP interface is considered as a better option.

2.3 Bus Based On-Chip Communication

Buses have been deployed for communication since the beginning of circuit design and made their way to SoC due to their well understood concepts, their compatibility with most of the available node processors, the area taken on the chip and the low latency after the arbiter has granted control.

Most of the recent designs of on-chip buses borrow their ideas from standard printed circuit board (PCB) buses. The bus architecture for SoC differs from PCB buses because SoC has faster transfer rate due to shorter propagation delays and no restrictions on number of pins due to packaging or signalling constraints.

Structurally, on-chip buses can be divided into Shared and Hierarchical buses. In hierarchical buses, two or more buses are connected via bridges. In most of the cases, the bus is divided into master and slave buses, with the master bus connecting the high speed devices and the slave bus acting as slave to the bridge and connecting low speed devices (discussed in section 2.4).

2.3.1 Shared Bus based communication architectures

A Global bus is the simplest example of shared communication architecture and is commonly found in many commercial SoCs [KYE-001]. All the processing cores are connected to a single global bus via interface. Bus access is granted on basis of different bus access protocols.

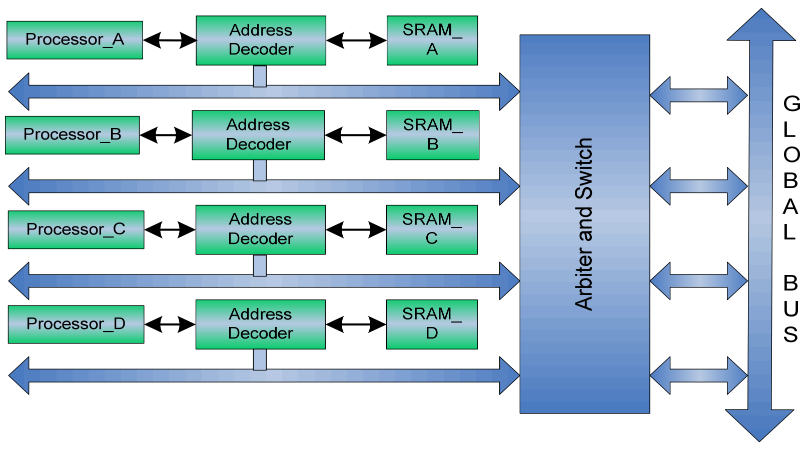


Figure 2-1 Global bus architecture with an arbiter [KYE-001]

The global bus can serve only one processing core at one time. A simple bus allocation mechanism can be in form of an arbiter that allocates the global bus to the processor core requesting to initiate communication. Figure 2-1 shows the global bus architecture where one global bus is shared by the processing cores and an arbiter is employed for granting bus access. Bus access can be granted in first come, first serve also known as first in, first out (FIFO), time sharing or by priority based allocation. Data ready flags are employed in one of the local processor memory (for example SRAM_D in the figure 2-1) by the processing core in pipelined operation to indicate that its operation is complete and data is available for next core to read and use. The data ready flag in SRAM_D is continuously checked by the processing cores for the appropriate data ready flag value to become available.

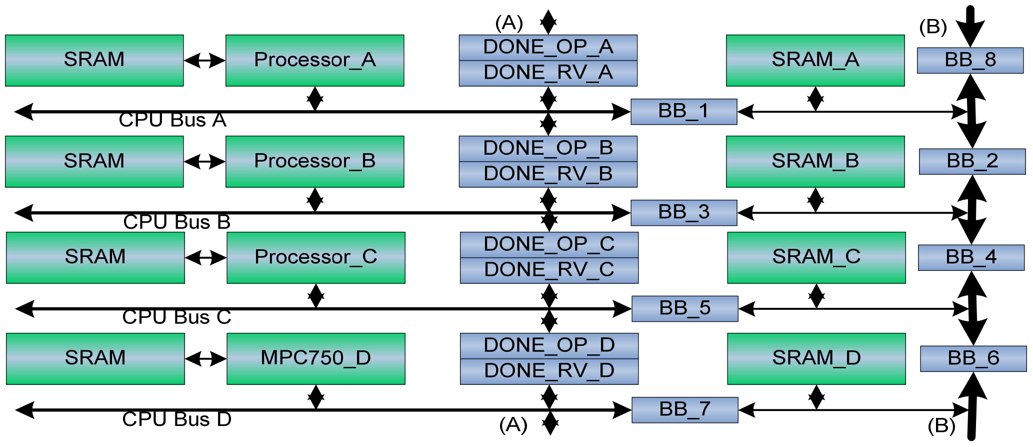


Figure 2-2 Global bus architecture with registers [KYE-001]

Another bus allocation mechanism employed consists of a set of two registers DONE_OP and DONE_RV. A flag is set in these registers after the data transfer or data receipt between the processing cores. Figure 2-2 shows the bus architecture with registers employed for bus allocation. Any processing core can access the memory of upper adjacent core through the segmented global bus. Bus bridges are employed to allow different processors to access data memory. For communication between processor_A and processor_B in the figure 2-2, the processor_A writes to local memory SRAM_A, the address decoder makes BB_1 connects to SRAM_A. The bus bridges BB_2 and BB_8 block the access to SRAM_A from any other processor. Handshaking is done by setting DONE_OP_B at the completion of its operation by processor_A. Processor_B resets DONE_OP_B and reads SRAM_A. When the processor_B finishes reading from SRAM_A, it sets DONE_RV_B. At this stage, processor_A then resets DONE_RV_B to zero and begins processing the next packet.

Bi-FIFOs in global bus architectures can also be employed for exchange of data between the communicating cores. Figure 2-3 shows a global bus architecture where data is exchanged through the Bi-FIFOs located between the cores. Bi-FIFOs can be accessed by two ports, an upper_zz port and a lower_xx port. Threshold value is defined by the user to indicate the status of the Bi-FIFO, a high threshold value indicates that the Bi-FIFO is full while a low threshold indicates that the Bi-FIFO is empty. After the completion of the assigned operation, data is pushed to Bi-FIFO until the data reaches the high threshold, at which stage an interrupt signal is generated for the adjacent core to indicate data availability. An interrupt signal is generated to the next core when the data in a Bi-FIFO reached the high threshold. The interrupted core reads the data from the Bi-FIFO until the data of the Bi-FIFO reaches the low threshold. For synchronisation through handshaking, two flag registers are used. These registers are located in the "REGISTERS" block in the Figure 2-3 along with the threshold registers.

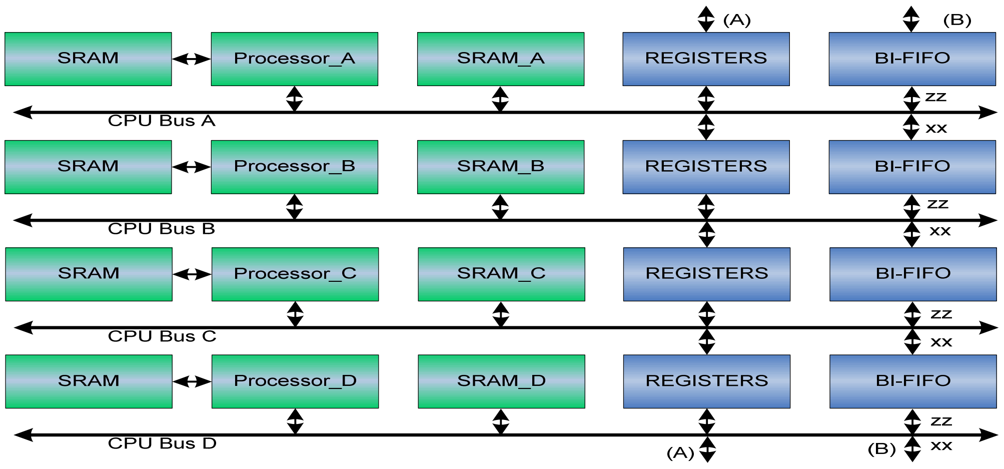


Figure 2-3 Global bus architecture with Bi-FIFO [KYE-001]

Although, system performance is heavily dependent on the application running on the system, the bus architecture with an arbiter outperforms the segmented bus architectures due to the performance difference arising from synchronisation protocols. In the segmented bus architecture with registers for data transfer, there is a gap during the read and write time in the current processing core. However, bus architecture with Bi-FIFO shows the best performance for an algorithm that has many local variables, small loops and strong data dependency between functions because they can sequentially process functions with fast memory pointer increments between processors [KYE-001].

A crossbar switch bus architecture connects multiple inputs to multiple outputs in a matrix manner. Figure 2-4 shows a crossbar switch bus architecture. Inputs and outputs are connected through the cross wires and junction switches. All the input paths and output paths have the same bus width and operation frequency. The bus width is kept same so any input path can be connected to output path. With this simple and regular structure, the crossbar switch provides simultaneous multiple connections between its inputs and outputs. When competition for the same shared resource occurs, an arbiter resolves this situation based on any of the arbitration techniques like FIFO, time shared etc.

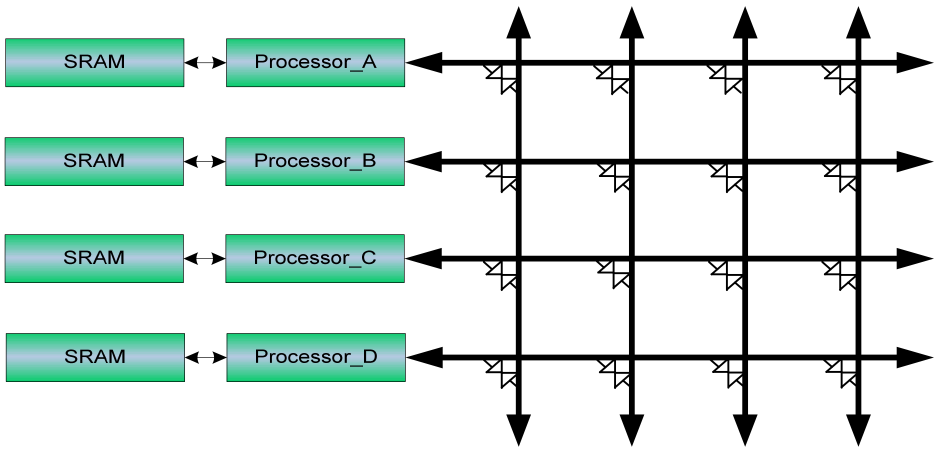


Figure 2-4 Crossbar Switch Bus Architecture [KYE-001]

2.3.2 Commercially developed shared buses

There are many shared bus architectures developed commercially, with most of the architectures developed to support the company's own cores. These bus architectures, share the same basic concepts, however, different performance tuning schemes are used to make them unique.

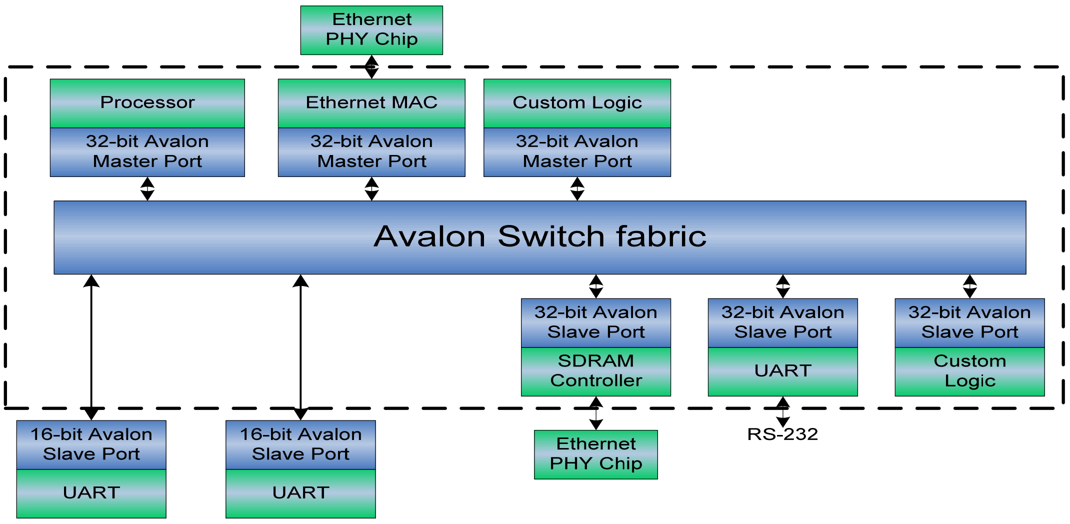


Figure 2-5 Avalon bus based system [ALT-002]

Developed by Altera, Avalon is a parameterised bus architecture mainly used for developing SoC architectures based on Nios processor [ALT-002]. Capable of transferring one data item 8-,-16-, 32-, 64-, or 128-bits wide, Avalon has a set of predefined signal types which can be used to connect the cores to its synchronous interface. Multiple bus master support is provided by Avalon along with complete specification for port connections and timing by which components communicate. Masters and slaves interact with each other based on a technique called slave-side (distributed) arbitration. Figure 2-5 shows an Avalon bus based system.

Data-path multiplexing, address decoding, wait state generation, dynamic bus sizing, interrupt priority assignment, latent transfer capabilities, and streaming read and write capabilities are some of the features provided by the Avalon bus for its attached peripherals. SOPC Builder is an exclusive Quartus II software tool that automatically generates the Avalon bus architecture to support the Nios based systems getting developed.

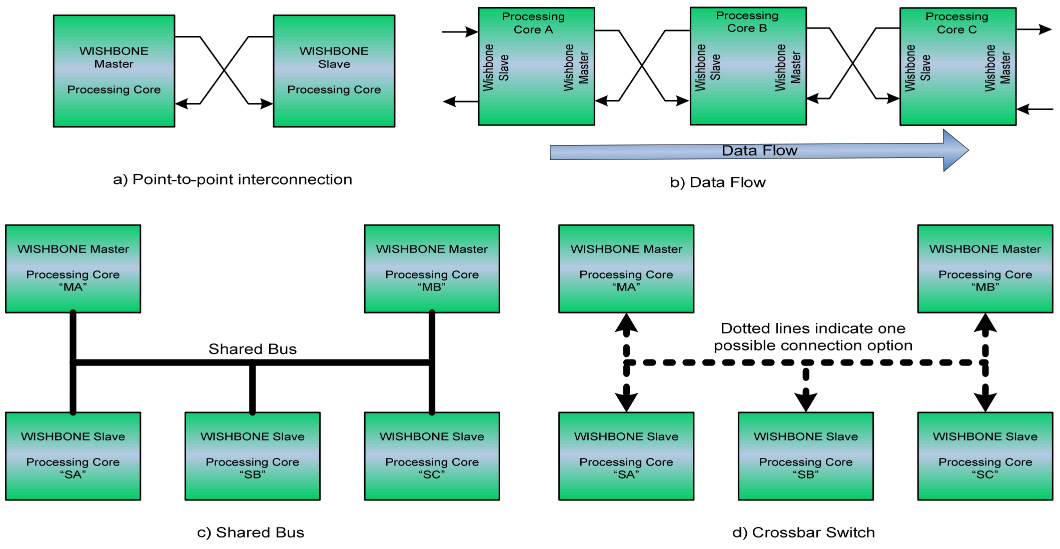

WISHBONE Master Processing Core
WISHBONE Slave Processing Core
a) Point-to-point interconnection
Processing Core A
Processing Core B
Processing Core C
Wishbone Slave
Wishbone Master
Data Flow
b) Data Flow
WISHBONE Master Processing Core "MA"
WISHBONE Master Processing Core "MB"
Shared Bus
WISHBONE Slave Processing Core "SA"
WISHBONE Slave Processing Core "SB"
WISHBONE Slave Processing Core "SC"
c) Shared Bus
Dotted lines indicate one possible connection option
d) Crossbar Switch

Figure 2-6 Wishbone interconnections [AYA-001]

Wishbone is an open source bus architecture developed by Silicore Corporation [WIS-001]. Wishbone defines different interfaces for master and slave cores [AYA-001]. Support is also provided for different types of bus transactions, such as read/write, blocking/unblocking access. In applications where two buses should exist, two separated Wishbone interfaces can be created for slow and fast transactions.

Different types of interconnect topologies (Figure 2-6) are supported by Wishbone including: a point-to-point connection for direct connection of two participants that transfer data according to some handshake protocol, a dataflow interconnection for linear systolic array architectures used in implementation of DSP algorithms, a shared bus for multi core SoCs organized around single system bus and a crossbar switch interconnection used in multi core SoCs where more than one master can simultaneously access several different slaves.

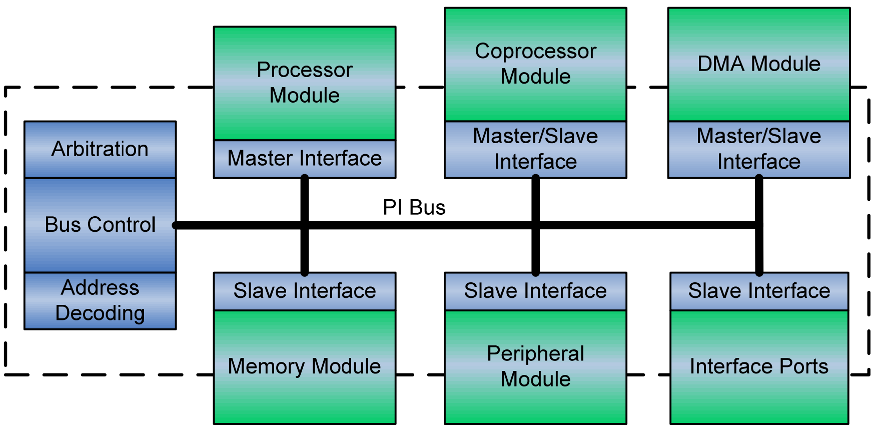


Figure 2-7 PI (Peripheral interconnect) bus architecture [MIL-001]

Peripheral Interconnect (PI) bus is an open standard published as part of a European project OMI (Open Microprocessor Initiative framework) in which several semiconductor companies (Advanced RISC Machines, Philips Semiconductors, SGSTHOMSON Microelectronics, Siemens, TEMIC/MATRA MHS) worked together to develop a bus architecture for use in modular, highly integrated SoC designs. For SoC design purpose PI bus System Toolkit is developed and synthesis scripts for different ASIC and FPGA technologies are also available [MIL-001].

PI bus is a synchronous bus with processor independent implementation and design. Address and Data bus is scalable up to 32 bits supporting 8-, 16-, and 32- bit data access. A broad range of transfer types from single to multiple data transfers is supported by PI bus along with multi-master capabilities (Figure 2-7).

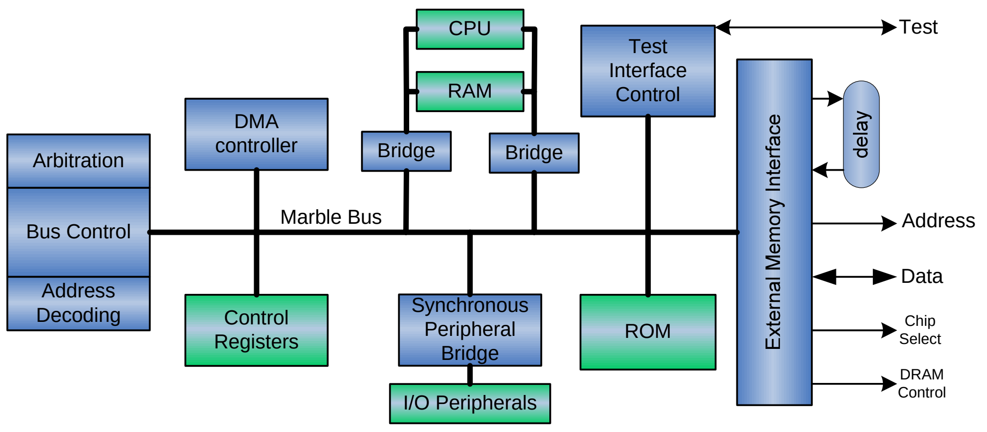


Figure 2-8 The AMULETH3H System [BAI-001]

Unlike the above mentioned examples of buses, MARBLE (Manchester Asynchronous Bus for Low Energy) is a fully asynchronous bus developed at the Manchester University [BAI-001]. The novelty of MARBLE lies in its clock-less operation and use of split-transactions for every transfer. Split-transfer architecture allows transfers between different initiators and targets to be interleaved without the need for retries, thus giving low energy operation and low latency.

In order to implement split transaction architecture, the MARBLE bus consists of two asynchronous multipoint channels. One of these channels carries the command from the Master to the slave. The other multipoint channel carries a response from the slave to the master, along with the read or write data in the appropriate direction.

Figure 2-8 shows the AMULET3H system with MARBLE bus as interconnect media connecting CPU core and DMA controller to RAM, ROM, and other peripherals.

2.4 Hierarchical bus

Despite the advantages of buses, the bus based architecture will not meet the increased communication requirement because the bandwidth of a bus is shared by all the attached devices and it is simply not sufficient [KEU-001]. Also every unit attached adds parasitic capacitance; therefore electrical performance degrades with growth. To overcome this problem, the concept of hierarchical bus originated.

SoC components are placed at the appropriate level in the hierarchy according to the performance level they require. Low performance SoC components are placed on lower performance buses, which are bridged to the higher performance buses so as not to burden the higher performance SoC components. Hierarchical buses provide an increase in data throughput over the shared buses due to decreased load per bus and the potential for transactions to proceed in parallel on different buses [LAH-002].

Examples of hierarchical bus include CoreConnect by IBM [IBM-001] and AMBA by ARM [ARM-001]. These examples are explained in the section 2.5.1 and 2.5.2 respectively.

2.5 Communication centric platform

Communication-centric platform is a bottom-up approach for SoC development. A bottom-up approach is bringing together individual modules to form the required bigger system. Communication centric platforms can be classified as SoC integration platforms where the main focus is on the hardware blocks to integrate and interact with each other efficiently [YOO-001]. In SoC integration the interface standards are important to simplify the development of complex systems and reduce the need to design glue logic that potentially degrades the performance of the system.

As mentioned earlier in this chapter, a communication centric platform only provides a verified interconnect architecture. Processor and other modules like memories and application specific cores etc. have to be added by designer to realise the complete SoC. This means that a communication centric platform can be used for different application domains.

Communication centric platforms developed commercially generally come with a standard bus interface and mostly provide hierarchical bus topology due to the reasons mentioned in section 2.4. Below are some of the commercially developed communication centric platforms.

2.5.1 IBM CoreConnect

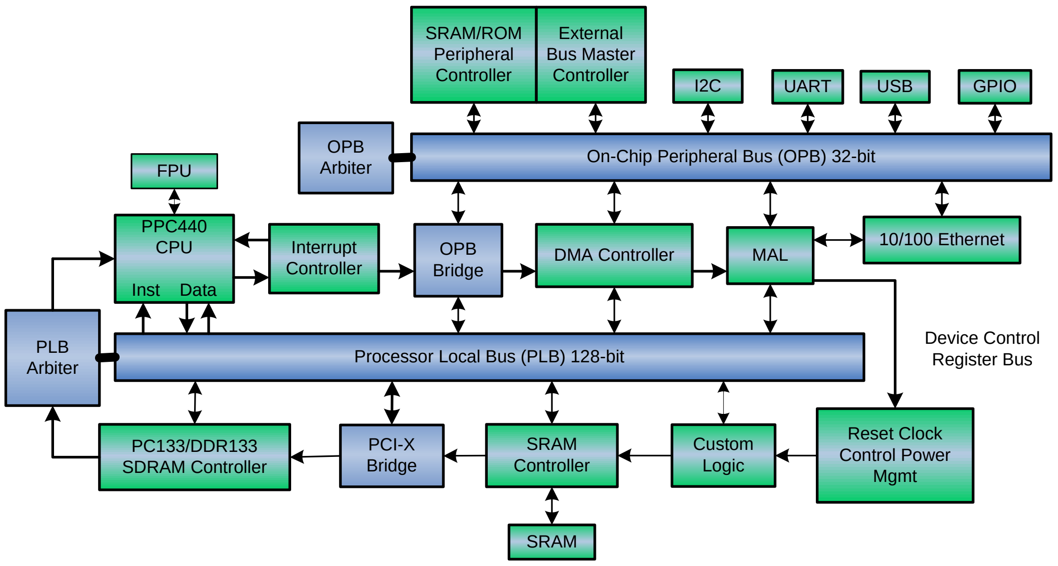

SRAM/ROM Peripheral Controller
External Bus Master Controller
I2C
UART
USB
GPIO
OPB Arbiter
On-Chip Peripheral Bus (OPB) 32-bit
FPU
PPC440 CPU
Inst
Data
Interrupt Controller
OPB Bridge
DMA Controller
MAL
10/100 Ethernet
PLB Arbiter
Processor Local Bus (PLB) 128-bit
Device Control Register Bus
PC133/DDR133 SDRAM Controller
PCI-X Bridge
SRAM Controller
Custom Logic
Reset Clock Control Power Mgmt
SRAM

Figure 2-9 CoreConnect platform architecture [IBM-001]

The CoreConnect bus architecture is developed by IBM to ease the integration and reuse of processor, system, and peripheral cores within standard product and custom SoC designs. CoreConnect serves as a foundation for the IBM Blue Logic IP collaboration program, which fosters close working relationships between IBM and select third party providers to facilitate customer utilisation of IP cores.

The IBM CoreConnect architecture provides three buses for interconnecting cores, library macros, and custom logic [IBM-001].

- Processor Local Bus (PLB)
- On-Chip Peripheral Bus (OPB)
- Device Control Register (DCR) Bus

The PLB and OPB buses provide the primary means of data flow among macro elements. Because these two buses have different structures and control signals, individual macros are designed to interface to either the PLB or the OPB. Usually the PLB interconnects high-bandwidth devices such as processor cores, external memory interfaces and DMA controllers.

Processor Local Bus

PLB provides high bandwidth data paths for the attached cores. PLB supports up to 16 masters, providing four priority levels for implementation of various arbitration schemes. Any number of slave devices can be attached to PLB, however, the number of masters and slaves attached to a PLB directly affects the maximum attainable PLB bus clock rate. This is because larger systems tend to have increased bus wire load and a longer delay in arbitrating among multiple masters and slaves.

PLB provides support for 16-, 32- and 128- bit data transfers and is extendable to 256-bit data buses. For bus arbitration, a bus arbitration and control unit is also incorporated in PLB to manage the address and data flow through PLB. Separate address and data buses are present that allow simultaneous transfer requests. The PLB arbitrates among these requests and directs the address, data and control signals from the granted master to the slave bus. The slave response is then routed from the slave bus back to the appropriate master.

On-Chip Peripheral Bus

In order to reduce capacitive load on the PLB, a secondary bus OPB is used. OPB supports multiple masters. Peripherals suitable for attachment to the OPB include serial ports, parallel ports, UARTs, GPIO, timers and other low-bandwidth devices.

As part of the IBM Blue Logic cores program, all OPB core peripherals directly attach to OPB. The OPB provides a fully synchronous protocol with separate 32-bit address and data buses and supports for multiple OPB bus masters.

Device Control Register Bus

DCR is a fully synchronous bus typically implemented as a distributed multiplexer. Lower performance status and configuration registers are typically read and written through the DCR Bus. The DCR provides a maximum throughput of one read or write transfer every two cycles to provide the required connectivity while minimizing silicon usage.

2.5.2 ARM AMBA

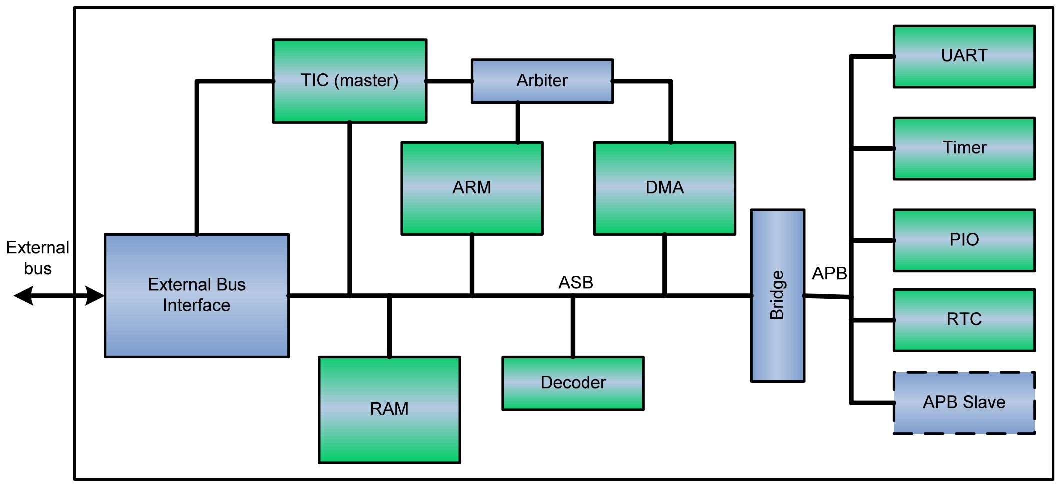


Figure 2-10 AMBA platform architecture [ARM-001]

AMBA is ARM's hierarchical bus designed to support the ARM processor cores. AMBA defines a multilevel bussing system, with a system bus and a lower-level peripheral bus [ARM-001].

- AMBA High-Speed Bus (AHB) or Advanced System Bus (ASB) and
- Advanced Peripheral Bus (APB).

The two buses are linked via a bridge that serves as the master to the peripheral bus slave devices. The system bus can be one of the two defined buses AHB or ASB. The peripheral bus, called APB for the Advanced Peripheral Bus, is a simpler, lower-speed, low-power bus for slower devices.

Advanced High-Speed Bus

The AHB takes on many characteristics of a standard plug-in bus. It's a multi master bus where all bus operations are initiated by bus masters. The master-generated address is decoded by a central address decoder that provides a select signal to the addressed bus slave unit. The bus master can "lock" the bus, reserving it with the central arbiter for a series of locked transfers.

AHB supports 32, 64, 128, and 256 bit data paths and 32 bit address bus and can be configured for Pipelined and split transactions

Advanced System Bus

Like the AHB, the ASB is a pipelined, multi master bus that supports bursting. It is a simpler bus and does not support split transactions. Like AHB, it enables a master to reserve the bus. ASB supports 32 bit data paths and 32 bit address bus.

Advanced Peripheral Bus

Designed to support low-speed peripherals such as universal asynchronous receivers/transmitters (UARTs), keypads, and programmed inputs/outputs (PIOs), the APB is a simple peripheral bus. All bus devices are slaves to the master, the bridge to the AHB, or ASB system bus. This is a static bus that provides a simple address, with latched address and control signals for easy interfacing. As a simple bus, the APB supports 8, 16 and 32 bit data bus and 32 bit address bus.

2.5.3 Palmchip CoreFrame

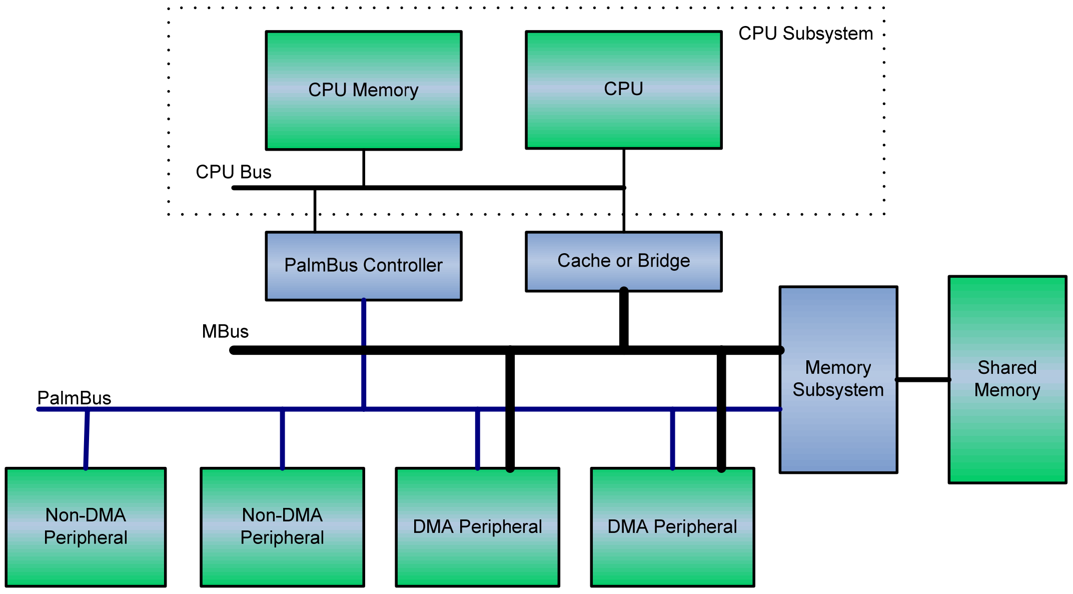


Figure 2-11 CoreFrame platform architecture [PAL-001]

The CoreFrame developed by PalmChip is low power, high performance on chip interconnect architecture that provides a platform for components to be integrated for rapid development of SoC architectures [PAL-001].

It can also be viewed as a system of independent parallel buses rather than a hierarchy of buses. The two most important buses are PalmBus and the MBus. The PalmBus is designed for low speed accesses from the CPU core to peripheral blocks. The MBus is designed for high speed accesses to shared memory from the CPU core and peripheral blocks. There is also a CPU bus that is used to connect CPU to PalmBus via palm-bus controller and to MBus through a cache or a bridge.

Channels are used to interface between the MBus and the Memory subsystem. The channel handles all interfacing between the peripheral and the MBus. Use of channels makes it easier to integrate peripherals that require DMA as many system issues including pipelining, memory addressing, arbitration and endianness do not have to be dealt by the peripheral. Address control is performed by simply passing the address from the channel interface to the MBus. The simplicity of the Channel interface makes the task of integrating peripheral blocks from multiple sources becomes much simpler. The channel library also includes verification stimuli and test benches [PAL-002].

2.5.4 Sonic's Silicon Backplane

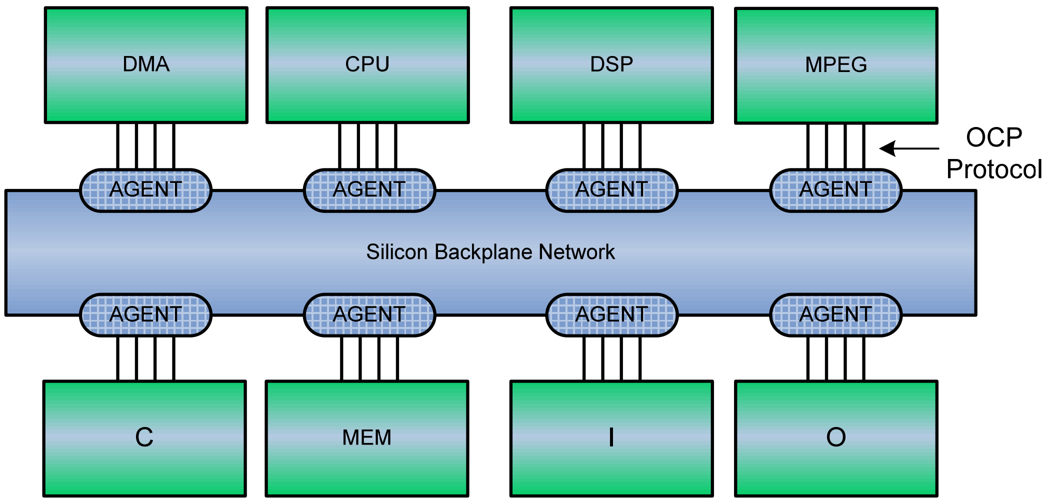


Figure 2-12 SiliconBackplane platform architecture [SON-001]

SiliconBackplane [SON-001] is an on-chip bus framework that connects processing cores in SoC. It is based on OCP bus interface protocol thus it does not require additional design work or glue logic to integrate cores to the platform. Sonics has also added a multicast feature that allows designers to send data to multiple slaves. The multi-backplane feature allows designers to create a hierarchical subsystem each with an independent clock frequency and data path width. The sub systems can be connected in tree or fully connected topologies to isolate local data flows, improving total system bandwidth while reducing SoC area and power consumption [SAN-002].

To overcome shortcomings of bus based communications, a temporal solution is proposed by Sonics [WIN-001]. Sonics decouples communication from computing and introduces a communication subsystem that can be tuned to the required bandwidth. Rather than using a fixed interface standard, an optimised interconnect interface targeted for the desired application is used. To decouple communication from computation, FIFO buffer and burst transfer of data is introduced by grouping the related transfers into bursts. The decoupling of computation-intensive data is effective for best-case performance but shows poor performance in satisfying real-time deadlines. TDMA is used to transfer the data across a higher-bandwidth channel with minimal buffering, and higher-level protocols are adopted to select the receiving device. The result is highly efficient interleaved transfers. [WIN-001]

2.6 Bus Features Overview

Table 2-1 SoC Bus Features Overview

Name	Structure	Uni/Bi-directional	Shared/point-to-point connections	Synchronous /asynchronous	Multiple clock domain	Transfer
AMBA	Hierarchical	Uni-directional	Shared	Synchronous	N/A	Split/Pipelined Transfer
Avalon	Shared	Bi-directional	point-to-point	Synchronous	Multiple clock domain	Pipelined Transfer
CoreConnect	Hierarchical	Uni-directional	Shared	Synchronous	N/A	Pipelined Transfer
CoreFrame	Hierarchical	Uni-directional	point-to-point	Synchronous	Multiple clock domain	N/A
MARBLE	Shared	Bi-directional	Shared	Asynchronous	Multiple clock domain	Split/Pipelined Transfer
PI Bus	Shared	Bi-directional	Shared	Synchronous	-	Split Transfer
Silicon Backplane	Shared	N/A	Shared	Synchronous	Multiple clock domain	Pipelined Transfer
Wishbone	Hierarchical	Uni/bi-directional	Shared	Synchronous	Multiple clock domain	N/A

The main features of the SoC buses mentioned in section 2.3 and 2.5 are summarised in Table 2-1. The features analysed include network topology features like structure, shared or point to point connections, clocking and data transfer type.

In section 2.3, some commercially developed shared buses were introduced. An important issue to consider in dealing with a communication medium is its scalability. Scalability of a communication medium refers to its capability to deal with an increase in data throughput requirement caused by addition of new resources. Although, shared buses are simpler than hierarchical buses, they have a drawback when it comes to scalability and suffer from poor resource sharing in time domain leading to high contention or resource utilisation. Hierarchical buses solve the problem of scalability and resource sharing to some extent but will still prove to be bottleneck in future high throughput SoC architectures.

Uni- directional buses are believed to be faster than bi- directional buses however; bi-directional signal lines save routing resources. Tri-state buffers are used in implementation of bi-directional lines that are not suitable for ASIC design due to difficulties in control and testing. Apart from AMBA, CoreConnect and CoreFrame most of the buses use uni-directional lines. Wishbone supports both uni- and bi-directional lines [SAL-001].

Apart from MARBLE Bus, all the buses are synchronous in nature. MARBLE bus employs globally asynchronous locally synchronous clocking that save power consumption as different modules can run at reduced frequencies irrespective of other modules in system. This comes at a cost of increased system complexity [APT-001].

PI Bus is the only bus that supports only split transfer. All the other buses support both split and pipelined transfer. Using pipelined transfer, higher clock frequencies can be obtained. On the other hand split transfer has the advantage that bus is not reserved for the whole operation by one bus master allowing other cores to utilise the bus efficiently.

2.7 Performance Metrics

Data throughput plays an important part in establishing the suitability of a bus to be used as communication media for a particular SoC. Another important performance metric is the latency of a transfer. Latency is the time taken to execute a transaction across the bus. It has two components, firstly the time it takes to access the bus, which depends on bus protocol and utilisation and secondly, the time it takes to transfer the data, which is directly determined by the protocol of the bus and bus width. Time is usually measured in clock cycles.

Bus bandwidth, also referred to as data throughput is the maximum capacity for data transfer as a function time and is given by:

$$\text{Bandwidth}_{\text{bus}} = \text{Width}_{\text{bus}} \times \text{Clock-frequency}_{\text{bus}} \quad \ldots\ldots\ldots\ldots\ldots \text{ Equation 2.1}$$

Bus width is specified in bits and the clock-frequency in megahertz then bandwidth is measured in megabits per second.

From Equation 2.1, it can be seen that clock frequency plays an important role in available bandwidth. In a system with a fixed bandwidth, a rise in bus clock frequency implies a shorter bus clock cycle period. e.g. a bus with clock frequency of 100 MHz has a bus clock cycle duration of 10 ns, whereas a bus with a higher clock frequency of 500 MHz has a bus clock cycle of only 2 ns [PAS-001].

Entering into the DSM era has got new challenges for on-chip buses. Buses are implemented as long metal lines on a silicon wafer and data is transferred using electromagnetic waves that have a finite speed limit. These metal lines do not decrease in size in proportion to the decreasing logic components. This results in relatively longer communication lengths between logic components. Moreover, with increasing clock frequencies, the distance that can be covered by a signal on the bus in a single clock has been reduced and it can take multiple cycles to send a signal across a chip. This increase in signal propagation time has serious consequences for the performance and correct functioning of the SoC design.

In order to tackle this signal propagation problem, hierarchical or split bus communication architecture are employed that partition the long bus lines into shorter ones, separated by bridges or tri-state buffer structures. This breakdown makes it possible for signal to transverse a bus segment in a single clock cycle and with hierarchical buses this separation allows different buses to operate at different bus clock frequencies.

2.8 Summary

This chapter introduced the basics of on-chip communication. A literature review has been done with respect to shared and hierarchical buses and concept of communication centric platform based design is introduced. Various commercially developed shared and hierarchical buses are reviewed and their architectural characteristics are listed.

Reviewing the buses brings us to the conclusion that most bus properties are similar with only a few exceptions of performance tuning parameters. In order to realise a bus, a designer would use either shared/hierarchical topology (depending on the application domain), synchronous, uni-directional lines. The data transfer can use pipelining or split transactions with handshaking. Dynamic arbitration is effective but complex in nature when it comes to implementation. The increase in complexity brings with it an increased area and power penalty. If this overhead is justified, support for multiple clock domains, dynamic reconfiguration and asynchronous clocking can be included to implement an optimised bus architecture. Coupled with an interfacing standard to ease the process of integration, a communication platform can be developed.

The main aim of this chapter was to establish the current trends in communication centric platforms and to establish the requirements for future communication centric platforms. It can be concluded that current communication centric platforms provide full framework for communication but take a hierarchical bus based approach which might not be the ideal communication medium for the different application domains. In addition to this, the time spent on customisation and modifying third party cores to integrate with the platform is considerably high. A proposed solution for future communication centric platform is suggested in chapter 4.

It is believed that bus based communication system will not satisfy the performance requirements of future multi-core SoCs [JAR-001]. To tackle this problem, notion of NoC is introduced. The next Chapter deals with the origin of NoC and establishes the connections between NoC and platform based designs.

Chapter 3
LITERATURE REVIEW
NOC as Communication Centric Platform

3.1 Network on chip

With the scaling down of device feature size and scaling up of design complexity, throughput limitations, signal integrity and signal latency are becoming a bottleneck in future communication centric SoC design. For future on-chip communication, a new communication medium is needed. This communication medium should aim to provide low power, utilise link efficiently, reduce contention and occupy less area on silicon [JAR-001].

NoC is considered to be the solution to this communication bottleneck. This chapter focuses on the NoC as an emerging communication medium. It starts with the basic concepts of NoC. NoC layered approach and NoC as an ideal candidate for future communication centric platform is then discussed. This is followed by a review of work done in the field of NoC. To conclude the chapter performance metrics of NoC are described.

3.2 Basic Concept of NoC

The basic concepts and techniques of NoC are built upon the successful and well established computer networking domain. However, NoC differs from the traditional network because of local proximity of attached cores and because NoCs are more predictable at design time. The main idea is to have the processing core abstracted as a node, and the nodes are interconnected by the micro network that can provide scalable and concurrent point to point or point to many connection. Figure 3-1 shows an example of a basic NoC architecture. 16 network routers are connected in a 4x4 2D Mesh topology.

Routers also serve as an interface for the modules to be attached. A module can be a general purpose processor, a DSP or a memory sub-system. Instead of routing design-specific global on-chip wires, the communication between modules occur through routing packets via interface provided by routers.

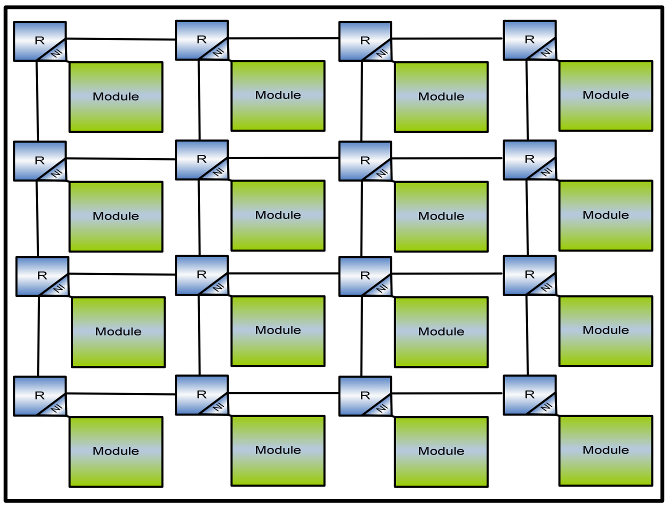


Figure 3-1 Basic NoC architecture

NoC communication is controlled by protocols also referred to as a micro network stack". It is designed in layers and is an adaptation of OSI seven layers scheme [WAR-001]. The micro network stack is composed of five layers, application, transport, network, data link, and physical layer as shown in table below [BEN-002].

Table 3-1 Micro Network Stack [BEN-002]

Software	Application Layer
Architecture & Control	Transport Layer Network Layer Data Link Layer
Physical	Wiring

As mentioned earlier in section 2.2, there are certain factors that have to be considered when designing effective on-chip communication architecture. The micro network stack caters for all those factors. In order to understand the working of NoC each layer is discussed below.

3.3 Physical Layer

The physical layer deals with the physical characteristics of the medium that connects the switches and resources with each other. The physical implementation of a communication channel is in the form of wires and the physical layer deals with the voltage levels, lengths and width of wires, signal timings, and number of wires connecting the switches, etc.

The Physical layer in NoC differs from board level and large scale networks as on-chip networks have abundant wiring resource. Dally [DAL-001] in his paper argued that on a small 3mm×3mm tile from a 12mm×12mm chip in 0.1µm CMOS with 0.5µm wire pitch, there can be up to 6,000 wires on each metal layer crossing each edge of a tile. This means that the designer can trade off wiring resources for network performance.

In future the wiring layers can and will exceed 10 levels, with global wires on the top metal layer. Wire widths will increase with wiring levels, with wires at the top level being wider than lower level wires [THE-001]. The advantage of having increased width wires is of low resistance and the advantage of having increased spacing between them is reduced capacitance.

The critical challenge for NoC is to provide adequate QoS with a limited energy budget. One of the QoS requirements is the reliability of the communication media. The most common sources of on-chip noise are crosstalk, power supply noise, electromagnetic interference and inter-symbol interference etc [BEN-003][SYL-001][BAK-001].

3.4 Architecture & Control

The architecture and control of the network is taken care of by the data link layer, network layer and transport layer. The data link layer ensures reliable communication over the physical layer, which requires error correction and detection. If the information is corrupt it will be left for upper layers to deal with.

The network layer deals with switching and routing aspects of the packetised data. Switching is the type of connection (packet switching or circuit switching) and routing is the path followed (predicted by the routing algorithms).

The transport layer provides the initial establishment of communication channels, the transfer of data and the final release of the channels. For data transfer it interfaces with the network layer to ensure an error free virtual point to point connection. Unlike the transport layer of the OSI model, this layer in NoC also deals with decomposition of data into packets at the source and their assembly at the destination. Packet size is application specific in SoC.

3.4.1. Data Link Layer

As described above, the data link layer abstracts the physical layer and treats it as a medium with a non-zero probability of errors in the transmitted bit stream. This probability of errors is increasing with the technology scaling down. Thus, the data link layer serves the job of increasing the reliability of the physical layer up to the minimum required level. It also serves to regulate the access to shared medium network where the contention for a communication channel is possible.

The two schemes that can be implemented for error detection and error correction can be classified into error detection with retransmission and error correction using the information transmitted in packet for data correction. Error correction/detection requires an encoder/decoder pair at the channel's end. Error detection is the first stage in both the schemes; however the difference occurs when data has to be retransmitted in the first case which has a price in terms of latency. The hardware overhead in this case is negligible as it only requires one extra bit of information per flit of data transfer. In latter case, the decoder is more complex because of correction circuitry.

There are several error detecting and correcting codes [WAR-001][BER-001][LIN-001]. When choosing the right error correcting code, latency and energy consumption are very important design parameters. Generally, in NoC, error detection or error correction schemes are rarely used because of the limited resources, except for very-high-speed links.

3.4.2 Network Layer

The network layer is responsible for the transfer of packets across multiple links or across multiple networks. Design issues of the network layer include the topology construction, routing scheme and arbitration policy for congestion control.

Before proceeding to the switching and routing aspects of the packets, it is important to look at network topologies and types of network that require the use of network layer. Due to different performance requirements, many different network topologies are designed. Networks can be categorised into two categories; direct and indirect networks [DUA-001].

In direct networks, processing cores are connected directly with each other by the network and routing is performed by the node for transmission of data. For example, orthogonal and octagonal topology. Orthogonal topology of connecting nodes is widely used in parallel computing platforms. The nodes are connected in k-ary n-dimensional mesh or k-ary n-dimensional torus formations. Due to the regularity of the network, the interconnect length between nodes is uniform [DAL-002]. Figure 3-2 shows a 4-ary 2-dim mesh and torus formation.

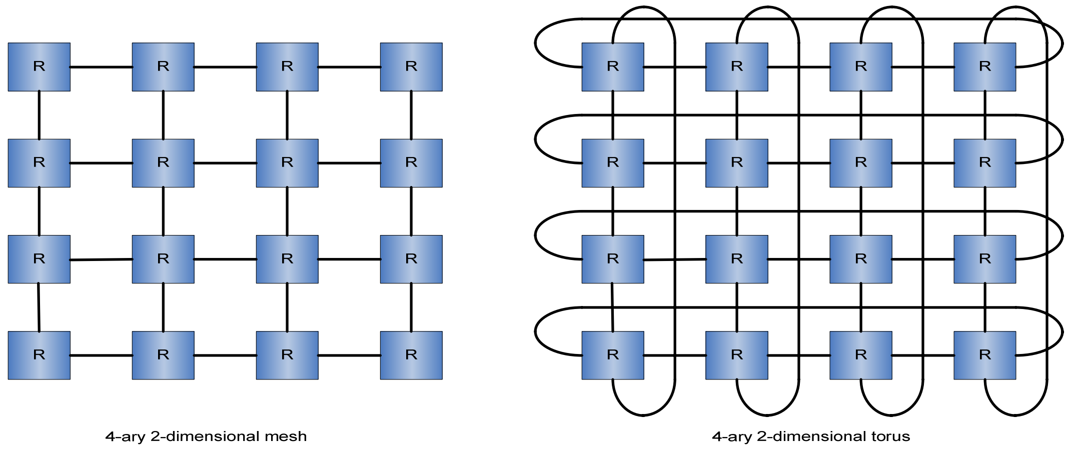


Figure 3-2 Orthogonal Topology – (Mesh and Torus networks)

Octagon topology is another example of direct network topology where eight processors are connected by an octagonal ring and three diameters. Figure 3-3 shows formation of an octagonal topology. It can be seen from the figure that within the local ring, the delays between any two node processors are no more than two stages and within the neighbouring rings, the delays are no more than three stages. By using one node processor more octagons can be added in the network making octagon network highly scalable [KAR-001].

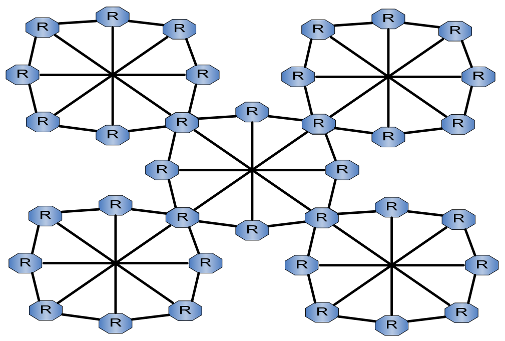


Figure 3-3 The Octagon Topology

In an Indirect network (Figure 3-4), Processing cores are connected by one or more intermediate node switches that perform the routing functions as well. A simple example can be of a crossbar network, in an N×N crossbar network N input ports are connected with N output ports. Any of the N input ports can be connected to any of the N output ports by a node switch on the corresponding cross-point. Buffers can be placed at inputs, outputs or at the cross-points of interconnection matrix. If buffers are placed at the cross-points of the interconnection matrix, a butterfly switch fabric is formed. A butterfly switch may require a large increase in the complexity of the switching element, as each cross point now requires memory [JON-001].

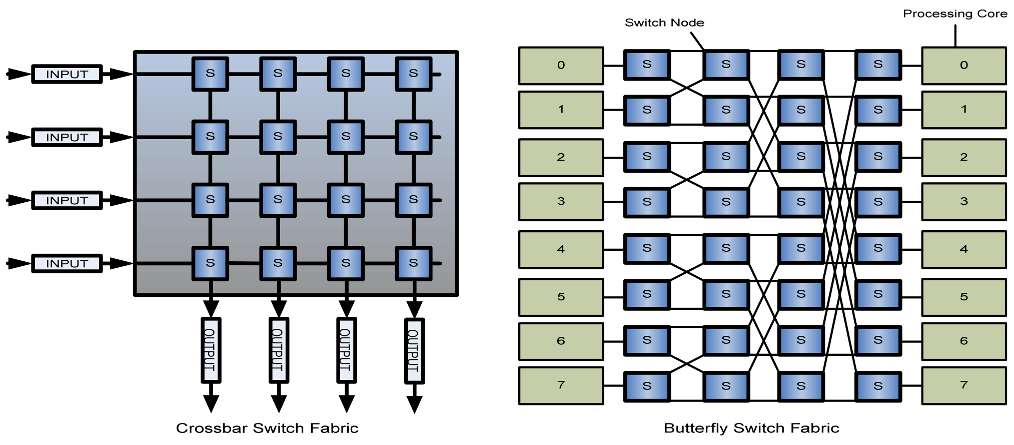


Figure 3-4 Indirect Network Topologies

3.4.2.1 Routing in NoC

For maximum system performance, a routing algorithm should have high throughput and should provide low latency message delivery, avoidance of deadlocks, live-locks, starvation, and ability to work well under various traffic patterns [FEL-001].

Network routing algorithms are classified in two types, deterministic and adaptive [MCK-001]. In deterministic routing algorithms, transfer path is determined in advance by the router, based on the source and the destination address. Deterministic routing is simple to implement, provides low packet transfer latency when there is no congestion, but data throughput decreases with increasing packet injection rate [RIJ-001].

XY routing is a popular deterministic routing algorithm [ZHO-001]. The message is divided into a sequence of fixed-size units of data, called flits. Flits are first routed in the X direction, until reaching the Y_{Target} coordinate, and afterwards in the Y direction. If some network hop is in use by another packet, the flit remains blocked in the switch until the path is released.

In adaptive routing algorithms, each packet's transfer path is calculated based on current network conditions [NIL-001][BEN-004]. When network congestion occurs, a better path is calculated to avoid congested links. Adaptive algorithms improve the packet's latency and throughput at cost of a more complex router implementation.

Store-and-forward, virtual cut through and wormhole are popular adaptive routing techniques. With store-and-forward, the message latency is the product of the number of hops taken and the sum of the average queuing delay and transmission time of the message per hop [BER-001], this implies that if a *b*-flit message transverse a path of length *d*, and is never delayed, then it will reach its destination in *bd* steps (assuming it is getting transmitted by each channel in each hop).

Virtual cut through routing differs from store-and-forward routing as it forwards the packet to the next node before it is entirely received by the current router. This reduces the buffer size and store-and-forward delays. However in case the next router is not available, then the whole packet has to be stored in buffer of current router.

In the wormhole routing technique [DAL-003], a message is divided into flits. If a communication channel transmits the first flit of a message, it must transmit all the remaining flits of the same message before transmitting flits of another message. In this method, the message latency is proportional to the sum of the number of cycles spent in waiting for suitable channels to route message flits, number of hops, and message length. In the example above with a *b*-flit message transverse a path of length *d*, the first flit does not wait for the rest of the message. It therefore arrives at its destination after *d* steps, and the last flit of the message arrives after *d+b-1* steps. The difference in time is due to a better utilisation of network edges by the wormhole router. In addition to reduced latency, wormhole routing also has the advantage that it can be implemented with small and fast switches.

Wormhole routing performance is prone to deadlock and live-lock issues [SHI-001]. A deadlock occurs when a message waits for an event that will never happen; a live-lock keeps a message moving indefinitely but not letting it reach the destination. To solve this problem, dimension-order-routing [DUA-002][WUJ-001] in which packets are routed in only one dimension till they reach the destination row, and then switch to other dimension until the destination, is used or virtual channel approach [DUA-002][DAL-004] is used in which one physical channel is split into several virtual channels. However, using virtual channel require the use of more buffer space. Store-and-forward routing is not very successful in NoC due to the fact that it requires buffer spaces in router for storage of packet thus increasing the area and also increases energy consumption due to extra switching involved.

Live-lock occurs when a packet is running in a circular motion around its destination. Hot potato routing [FEI-001] can be considered as an example of live-lock. Hot potato routing is based on the assumption that every switch (router) has equal number of input and output channels. Thus when contention occurs and the desired channel is not available, the packet instead of waiting is routed to the other channel. Proper deflection rules can be defined to avoid live-lock problem. Significant research is carried out in the field of network routing. Analysis of routing algorithms can be found in [GLA-001][DUA-002][CHI-001][SHI-001].

3.4.2.2 Contention Awareness

In order to decrease the switch buffer size and to efficiently route packets, contention awareness can be used. Since NoC can take advantage of dedicated control wires, the state of neighbouring router can be exchanged to help the routers make switching decisions to route the data packets to the links with less contention [NIL-001]. Section 3.4.3.1 explains the concept of contention in NoCs and suggests methods for contention free communication.

3.4.3 Transport Layer

The transport layer performs packetisation and de-packetisation of messages at source and destination respectively. The size of the packets has a direct impact on both performance and energy consumption. The optimal packet length for performance may differ from the optimal packet length from an energy standpoint [BEN-001]. The transport layer also deals with the network flow control.

3.4.3.1 Network flow Control

NoC performance greatly depends on an effective network flow control mechanism. Network flow control is responsible for the correct delivery of packets. This involves coordination between the sender and receiver and effective contention resolution. Contention can cause packets to get blocked, stalled, detoured or simply dropped. Thus, the presence of a flow control mechanism is important for the efficient bandwidth utilization.

A flow control mechanism can be divided into buffer-less flow control and buffered flow control [DAL-005]. As the name indicates, buffer-less flow control does not rely on switch buffers. Due to the absence of buffers in the switch, packets cannot be stored and thus packets contend for bandwidth. Arbitration is generally employed to deal with contention between contending packets.

End-to-end flow control algorithms is used in buffer-less flow control that conserves the number of packets in the network by regulating the packet injection rate at the source. End-to-end algorithms are generally not employed in NoC due to the large overhead associated with sending the feedback information and the instability that can occur if there is unpredictable delay in feedback loop [UNI-001].

In buffered flow control, blocked packets are stored in switch buffers while they wait for the access to the network resources. Store-and-forward, virtual cut-through and wormhole switching techniques adopt buffered flow control. Link-level flow control mechanisms, in which the buffer availability information is propagated between switches, are used in buffered flow control.

Credit-based, on/off, and ack/nack are three common types of link-level flow control techniques [DAL-005]. Since packet injection rate is not monitored and flow control relies on propagation of congestion information back to the source, packets can cause congestion in the network if the information is not processed efficiently.

3.5 NoC as Communication Centric Platforms

The NoC layered approach discussed in this chapter has the advantage that it decomposes the communication problem into more manageable components at different hierarchical layers. Each layer has different functionality implemented independently from other layers. Adding a new function to a layer only requires modifying the functionality of one layer and reusing the functionalities of other layers to generate a new NoC architecture.

As mentioned in previous chapter, developing a programmable, reconfigurable and scalable communication platform is essential for SoC designs. NoC serves as a communication and integration platform providing hardware communication architecture and interfaces for integrating hardware cores. NoC also favours architecture level reuse which makes it an ideal candidate for an efficient communication centric platform.

As a platform, NoC provides well-defined interfaces for application programming and component integration. In order to connect commercially developed hardware cores, interfacing standards like OCP [OCP-001] or VCI [VIS-001] should be supported. On the other hand interfaces should be provided for integrating hardware logic via a communication adapter and for programming embedded software.

3.6 Development History

NoC has been under the spotlight since it was first introduced and many research groups are working on different aspects of NoC design. In 2000 Hemani et. al. [HEM-001] proposed a packet switched architecture with switches surrounded by six resources and connected to 6 neighbouring switches. The architecture was called Honeycomb due to the hexagon based pattern of switches and resources. The concept of packet switching re-appeared in other consecutive approaches but the topology simplified in most proposals to a mesh of resources and switches [GUE-001].

In 2001, W. Dally and B. Towels [DAL-001] proposed replacing global wiring with a general purpose on chip interconnection network with an area overhead of 6.6%. A 12mm×12mm chip in 0.1μm CMOS technology was developed. Dally concluded that there can be up to 6,000 wires on each metal layer crossing each edge of a tile(3mm×3mm). It is quite easy to achieve over 24,000 'pins' crossing the four edges of a tile. By effectively choosing the network topology these abundant wiring resources can be converted into bandwidth.

L. Benini [BEN-002] proposed a layered design methodology in 2002 borrowing models, techniques and tools from the network design field and applying them to SoC design. Several open problems at various layers of the communication stack were addressed and a basic strategy was given to effectively tackle them for energy efficient design. Xpipes compiler was also presented as a tool for automatically instantiating an application specific NoC for heterogeneous multi-processor SoCs.

S. Kumar [SHA-001][SUN-001] constructed a model of NoC using a public domain network simulator NS-2 and evaluated design options for a specific NoC architecture which has a two dimensional mesh of switches. S. Kumar analysed the series of simulation results to determine the relationship between buffer size in switch, communication load, packet delay and packet drop probability. The results are useful for the design of an appropriate switch for the NoC.

Shin et al. [SHI-002] proposed a hybrid switching scheme that dynamically combines both virtual cut-through and wormhole switching to provide higher achievable throughput values compared to wormhole switching alone. J. Hu [MAR-001] proposed a smart NoC, which combines the advantages of both deterministic and adaptive routing schemes in a NoC environment.

NoC is relatively a new concept but it has been rapidly accepted in academia with much industrial interest in it. A comprehensive survey on current research and practices of NoC can be found in [BJE-001].

3.7 Advantages of NoC

NoC offers certain advantages over the traditional bus based and crossbar communication media, especially due to its layered approach,

Scalability: In a bus based communication, the number of modules attached is limited by the bandwidth available. However, NoC does not have such problem and integration of an additional module means introduction of a router to the network that scales up the network bandwidth.

Throughput: The bandwidth of a bus is shared by all the attached devices and it is simply not sufficient for future SoC with throughput requirement of over 10 GB/s. In NoC, throughput is dependent on the actual physical transport media and can be realised by allocating several physical links for a logical path. Because the switch fabric does not store transaction state, throughput simply scales with the operating frequency, number and width of switches and links between them.

Optimization: The layered structure of NoC offers the advantage of separate optimizations of transaction and physical layers. The transaction layer is mostly influenced by application requirements, while the physical layer is mostly influenced by Silicon process characteristics.

Quality of Service: QoS defines the level of commitment for packet delivery. This can be in the form of correctness of the result, completion of the transaction or on the performance [GOO-001]. On-chip message delivery and can be achieved through different means at different levels. For example, packet integrity can be ensured not only by error-correction at the link layer but also by re-transmission at the upper layers. NoC offers a far better QoS then traditional on-chip interconnects.

Verification: The layering approach presents an ideal case for communication system verification, tackling one layer at one time.

Customisation: User-specific information can be easily added to packets and transported between routers. Custom-designed NoC units make use of such information broadening the application domain for NoC products.

In designing a communication medium for an SoC architecture, latency constraint is an important issue to consider. Thus, adaptation of concepts from data communication networks that tend to be focused on bandwidth related QoS requires a constant focus on appropriate trade-offs.

3.8 Analysis of NoC

As mentioned in section 2.7, throughput and latency plays an important part when determining the suitability of a communication media for any application domain. This is becoming more important with the future SoC moving to a communication centric approach.

In NoC, throughput signifies the maximum value of the traffic that a network can handle. It is related to the peak data rate sustainable by the system. Throughput in a packet based on-chip system can be given by:

$$Throughput = \frac{(Total\ Packets\ Completed) x\ (Message\ Length)}{(No.of\ Cores) x\ (Total\ Time)}$$

Where,

Total messages completed refers to the number of whole messages that successfully arrive at their destination.

Message length is measured in flits,

Number of Cores is the number of functional processing cores involved in the communication,

and *Total time* is the time (in clock cycles) that elapses between the occurrence of the first message generation and the last message reception.

Thus, message throughput is measured as the fraction of the maximum load that the network is capable of physically handling. Accordingly, throughput is measured in flits/cycle/core.

Latency is defined as the time elapsed between the occurrence of a message header injection into the network at the source node and the occurrence of a tail flit reception at the destination node. Latency depends heavily on the routing algorithm.

In the case of store-and-forward routing, where network nodes receive an entire packet before forwarding it to the next node. Both link bandwidth and buffers are allocated at the packet-level. Assuming no contention, the latency is given by,

$$Latency = \left(\frac{No.of\ Flits}{Link\ Bandwidth} + \ Routing\ Delay\ per\ node\right) \times \ no.of\ hops$$

Where,

Number of hops is the number of hops from the source node to the destination node.

In case of virtual cut through, a network node does not wait for the reception of an entire packet. It receives a portion of the packet and then forwards it to the next router, provided the buffer space in the next switch is available. In case of blocking, the entire packet is shunted into the allocated buffers. Assuming no contention and that the packet is forwarded as soon as possible, Latency is reduced to,

$$Latency = \frac{No.of\ Flits}{Link\ Bandwidth} + Routing\ Delay\ per\ node\ \times\ no.of\ hops$$

In wormhole routing, due to the pipelined transmission, Latency is same as that for virtual cut through routing.

3.9 Summary

This chapter introduced the basics of NoC communication and describes the layered structure of NoC. In order to develop an effective NoC architecture, a suitable topology, effective switching and routing scheme and flow control mechanism has to be considered. One of the main reasons for this research was to evaluate the suitability of NoC for communication centric platform based designs. It can be concluded that NoC provides a complete framework for effective on-chip communication architecture and a complete platform for integrating hardware cores. Coupled with an interfacing standard, NoC is an ideal candidate for an efficient communication centric platform.

Review of current NoC architectures highlights a major design drawback. The architectural parameters that greatly affect NoC performance are fixed at design time. In a future multi-core SoC architecture, this can result in inefficient utilisation of resources affecting the overall system performance and power. To tackle this problem, a dynamically reconfigurable NoC is proposed in chapter 6.

Chapter 4

Proposed Communication Centric Platforms

4.1 Introduction

In chapter 2, a review of current on-chip communication and current communication centric platforms was presented. In today's electronic industry, time to market is one factor that greatly determines the success of the developed product. Re-use design methodologies and ease of integration is seen as the way forward to reduce this time to market constraint.

Reconfiguration has not only emerged as an efficient way of development of low power architectures for multi-standard applications but also plays an important part in the reuse design paradigm. Moore's law is driving the integration of many cores in a single chip. This has enabled mixing of various traffic types in the same SoC design. These traffic types, although very different in nature must now share interconnect resources that were handcrafted to the particular traffic in the past.

When talking about integration of cores, interconnect interface plays an important role when making decision about choice of communication media. Following an industrially accepted interface standard makes it easy for designers to integrate the developed cores and greatly reduces the design time.

Taking the platform based approach for ease of integration in future SoC architectures, a communication centric platform based approach is proposed. Four different communication platforms are developed consisting of a traditional bus based, crossbar based, hierarchical bus based and a novel hybrid communication medium based.

A platform is taken as a framework where the components, be it fixed, reconfigurable, processor cores or memory blocks etc. can be integrated in plug-and-play fashion[AHM-005]. Looking at the fact that different types of communication media are effective for different application domains, the proposed communication centric platforms offer different communication media to cater for constraints like throughput, power and area in different application domains. For ease of integration and reduced design time, an OCP based socket approach is taken when designing the interfaces for the platforms.

A novel hybrid communication architecture is integrated in the platform. The hybrid communication medium combines the advantages of bus based and crossbar based communication media in one SoC. The co-existence of crossbars and a shared bus in the architecture, allows the designer to attain better throughput and power characteristics if the cores are optimally placed on the communication system [AHM-001].

This chapter begins with an introduction to the proposed platform based design [AHM-005], the proposed novel hybrid communication media is discussed, followed by the modelling of developed platforms [AHM-001]. A detailed analysis of implemented platforms is presented and its effectiveness is shown by a real life example [AHM-007] [AHM-009] [AHM-011].

4.2 Proposed Communication Centric Platform

As opposed to the traditional approach of platform based designs, discussed in chapter 2, where the flexibility of a platform instance, i.e. its capability of supporting different applications, is provided by programmable components, such as microprocessor or by reconfigurable logic blocks such as FPGAs. The proposed platforms have a built in controller that facilitate the integration of components in truly plug and play fashion that do not require any programming after the components are integrated in the system.

The proposed platforms can be divided into three major parts as shown in Figure 4-1: Platform controller, interfacing and integrated Communication medium (global bus, crossbar, hierarchical bus and a hybrid medium).

The components and their implementation will be discussed later in this chapter. However, before proceeding further, let us consider an example to demonstrate the novelty of the proposed platforms. We want to develop a platform with 20 integrated cores, including a micro processor (e.g. a RISC based ARM processor), memory modules, UART, fixed processing cores and hardware accelerators. For simplicity, let us also assume that they support the interface needed to integrate them to the platform. The platform has integrated ports where the cores can be integrated in a plug-and-play fashion. Each port has an address that is used by the platform controller for referencing the connected cores. Not every core integrated needs to communicate with every other core. This simplifies controller's job and thus only the integrated core has to be notified of the address of the cores it would communicate with (discussed in details later in this section). Not having to modify the pre-verified platforms means that the system designer does not have to re-compile and verify the functionality of platform after integration of cores.

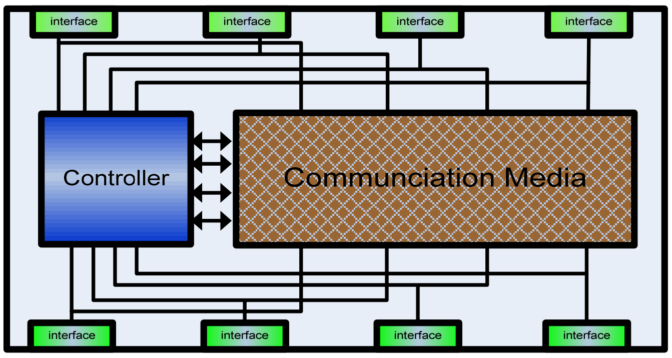


Figure 4-1 Block diagram of Proposed Platforms

The flow chart in Figure 4-2 shows the steps involved in the development and working of a system generated using the proposed platform based designs based on shared communication medium. The cores are integrated with the platform via the provided ports and the communication initiators (Master cores) are notified of the target cores they will be engaged in communication with (Slave cores). The system controller then resets the communication media and the cores connected by sending a reset signal. The system is now ready to work. The communication controller listens to request for data transmission from master cores. On receipt of a data transmit request, controller checks channel and slave availability and if available, it connects the master with the slave. During this communication, controller keeps a check on requests from other masters. Once the data transaction is completed, the controller closes that connection and declares the channel to be available for use by other cores. At this stage any pending master requests are dealt with by the controller. In the absence of any master request, controller goes back to waiting for request from any master. This system continues to operate till it is shut down or reset manually.

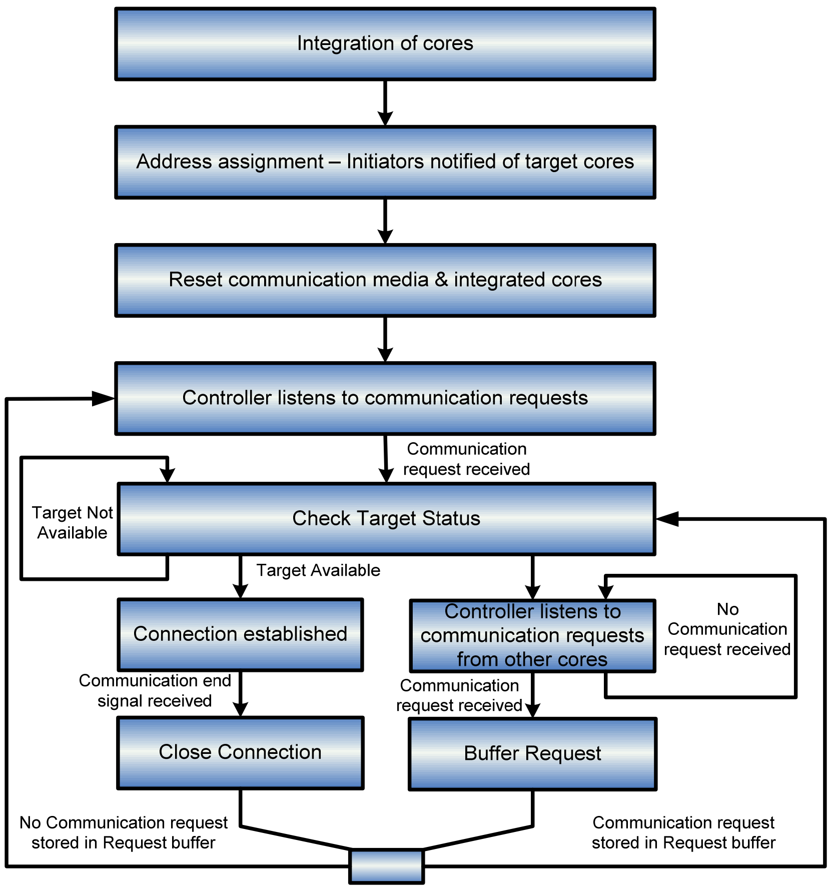

Integration of cores
Address assignment – Initiators notified of target cores
Reset communication media & integrated cores
Controller listens to communication requests
Communication request received
Target Not Available
Check Target Status
Target Available
Connection established
Controller listens to communication requests from other cores
No Communication request received
Communication end signal received
Communication request received
Close Connection
Buffer Request
No Communication request stored in Request buffer
Communication request stored in Request buffer

Figure 4-2 Working of proposed communication-centric platforms

4.2.1 Platform Controller

The platform controller is one of the most important parts of the platform and is responsible for assigning addresses to the attached cores, dealing with requests and grant signals and arbitration i.e. allocation of communication media to the core that requests communication. One of the important features of developed platforms is their reusable design and ability to deal with reconfigurable cores. This makes the platform controller design highly complex.

The platform controller can be divided into two parts as per its function: the system controller provides the reset control and power management framework for the SoC; the communication controller deals with the communication aspects of the platform. The communication controller is really a core in itself and handles the following functions:

- Establish communication sessions between communicating cores.

- Manage data communication over communication links by controlling the flow of data.
- Monitor data transmission request requests from master cores.
- Buffer incoming data transmission requests from master cores.
- Data arbitration.

Address assignment - As mentioned earlier, the proposed platforms support core integration in plug-and-play fashion and is designed for reuse. One of controller features supporting this is the address assignment. Ports are developed in the platform design for the integration of processing cores. Each port has an address that is used by the controller for communication and control. 4 bits are used in the developed prototype platforms in order to facilitate the integration of 16 cores in the system. Figure 4-3 shows a generic structure of platform with address assigned to ports.

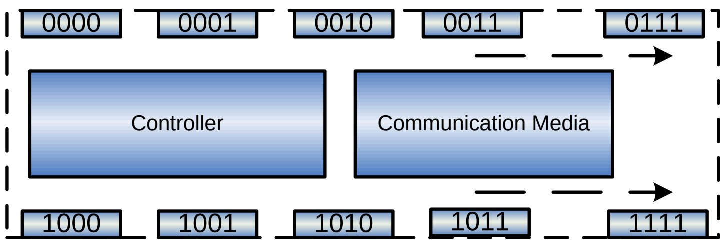


Figure 4-3 Port address assignment in the proposed platforms

Arbitration - Arbitration is performed by the controller. Control signals deal with the communication between the cores and the controller. Resource allocation can be done by any of the following methods; FIFO (First in, First Out), Round robin, Shortest Job First (SJF) and Priority based.

In FIFO, shared communication media is assigned based on order of requests. It is the simplest of allocation algorithm but is non pre-emptive, i.e. access cannot be blocked once granted and data transfer is complete, causing short jobs to get stuck behind the long jobs. Round robin on the other hand, releases the communication media from long running tasks based on timer interrupts so short jobs can get fair share of communication media. However, if the time slice is too long, scheduling degrades to FIFO and, if the time slice is too short, then the throughput suffers. In order to tackle this, the shortest-job-first algorithm was proposed where, whichever tasks requires the bus for the least time is granted the access first. It is ideal for short jobs and there is only a small performance degradation for long jobs. Another option is for priority based allocation where each request has priority associated and the task with highest priority gets the access first.

Although the pre-emptive scheduling algorithms promise to offer better performance they have a great disadvantage when employed for communication media access. When a task being pre-empted, the process is forced to leave its running state and get blocked. However during communication, this can cause starvation which is not the ideal case for communication media. Also for jobs of equal size, FIFO scheduling algorithm proves to be the best option. Thus for the prototype platforms, FIFO based arbitration is used.

4.2.2 Interfacing

To allow for wider connectivity, all the created platform ports are given a standard socket interface. A socket is universal and is targeted for use in virtually any application, while a bus interfaces e.g. AMBA, is targeted at a unique application, where all of the arbitration logic and interface circuitry is defined for that particular application. Consequently, with the change in application design, all of the arbitration logic and interface circuitry needs to be taken apart and re-designed for the new application. However, a socket can be targeted for any given application. This promotes design reuse and also aids the verification problem. OCP [OCP-001] is chosen as the socket standard in the proposed designs.

4.2.3 Communication media

The communication media considered in the developed platforms include bus based, crossbars, hierarchical and hybrid media. This section lists the basic concept of the communication media in platform. Details will be discussed in the implementation section. Buses have been deployed for communication since the beginning of circuit design and made their way in SoC due to their well understood concepts, their compatibility with most of the available node processors, the area taken on the chip and the zero latency after the arbiter has granted control. In the Global Bus architecture, a single bus is shared by all the components in a system. An arbiter is used to grant access of the bus to the core. The access is granted in FIFO manner. Bus bandwidth is parameterisable, so depending on the application, the bus architecture can be configured to meet the desired requirements. Figure 4-4 shows the bus based platform providing interface for 8 cores.

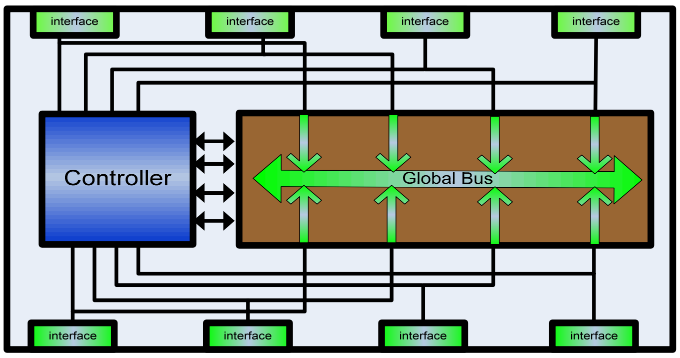


Figure 4-4 Bus Based Platform

The second implementation scenario considered is using a crossbar switch. A crossbar switch is capable of channelling data between any two core modules that are attached to it up to its maximum number of ports. The paths set up between cores can be fixed for some duration or changed when desired and each core-to-core path (going through the switch) is usually fixed for some period.

Crossbar switch fabric offers a major advantage over shared buses, as the traffic between any two modules connected via crossbar increases, it does not affect traffic between other modules. In addition to offering more flexibility, a crossbar switch environment offers more scalability in terms of integration of cores than a bus environment [WIJ-001].

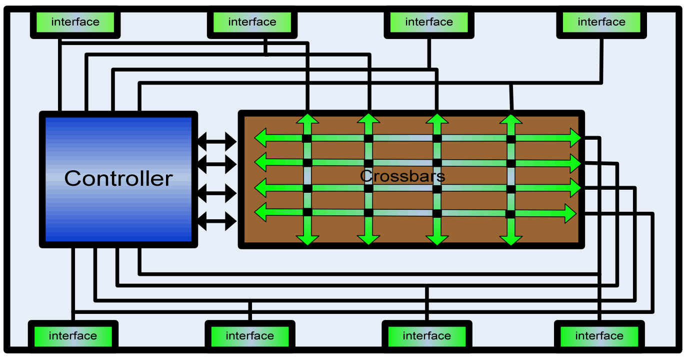


Figure 4-5 Crossbar Based Platform

As with a bus based platform, a crossbar based platform is parameterisable The Figure 4-5 shows crossbar implemented platform. FIFO arbitration is used in the designed platform. The simplest arbitration performed is in the case of bus based, where access is granted to the core that requests for it first. Once the communication is ended, bus access is granted to the other cores. During the communication, if any other core requests bus access, the request is stored in the request buffer and bus access is granted on basis of first come first serve to the next in line core. Arbitration in crossbar is based on the same principle as the bus, apart from the difference that the presence of bus matrix in crossbar medium requires arbitration of each channel separately. Hence, arbitration is done for four channels in the prototype.

Chapter 2 and 3 list a lot of work carried out in the field of on-chip communication from bus based systems to complex NoCs. However no work is done where two different communication structures with different power and throughputs co-exist in same SoC. The term “hybrid communication” is so far limited to existence of two different topologies in a same communication medium. E.g. in [HUA-001] a hybrid interconnect structure has been proposed which takes advantages of both mesh and tree topologies. As part of this thesis, a hybrid communication medium is developed combining the advantages of bus based systems with crossbar based systems, thus, both the bus and crossbar co-exist in a single system [AHM-001].

The third scenario considered is a hybrid structure that utilises crossbars and a global bus in one SoC. As previously investigated in [AHM-001] and by R. Huang and R. Vemuri [HUA-001] it can be concluded that hybrid interconnect can significantly reduce the routing area and achieve high performance. The aim of a hybrid communication medium based platform is to exploit the advantages of hybrid interconnects in a system and utilise its effectiveness for reconfigurable SoC architectures. Figure 4-6 shows the hybrid platform architecture. Implementation, simulation results of hybrid medium and its comparison with traditional communication media is shown later in the chapter.

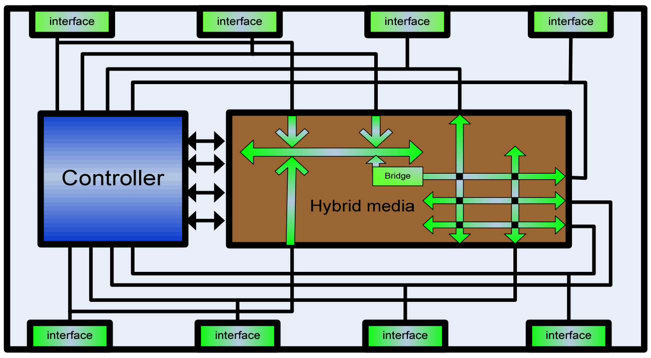


Figure 4-6 Hybrid communication medium based Platform

The fourth scenario considered is a hierarchical structure where two buses are combined by a bridge. It is based on the concept of the AMBA bus [ARM-001]. The advantage of this placement lies in the availability of two buses that can be used at different speeds by different masters. The bridge acts as a way to communicate between the two buses. A hierarchical bus based platform is shown in Figure 4-7.

The arbiter in hierarchical bus and hybrid medium can be divided into two parts, each part dealing with one section i.e. in case of hierarchical bus, one section of bus is dealt with by one part of arbiter and other section by the other arbiter part, similarly, In case of hybrid, arbitration in crossbar and bus section is dealt separately. However, if communication is requested between the cores that exist on the different sections, then the part of arbiter dealing with the section of communication media where the requester exists checks with the arbiter part dealing with the other section through the bridge, to check the availability of the requested core/medium for connection establishment (discussed in section 4.4).

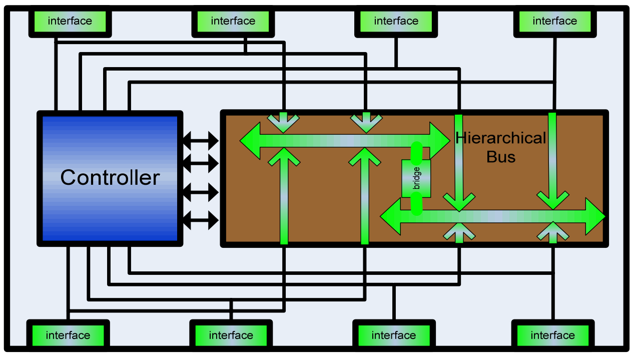


Figure 4-7 Hierarchical bus based Platform

4.3 System Level Modelling

Conventionally, functional verification follows the RTL coding stage in the system design process. The design errors exposed at this stage can cause changes to the architecture and hence, causing re-coding the RTL. This iterative process may repeat several times before the design can be considered safe. To tackle this time consuming task, higher level of abstractions called system level modelling is used [KEU-001].

In system level modelling, higher level modelling languages like c/c++ [RAH-001], systemC [GRO-001], specC [GAJ-001] or SystemVerilog [HAN-001] gives an estimate of system characteristics early in design flow. Communication architecture exploration can be performed at several different levels of abstraction.

Register-Transfer Level: RTL also referred to as cycle accurate modelling is the lowest level of modelling and can be classified as RTL model. In RTL models system components and the communication architecture characteristics are captured at a cycle and pin accurate level. It can be argued that RTL cannot be classified as modelling but it will be unfair to leave it out of the modelling paradigm. These models offer only a little speedup over the RTL models [YIM-001].

Transfer level: Transfer level systems area characterised by cycle true behaviour. They behave the same as the corresponding RTL system and follow the communication protocol fully. The behaviour inside component need not be scheduled at every cycle boundary which allows rapid system prototyping and considerable simulation speedup over RTL. It can be argued that the transfer level functionality can be achieved in RTL, however, transfer level offers a simpler netlist, i.e. only a single wire for the whole communication interface. Its main uses are to model cycle accurate test benches, cycle accurate performance simulation and for comparison with RTL.

Transaction Level: These models make use of high level interface functions with a few signals to maintain bus cycle accuracy, for example, read/write interface functions. These are timed but not cycle accurate and the implemented system is event driven. The simpler interface reduces modelling effort and the function call semantics resulting in fast simulation speeds [PAS-002].

Message Layer: Message Layer Models are very high level bit accurate models that replace the high level specifics with functions like read() and write (). These models are un-timed and the system implemented is event driven. Message Layer models are extremely fast to simulate and can be used to gain a very high level estimate of data traffic between components for communication system exploration. This level is also useful for algorithmic performance, behaviour and control statistics.

4.3.1 Proposed Modelling Level

In order to analyse performance and to help identify bottlenecks, a new modelling abstraction has been proposed for communication centric platform modelling. The proposed modelling layer lies between transfer level and transaction level providing cycle accurate models compatible with read() write() function and signals to maintain bus cycle accuracy. Clock cycle accuracy allows accurate communication space exploration and modelling at the boundary of the transaction layer provides faster simulation times.

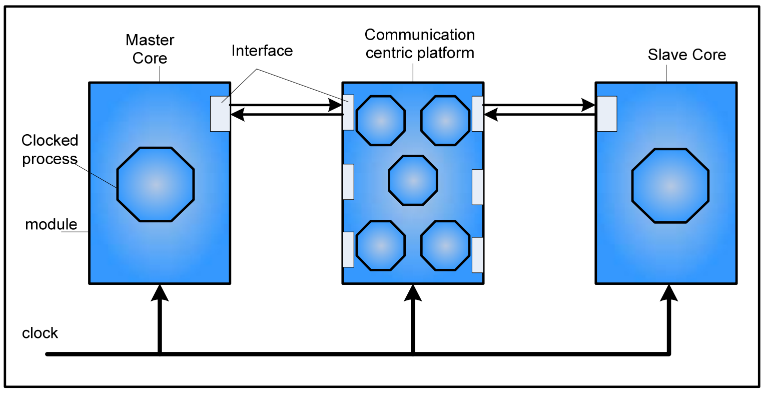


Figure 4-8 A system showing the proposed modelling level

The system thus implemented follows a specific communication protocol in order to explore its characteristics and for system verification. In order to increase the compatibility of proposed platforms an OCP socket based standard is adopted.

Data Interface - The data transfer supports passing along pointers as well as explicitly transferring the data to the channel. The channel contains a pointer that is shared between the master and the slave. The data pointer is set to the internal buffer in the master that is transmitting the data. In case of read request, the pointer is pointed to the buffer in the slave. In case of the data copy method, the data has to be copied to and from the channel. A mixture of both is also possible where one copies and other points to the buffer in during the same initiated communication.

Control interface – The control interface is needed for the synchronisation and sequencing mechanism used with the data interface to complete the transaction. The control flow is monitored closely by the controller. As it is cycle driven, the control signals can only be called at clock edges. The channel is implemented as clocked and stores state information between clock cycles. Some of the synchronisation mechanisms supported include,

- A request mechanism that the master core has data available to transmit.
- A response notification that the slave core has data for the master core.
- An acknowledgement that the master core has processed the data.

- Acknowledgement that the slave core has processed the data.
- Flag to show that the slave/master core is busy in some other transaction.

4.3.2 Programming Language Choice

As mentioned above, many high level modelling languages have appeared on the SoC design canvas in the past decade e.g. SystemC, SpecC, SystemVerilog etc. systemC is chosen as a preferred modelling language in the proposed platform based designs. SystemC is a set of library routines and macros implemented in C++ which makes it possible to simulate concurrent processes, each described by ordinary C++ syntax. Getting its basis from C++, SystemC offers object oriented design partitioning and template classes making it not only a hardware description language but a simulation kernel for rapid simulation of SoC architectures [OSC-001]. SystemC supports TLM, HW/SW co-design providing grounds for SoC architectural analysis and optimization. Modelling in systemC gives the high configurability needed to address issues like ports, arbitration, address management and control signals e.g. request and response etc.

4.4 Proposed Communication Centric Platform modelling

As mentioned above, there are three main parts of the platform: platform controller, communication media and interfaces. In this section, the implementation of each of these sections will be discussed. Figure 4-9 shows a generic model of the proposed platform highlighting its three important parts.

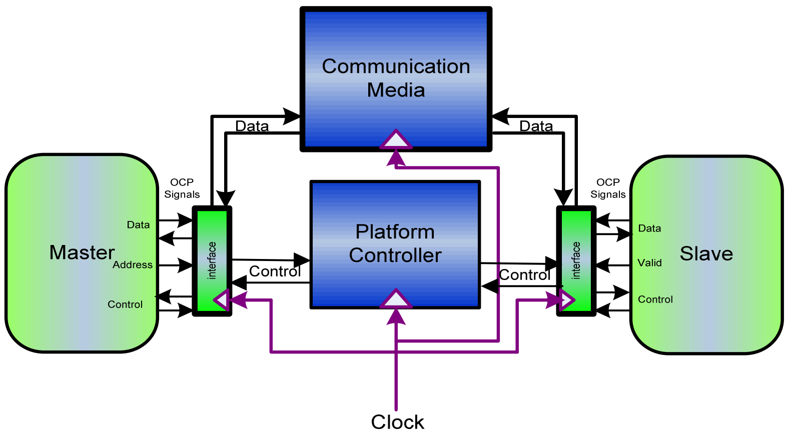


Figure 4-9 Generic model of the proposed platform

In Figure 4-9 a cross section of the platform with a master and slave core connected to the platform controller and communication media through the interface is shown. The OCP socket standard is used as interface to make the integration of a broad range of components easier in the platform. The Clock signal is sent to the master and slave through the interface thus making a single clock to the platform distributed to all the components of the system. The platform controller works differently for different communication media, however, the overall design concepts remain the same.

The proposed platforms follow OCP signals; there are three important type of signals for a master core, the data flow that can be 8-, 16- , 32- and 64- bit, the address signal (replaced by data valid Signal in slave core) and the control signals. Table 2 outlines the signal API. Signal names have been changed to keep the design simple to understand. Signals starting with “m” and “s” are originating or destined for master and slave cores respectively. The character “x” in the signal name refers to the master or slave port connected to the core. For example, the signal “m_req_x” refers to the signal from master connected to port “x” of the platform.

Table 4-1 Platform interface signals

M_req_x	: Request signal from Master to the controller. Possible signals are ,
M_REQUEST	: Master requesting bus access
REQ_END	: Master informing controller about end of communication channel.
M_addr_x	: address of slave . possible signals are
SLAVE_1	: Slave 1 address
SLAVE_2	: Slave 2 address
SLAVE_3	: Slave 3 address
SLAVE_4	: Slave 4 address
ADD_AVAIL	: LSB of m_addr_x
Clk	: clock signal.
reset	: reset the controller.
M_x_out_control	:Control signal from master to slave
M_x_databus	:Databus between master and slave
S_x_out_control	:Control signal from slave to master
S_x_databus	:name of data bus signal for slave
S_status_x	: status of slave, possible signals are
S_FREE	: slave available for communication
S_BUSY	: slave busy in communication
M_ack_x	: signals from controller to master.(replies of “req” above”. Possible signals are,
REQ_ACK	: request acknowledge
ADD_ACK	: address is received
ACK_GRANT	: bus is available for communication
M_x_in_control	: Control signal from slave to master
S_x_in_control	: Control signal from master to slave

Figure 4-10 outlines the step by step procedure involved in the establishment of connection between a master and slave core. Master first requests the controller to check the availability of the slave device. Once the slave is free and ready to communicate, the arbiter gives control back to the master to directly communicate with the slave. Just to clarify, the term master refers to communication initiators and slaves are the target devices that can not initiate communication.

After the connection is established between master and slave, control signals are directly exchanged between the two cores. Once the master is done with reading/writing data to the slave, it sends an “end communication” signal to the arbiter, which then releases the communication medium and sets the status of slave available to accept communication.

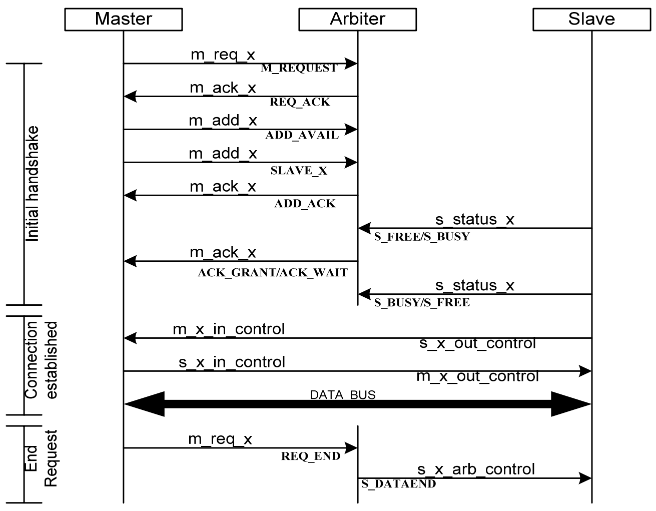


Figure 4-10 Connection establishment steps

In order to implement a cycle accurate model of the platform, an FSM based approach is taken. Signals are monitored at every clock cycle to determine the next state of the FSM. Figure 4-11 shows the FSM used for arbitration of bus and crossbar based communication media. Arbitration is FIFO based and the controller keeps track of the requests while it is dealing with one request. These requests are stored in a request buffer. Once the arbiter is done dealing with the current request (i.e. at the end of communication), it checks the request buffer for a pending transaction request, in the case of an empty buffer, the arbiter starts monitoring the REQ signals from master cores.

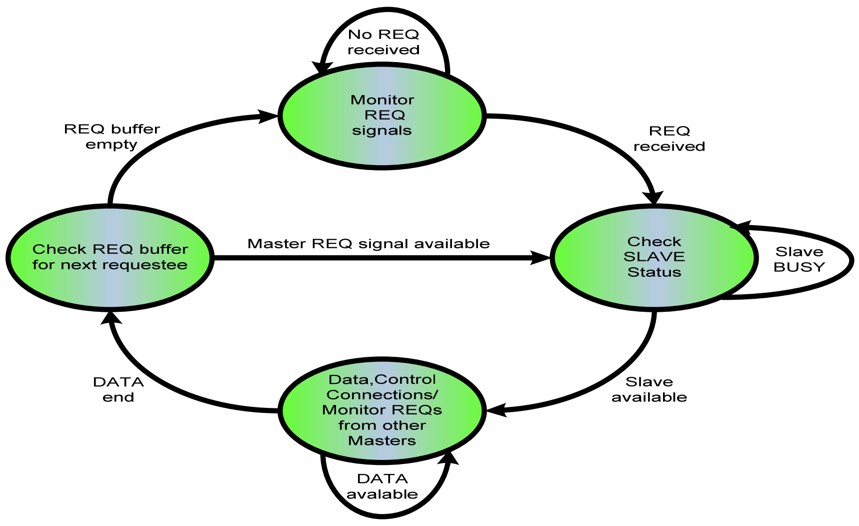


Figure 4-11 FSM of arbiter for bus/crossbar

In the case of hierarchical bus and hybrid communication, the platform controller can be thought of as composed of two parts. One controller to deal with one bus and second controller to deal with other bus in case of hierarchical bus based media. In case of hybrid media, one controller deals with the bus part and second with the crossbar part. However, when a master requests to communicate with the slave located on the other bus or on the second communication media in case of hybrid, then the bridge approach is used. The concept of bridge is shown in the Figure 4-12.

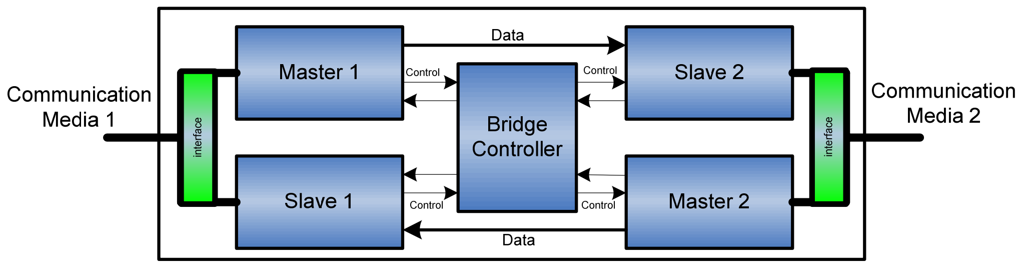


Figure 4-12 Bridge between two communication media

The bridge acts as a slave on the bus where the master is requesting to transmit the data and acts as a master device on the communication part where the slave exists. This simplifies the process of bus allocation process. Once the access is granted, the bridge simply transfers the control signals and connects the data communication lines. The arbitration flow diagram is shown in the Figure 4-13. The arbitration works in a similar fashion as in the case of shared bus and crossbar based media apart from the difference when a master requests for communication with a slave across the bridge.

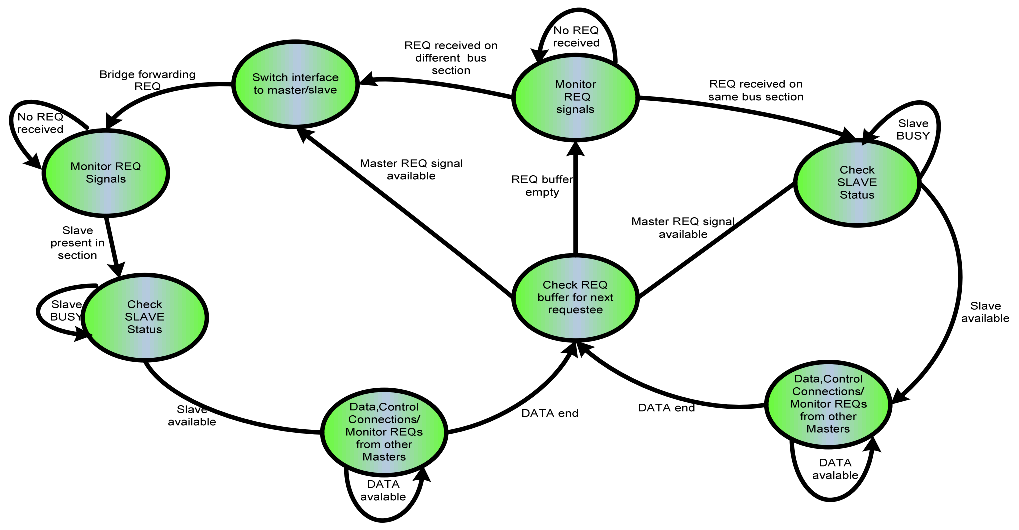

No REQ received
Monitor REQ signals
REQ received on different bus section
Switch interface to master/slave
Bridge forwarding REQ
No REQ received
Monitor REQ Signals
REQ received on same bus section
Slave BUSY
Check SLAVE Status
Master REQ signal available
REQ buffer empty
Master REQ signal available
Slave present in section
Slave BUSY
Check SLAVE Status
Check REQ buffer for next requestee
Slave available
Slave available
Data,Control Connections/ Monitor REQs from other Masters
DATA end
DATA end
Data,Control Connections/ Monitor REQs from other Masters
DATA avalable
DATA avalable

Figure 4-13 FSM of arbiter for hierarchical bus/hybrid communication media

4.5 Simulation Results

Before simulating the developed platforms for throughput comparisons, simulations were first carried out to verify the transaction behaviour of the developed communication media. The section below lists the test scenarios and the working of the platforms under simulated traffic.

4.5.1 Verification of Bus based platform

Scenario 1: one master core communicating with one slave core

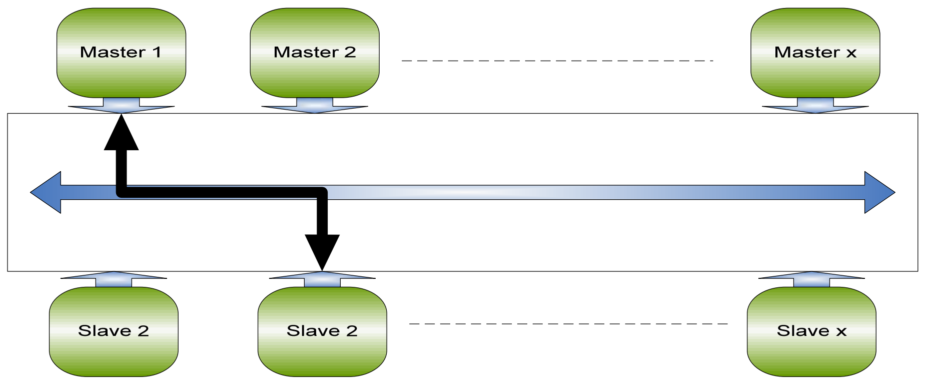


Figure 4-14 Scenario 1 - one master core communicating with one slave core

In order to verify the bus system, the first scenario considered was of one master core trying to communicate with one slave core. Figure 4-14 shows the scenario where master 1 is communicating with slave 2. The result of this simulation and other simulations in this section are obtained as a binary file and later displayed as timing graph.

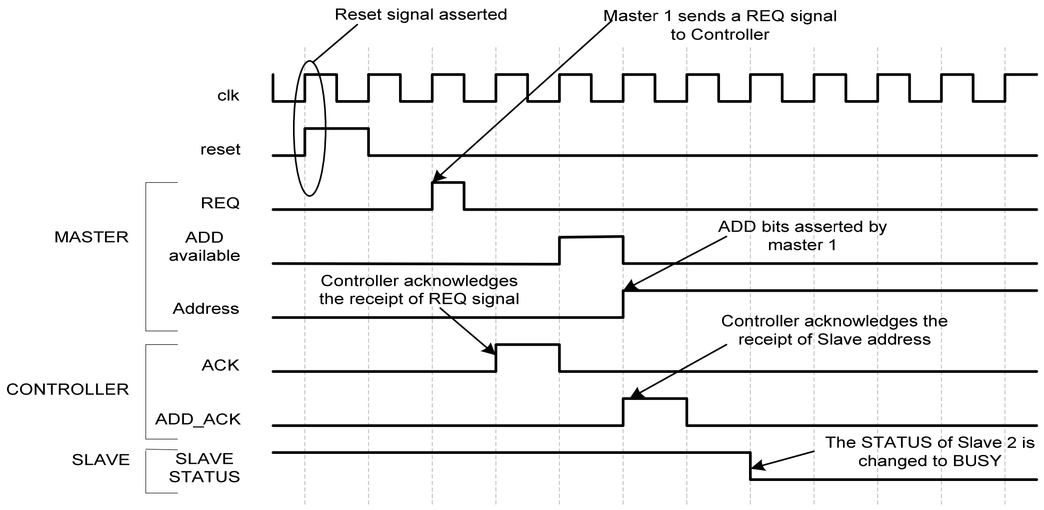

Reset signal asserted
Master 1 sends a REQ signal to Controller
clk
reset
REQ
MASTER
ADD available
ADD bits asserted by master 1
Controller acknowledges the receipt of REQ signal
Address
Controller acknowledges the receipt of Slave address
ACK
CONTROLLER
ADD_ACK
SLAVE
SLAVE STATUS
The STATUS of Slave 2 is changed to BUSY

Figure 4-15 Bus scenario 1 Transaction behaviour

The simulation begins by asserting the reset signal manually; this resets the platform controller, master, slave and the communication media. Whenever a master requests to get access to the communication medium, it has to send a REQ signal to the controller. The controller acknowledges the REQ signal by sending an ACK signal back. Master then sends the ADD available signal and sends the address of target (slave) core. Controller checks the slave status, as slave is free; controller sets slave and master status to busy establishing the communication between them. Connection establishment means, connecting master and slave data buses to shared bus and connecting master and slave control signals.

Scenario 2: two master cores communicating with one slave core

The second scenario considered is of two bus masters trying to communicate with one Slave core. In the Figure 4-16, master 1 and master 2 want to communicate with slave 2. In case of more than one request for a shared medium, the controller has to do more than one task when granting access of shared resource to a master.

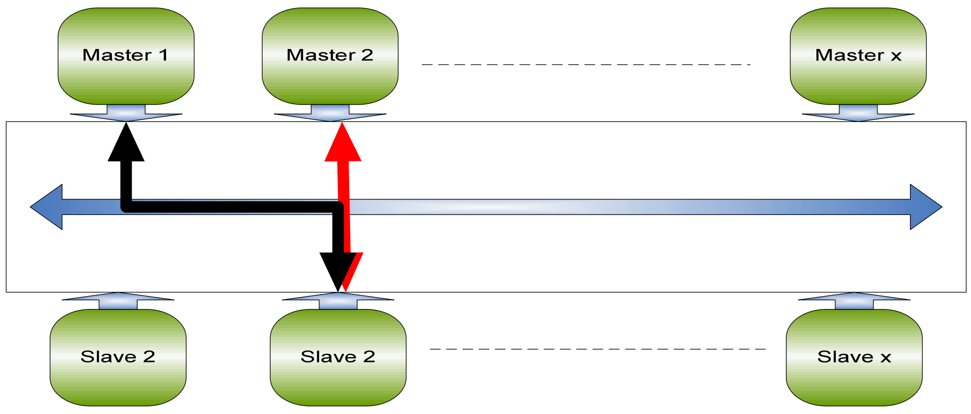


Figure 4-16 Scenario 2 - two master cores communicating with one slave core

- Keep note of the first master that requested.
- Check to see if slave device requested is free.
- Keep track of all the other requests that the controller receives while processing the request of the one already getting dealt with.
- Monitor the communication in progress to end.
- Once the communication medium is free, the controller than takes the next request in the FIFO buffer and grants access to the master provided the requested slave is free.

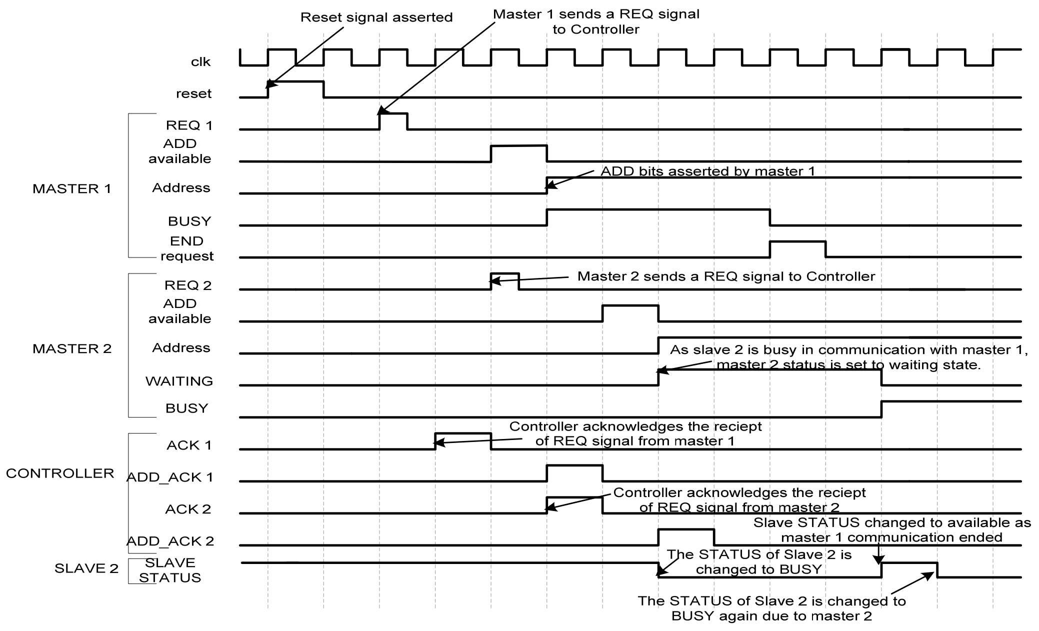


Figure 4-17 Bus scenario 2 Transaction behaviour

Figure 4-17 shows the result of simulation in case of scenario 2, Master 1 requests to communicate with slave 2, as the slave status is set to be available; the controller grants shared bus access to master 1 for communication with slave 2. While the controller is dealing with master 1 request, master 2 sends a request for transmission of data as well. Being a shared medium, only one core can transmit data at any one time, thus, the controller acknowledges the requests of master 2, however, a wait signal is sent to master 2 setting its status to waiting. Once the communication of master 1 has ended, the controller changes the status of slave back to being available and checks for the target address from master 2. In the scenario considered, it is slave 2, thus, the controller grants bus access to master 2 and sets master 2 and slave 2 statuses to busy.

Scenario 3: 2 master cores communicating with 2 different Slave core

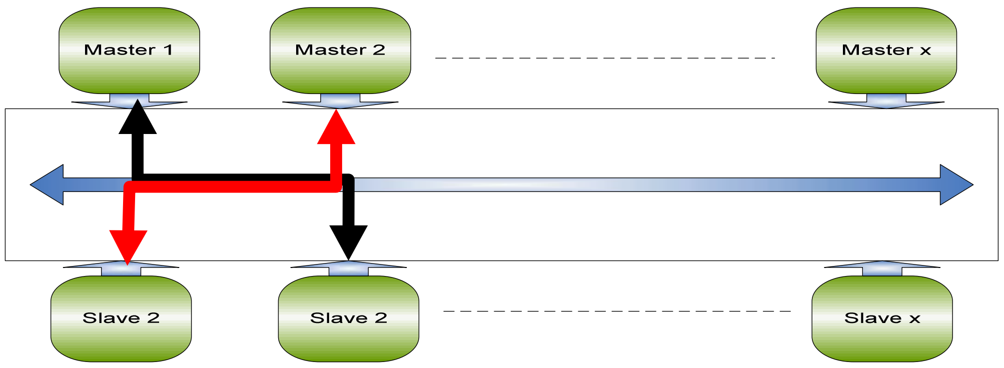


Figure 4-18 Scenario 3 - two master cores communicating with two different slave cores

In the third scenario, two master cores request for bus access to communicate with two different slave cores (see Figure 4-18). Assuming the request order of scenario 2, master 1 will get bus access first and master 2 will have to wait till the shared bus is free to be used for communication by master 2. Although this scenario looks different from scenario 2, however, as it is a shared bus based platform, access can only be allocated to one core at one time. Thus the system reacts the same way and shows similar transaction behaviour.

4.5.2 Verification of crossbar based platform

Scenario 1: Two master communicating with one slave core

In case of a 4x4 crossbar switch, the controller has four buses to allocate for communication. This makes the task of the controller more complex than for the simple shared bus medium. In case of only one master requesting to transmit data to a slave, the controller acts in the same way as in case of scenario 1 of bus based system.

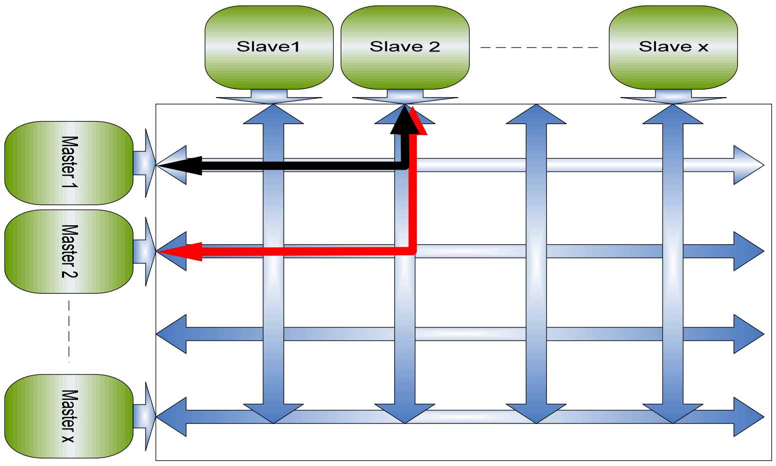


Figure 4-19 Scenario 1 - two master cores communicating with one slave core

When more than one master requests communication medium access, it becomes more complicated. First scenario considered in case of crossbar was of two master devices, master 1 and master 2 requesting to communicate with slave 2. Figure 4-19 shows the particular scenario. Although, the controller has possibility of allocating different buses to masters, due to the slave being the same, resource allocation depends entirely on the status of slave 2.

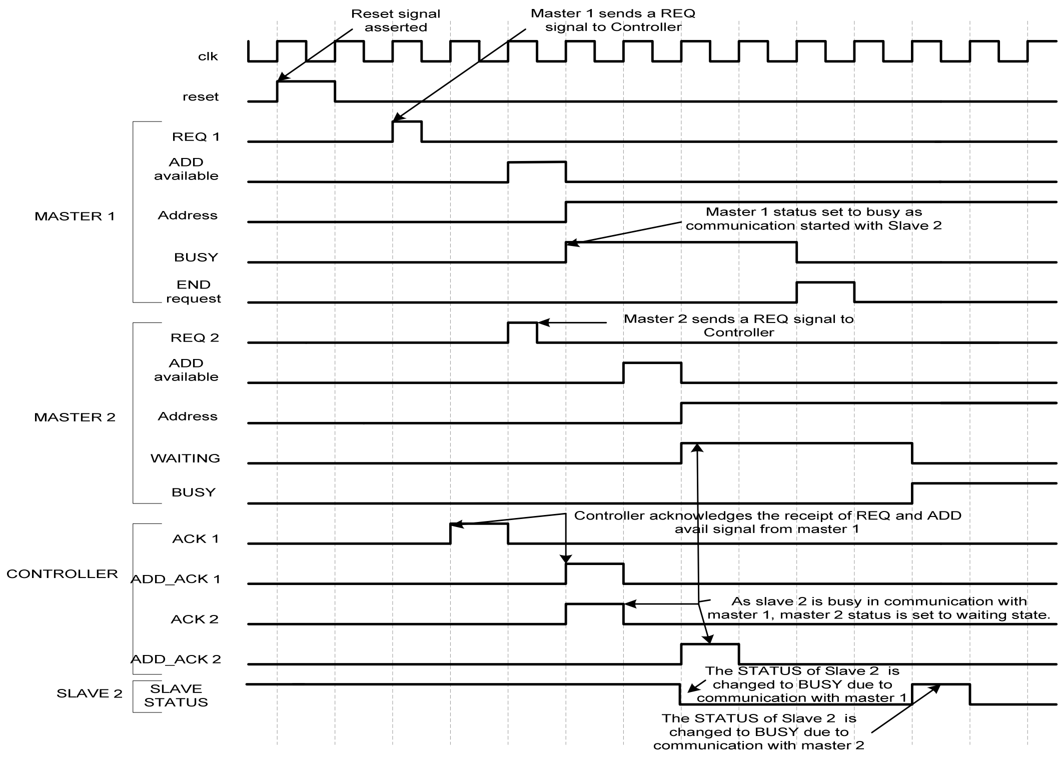

Reset signal asserted
Master 1 sends a REQ signal to Controller
clk
reset
MASTER 1
REQ 1
ADD available
Address
BUSY
END request
Master 1 status set to busy as communication started with Slave 2
MASTER 2
REQ 2
Master 2 sends a REQ signal to Controller
ADD available
Address
WAITING
BUSY
CONTROLLER
ACK 1
Controller acknowledges the receipt of REQ and ADD avail signal from master 1
ADD_ACK 1
ACK 2
As slave 2 is busy in communication with master 1, master 2 status is set to waiting state.
ADD_ACK 2
SLAVE 2
SLAVE STATUS
The STATUS of Slave 2 is changed to BUSY due to communication with master 1
The STATUS of Slave 2 is changed to BUSY due to communication with master 2

Figure 4-20 Crossbar scenario 1 transaction behaviour

The transaction behaviour in Figure 4-20 shows the scenario under consideration. The controller receives master 1 request for communication first, thus the crossbar path is allocated to master 1 for communication with slave 2. The controller in crossbar platforms differs in working from that of bus based platforms since in crossbar there are three remaining communication paths that the controller can allocate for communication. However, in this particular scenario, master 2 requests for slave 2. Status of slave 2 is set to busy as it is in communication with master 1. Thus, status of master 2 is set to waiting till the slave 2 status becomes available.

Scenario 2: two master communicating with two slave cores

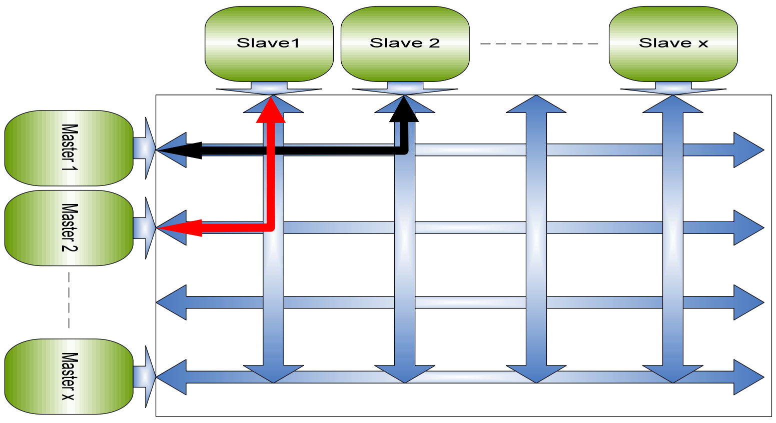


Figure 4-21 Scenario 2 - two master cores communicating with two different slave cores

Crossbar connection can also be thought of as point to point links between the cores in the system. In order to verify working of crossbar when two different masters request to communicate with two different slaves. A scenario is considered where, master 1 requests to communicate with slave 2 and master 2 requests to communicate to slave 1 (Figure 4-21). In this particular scenario, the controller can allocate the crossbar resources to both the masters for communication.

Figure 4-22 shows the signals involved in scenario 2. Master 1 requests to transmit data to slave 2 and as slave 2 status is set to be available, the controller allocates a path to master 1 to communicate with slave 2 core. In during this transaction, master 2 requests controller for access to slave 1. Again as the slave 1 status is set to be available, controller allocates a path to master 2. The crossbar platform controller monitors requests from all the masters, and monitors all the communication in progress simultaneously making its design more complex in nature.

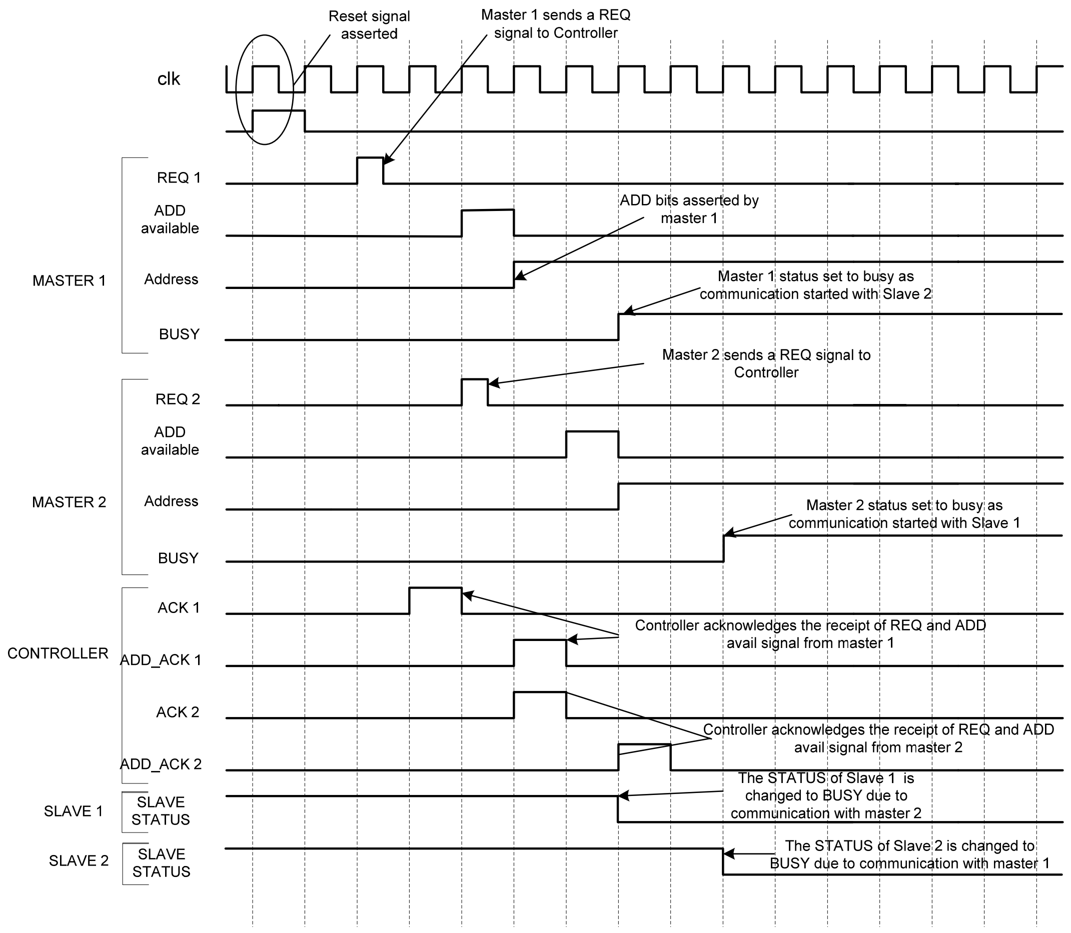

Reset signal asserted
Master 1 sends a REQ signal to Controller
clk
REQ 1
ADD available
Address
BUSY
MASTER 1
ADD bits asserted by master 1
Master 1 status set to busy as communication started with Slave 2
Master 2 sends a REQ signal to Controller
REQ 2
ADD available
MASTER 2
Address
BUSY
Master 2 status set to busy as communication started with Slave 1
CONTROLLER
ACK 1
ADD_ACK 1
ACK 2
ADD_ACK 2
Controller acknowledges the receipt of REQ and ADD avail signal from master 1
Controller acknowledges the receipt of REQ and ADD avail signal from master 2
SLAVE 1
SLAVE STATUS
The STATUS of Slave 1 is changed to BUSY due to communication with master 2
SLAVE 2
SLAVE STATUS
The STATUS of Slave 2 is changed to BUSY due to communication with master 1

Figure 4-22 Crossbar scenario 2 transaction behaviour

4.5.3 Verification of Hierarchical and hybrid platforms

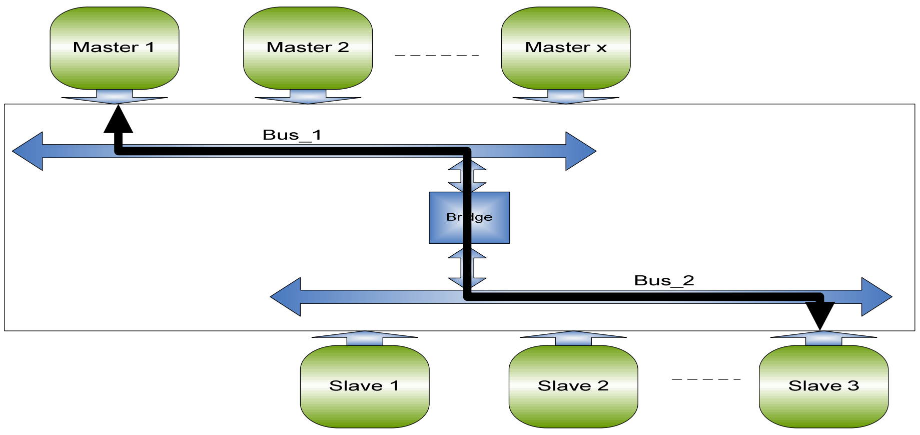


Figure 4-23 verification of communication across bridge

In case of hierarchical bus and hybrid communication media, if the communication request is for a slave core present locally i.e. on the same bus in case of hierarchical bus based platform and on either the same bus, or on crossbar fabric in case of hybrid media based platform, then the controller acts the same way as the above explained bus based and crossbar based platform scenarios. However, when the slave core requested by the master is not located locally then communication via a bridge takes place.

In order to verify the working of the bridge, an example of hierarchical bus is taken. The bridge design is same for both hierarchical bus based platform and hybrid media based platform. In the scenario considered, master 1 located on Bus_1 requests its local controller for access to slave 3 located on Bus_2. Figure 4-23 shows the considered scenario.

As mentioned in section 4.4 of this thesis, communication in this scenario has to go via the bridge. The bridge acts as a slave on bus_1 for master 1. Thus the controller 1 (master 1 local controller) will establish communication between master 1 and bridge. Bridge acts as a master on Bus_2 and requests the controller 2 (slave 2 local controller) for slave 2 access. Once the access has been granted, the bridge signals back to master 1 and sets the status of both master 1 and slave 3 to busy. Figure 4-24 shows the transaction behaviour of the considered scenario. In case the slave 3 is busy, the bridge will be set to waiting stage, which will be propagated back to master 1, setting its status to waiting as well.

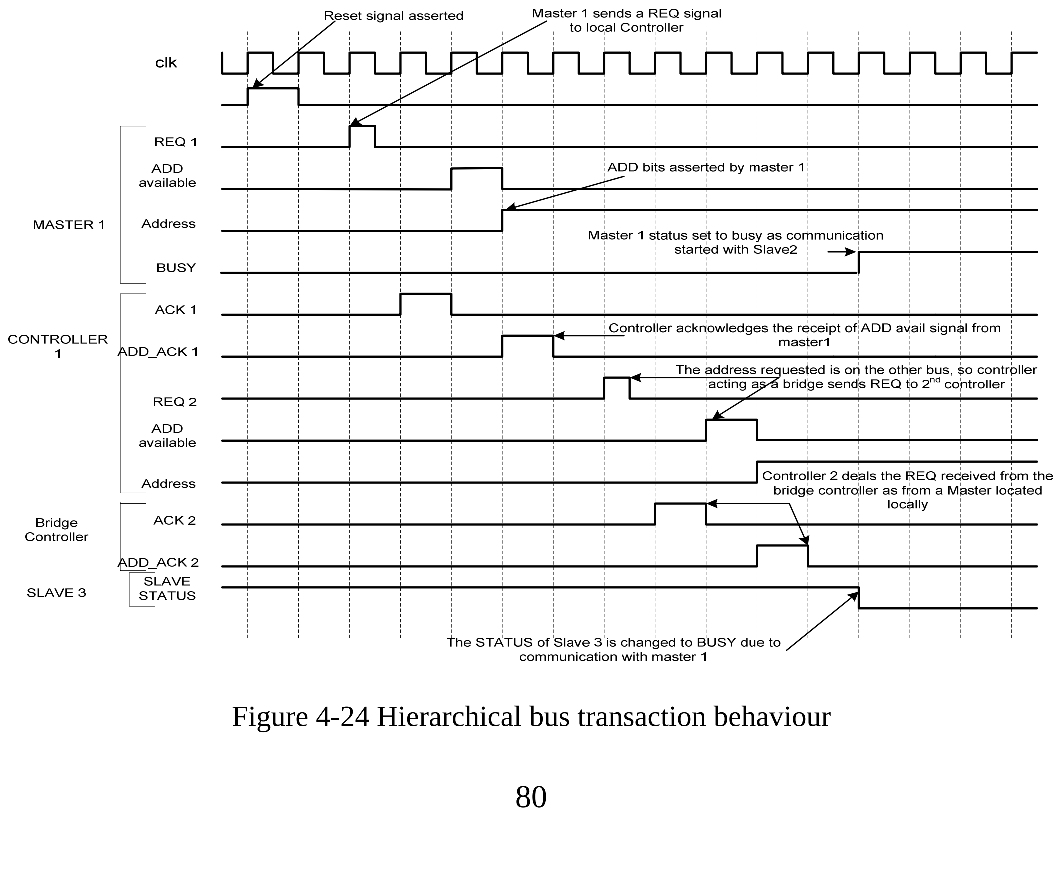

Reset signal asserted
Master 1 sends a REQ signal to local Controller
clk
REQ 1
ADD available
MASTER 1
Address
BUSY
ADD bits asserted by master 1
Master 1 status set to busy as communication started with Slave2
ACK 1
CONTROLLER 1
ADD_ACK 1
Controller acknowledges the receipt of ADD avail signal from master1
REQ 2
The address requested is on the other bus, so controller acting as a bridge sends REQ to 2nd controller
ADD available
Address
Controller 2 deals the REQ received from the bridge controller as from a Master located locally
Bridge Controller
ACK 2
ADD_ACK 2
SLAVE 3
SLAVE STATUS
The STATUS of Slave 3 is changed to BUSY due to communication with master 1

Figure 4-24 Hierarchical bus transaction behaviour

4.5.4 Throughput comparison

In order to further analyse the developed platforms, a series of simulations is carried out. The aims of simulations are to study the throughput characteristics of the developed platforms, to explore effect of data injection rate on the different communication media and to explore their ability to meet the timing constraints in form of delay to start the communication.

A system of four masters and four slave cores is considered. The placement of cores on the four developed platforms is shown in Figure 4-25. Different master-slave combinations are used in simulations to identify the strengths and weaknesses of the implemented platforms. Just to re-iterate, master cores are traffic generators that can produce different type of traffic patterns including the control signals to communicate with the controller. The Slave cores only accept and acknowledge the data. At this stage it is assumed that reading and writing data in a slave core takes the same time. Traffic patterns and master-slave pairs are described in the relevant experimental scenarios.

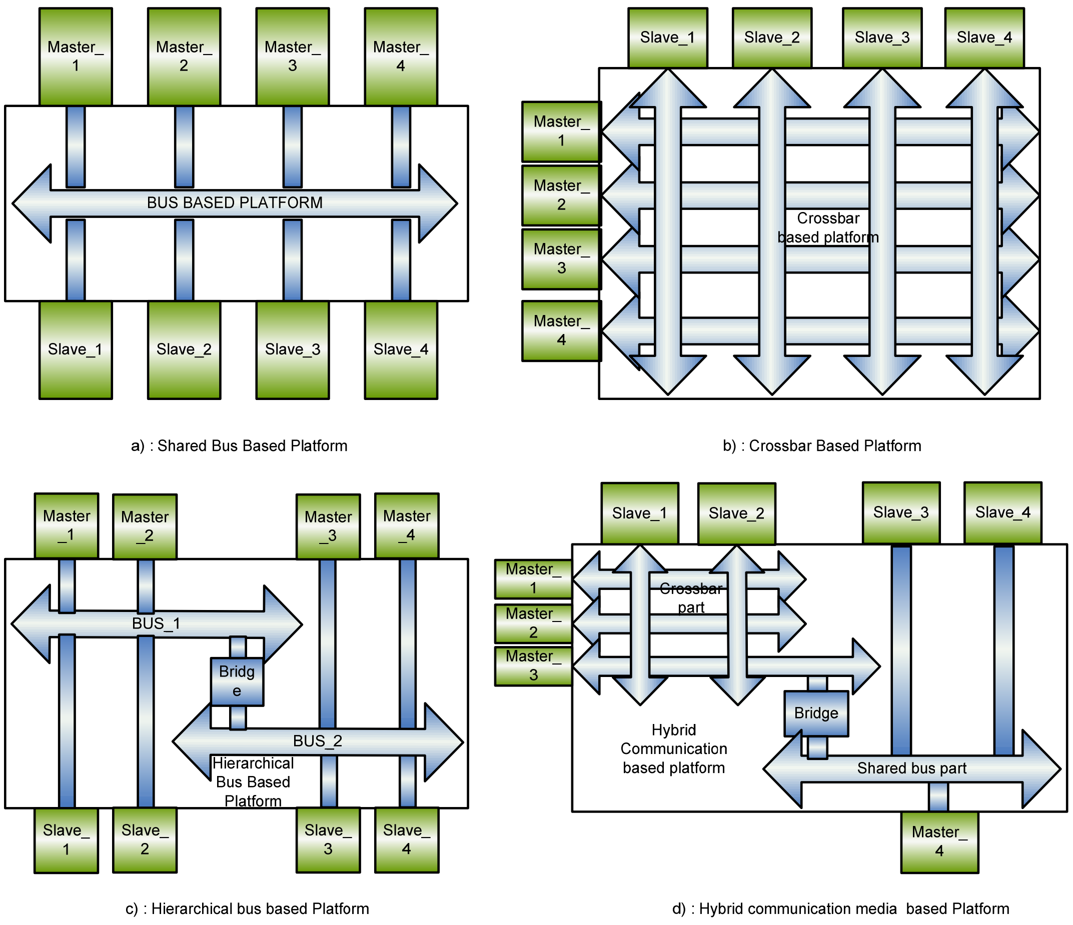


Figure 4-25 Placement of cores for simulations

Average throughput is calculated by using the equation [LAH-001],

$$Average\ throughput = (bus\ word\ size\ \times clock\ frequency\ )\ \times (\frac{No. of\ words\ to\ transmit}{wait\ cycles + transmission\ cycles})$$

Where, wait cycle is the number of clock cycles between the assertion of the REQ signal by a master core and the access granted signal by controller. Bus word size is 2 bytes or 16-bits and a clock frequency of 100 MHz is considered in simulations. Simulations are performed on 16-bit bus width.

Scenario 1: One master communicating with one slave core

The main aim of the first scenario is to evaluate the performance of the communication system when only one master core is communicating. The data packet generator is programmed to generate a traffic pattern where traffic bursts of 20 words are used. Two bursts per case are used at an interval of 20 clock cycles or 200 ns. Experimental setup of master-slave pairs are shown in Table 4-2. For hierarchical bus and hybrid medium based platforms, more than one simulation was conducted to study the effect of locality on communication parameters. In the case of hierarchical and hybrid medium, the simulations are also carried out to see the performance when both cores are connected on same communication medium i.e. master-slave pair on crossbar part and master-slave pair on bus part of the hybrid medium.

Table 4-2 Scenario 1 - Experimental setup

	Platform Type	**Initiator**	**Target**	
Case 1	Shared bus	Master 1	Slave 4	
Case 2	Crossbar	Master 1	Slave 4	
Case 3	Hierarchical	Master 1	Slave 2	Both cores on same bus
Case 4	Hierarchical	Master 1	Slave 4	Cores on different buses
Case 5	Hybrid	Master 4	Slave 4	Both cores on bus part
Case 6	Hybrid	Master 1	Slave 2	Both cores on crossbar part
Case 7	Hybrid	Master 1	Slave 4	Cores on different media

Figure 4-26 shows the average throughput of the master-slave cases considered in this experiment. It can be seen that without any contention on the communication channels and only one master-slave pair communicating on the platform, the throughput values are in the same range apart from the cases when master-slave pair do not exist locally and bridge is used i.e. in case of hierarchical and hybrid communication medium. Even then the throughput difference between the two cases is negligible for high data rate communicating cores.

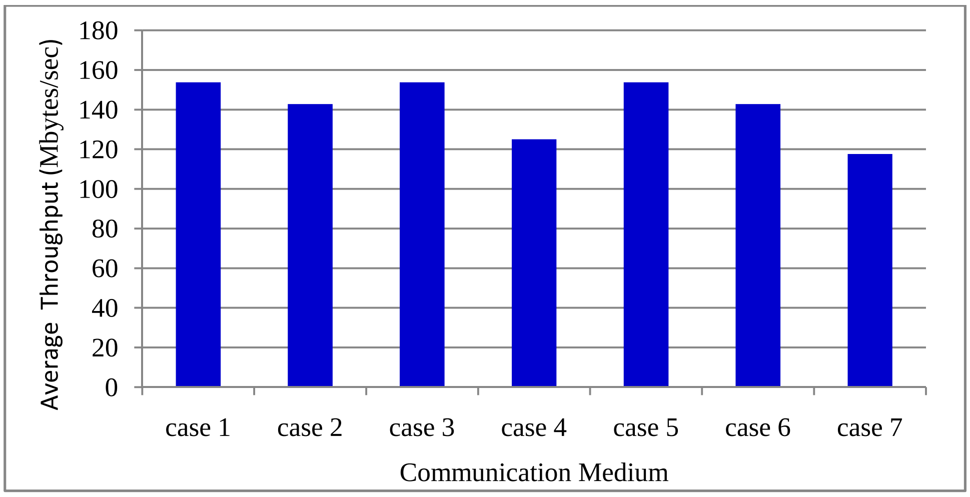


Figure 4-26 Scenario 1 - Simulation results

It can be seen from the figure 4-26 that the case 1, case 3 and case 5 gives the exact average throughput, this is because of the existence of both the master and the slave on shared bus medium, thus, handshaking (connection establishment) takes the same number of clock cycles. Similarly, the case 2 and case 6 gives the same results due to the master and the slave being on crossbar matrix. The decrease in average throughput in crossbar cases is due to the additional clock cycles utilised by the controller (arbiter) in handshaking.

In the case 4 and case 7, communication takes place via the bridge that connects the two different parts of the communication medium. To access the bridge, the master on the requester part of communication medium has to go through the handshaking stage. Once the connection with the bridge is established, the bridge acts as a master on the requested part of communication medium and goes through the handshaking stage again to gain access to the slave core. This additional effort causes loss of precious time, thus, degrading the average throughput of the communication medium. This scenario is a very simple example and shows that if only two cores are communicating without any other core competing for the communication medium, the different communication platforms act the same way as emulating a basic point to point link.

Scenario 2: Multiple masters with multiple slaves

The master-slave pairs are simulated for three different time interval patterns (data injection rate). The time interval patterns are the start times for the master cores to initiate communication. At the start time, the master core sends a request to gain access of communication channel. if the communication medium is available, the communication between the master and the slave core starts. However, if the channel is busy, the master core has to wait till the communication medium becomes available. Master-slave pairs and their start times under the three time patterns are shown in Table 4-3.

Table 4-3 Scenario 2 - master-slave pairs with expected start times

	Initiator	**Target**	**Start time for Time Pattern 1**	**Start time for Time Pattern 2**	**Start time for Time Pattern 3**
Link 1	Master 1	Slave 2	0 ns	0 ns	0 ns
Link 2	Master 3	Slave 3	0 ns	0 ns	0ns
Link 3	Master 1	Slave 4	100 ns after end of Link 1	200 ns after end of Link 1	300 ns after end of Link 1
Link 4	Master 3	Slave 1	100 ns after end of Link 2	200 ns after end of Link 2	300 ns after end of Link 2

The aim of this simulation is to study the average throughput of different communication media under different data injection rate. Data injection rate can be thought of as traffic load and is simulated by starting communication between links at different timings. In order to model realistic traffic pattern, the cores are made to request for data transmission in a random order.

Figure 4-27 displays the average throughput obtained by the four platforms under the three time patterns. It can be seen that crossbar based platform achieves the highest throughput under all the time patterns, followed by hybrid communication medium. This is due to the parallelism in communication system provided by crossbar medium, which allows the cores to communicate simultaneously.

In this simulation, the shared bus based platform unexpectedly outperformed the hierarchical bus based platform under time pattern 2 and time pattern 3. The hierarchical bus with two communication channels available was expected to achieve better throughput results than a shared medium. This performance downgrade can be narrowed down to the placement of the cores,i.e. their locality. The communication via the bridge in case of the hierarchical bus adds the additional handshaking overhead causing the decrease in average throughput. This will be further investigated in the following simulations.

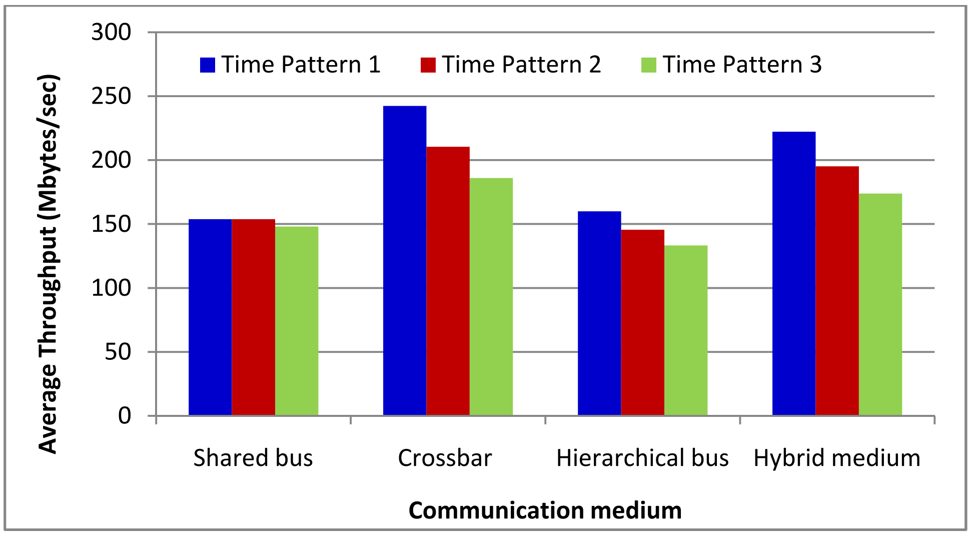


Figure 4-27 Scenario 2 - Simulation results

Scenario 3: Multiple masters with multiple slaves – scenario 2 with changed communication order

In order to further investigate the causes of degradation in performance of the hierarchical bus in scenario 2, the master-slave pair from scenario 2 was considered with the order of link 2 and link 4 switched. The master 3 is made to communicate with the slave 1 at the expected start time of 0 ns followed by the master 3 communicating with the slave 3 after the time delays as specified in the Table 4-4.

Figure 4-28 shows the simulation results for scenario 3. It can be seen that the performance of both hybrid medium and hierarchical bus has degraded. The performance of shared bus and crossbar medium remains the same as in scenario 2 and outperforms the hierarchical bus and hybrid medium. There are two important factors that cause this performance degradation.

Table 4-4 Scenario 3 - master-slave pairs with expected start times

	Initiator	**Target**	**Start time for Time Pattern 1**	**Start time for Time Pattern 2**	**Start time for Time Pattern 3**
Link 1	Master 1	Slave 2	0 ns	0 ns	0 ns
Link 2	Master 3	Slave 1	0 ns	0 ns	0ns
Link 3	Master 1	Slave 4	100 ns after end of Link 1	200 ns after end of Link 1	300 ns after end of Link 1
Link 4	Master 3	Slave 3	100 ns after end of Link 2	200 ns after end of Link 2	300 ns after end of Link 2

As discussed in scenario 2, the first factor is the locality of the master and the slave core on the communication medium. In case of the hierarchical and the hybrid medium, both the master and the slave cores have to communicate through the bridge which adds a handshaking overhead in terms of clock cycles. Secondly, the data injection rate (referred to as the time pattern in Table 4-4), which dictates the start of communication between the two cores, affects the availability of the communication channel. In case of the hierarchical and the hybrid medium, due to the placement of the cores, the bridge has to be used for communication between all the master-slave pairs. Thus, for a communication link to be established, it has to wait for prior communication between the cores to end. This causes a decrease in average throughput of the hierarchical and the hybrid communication medium.

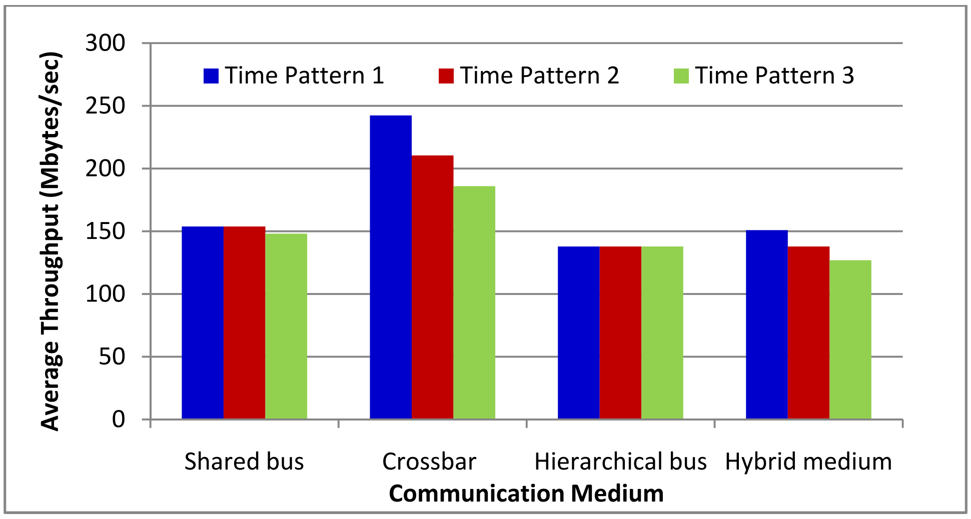


Figure 4-28 Scenario 3 - Simulation results

In order to further elaborate it, let us consider the communication scenario of time pattern 1. In case of a shared medium, after the initial platform reset, communication between the master 1 and the slave 2 begins, followed by the master 3 and the slave 1. Both these communication links should have been established at 0 ns, but due to the shared communication medium, only one link was established making other to wait. This delays the establishment of link 3 and link 4. In case of the crossbar medium, link 1, link 2, link 3 and link 4 start communication at the desired time due to the existence of parallel communication channels. The hierarchical bus medium and the hybrid medium also provide parallel communication channels, however, the cores are located at different parts of the communication medium, thus, and the bridge has to be used when establishing the communication path. The communication medium acts like a shared medium allowing only one communication path to be established via the bridge. The addition of handshaking overhead in connection establishment through the bridge causes the average throughput of hierarchical and hybrid medium to be less than that of the shared bus medium.

Scenario 4: Multiple masters with multiple slaves – optimised communication order for throughput

In scenario 2 and scenario 3, locality of the communication cores and the affect of traffic patterns were explored. It was discovered that the traffic patterns indirectly affects the availability of communication medium. Thus, depending on the locality of the communicating cores on the communication medium, traffic patterns influence the overall average data throughput.

The hierarchical bus and the hybrid communication medium provide parallelism in communication, however, it was seen in the above scenarios that they were outperformed by the shared bus medium. In order to find a scenario where all the communication medium are getting effectively utilised, the master-slave pairs used in scenario 3 are considered with changed communication order.

In this scenario, master 1 communicates with slave 4 and master 3 communicates with slave 1 at 0 ns (beginning of simulation). At the end of communication between these links, master 1 and master 3 communicate with slave 2 and slave 3 respectively after the time delays specified in table 4-5.

Table 4-5 Scenario 4 - master-slave pairs with expected start times

	Initiator	**Target**	**Start time for Time Pattern 1**	**Start time for Time Pattern 2**	**Start time for Time Pattern 3**
Link 1	Master 1	Slave 4	0 ns	0 ns	0 ns
Link 2	Master 3	Slave 1	0 ns	0 ns	0ns
Link 3	Master 1	Slave 2	100 ns after end of Link 1	200 ns after end of Link 1	300 ns after end of Link 1
Link 4	Master 3	Slave 3	100 ns after end of Link 2	200 ns after end of Link 2	300 ns after end of Link 2

Figure 4-29 shows the results of this simulation. It can be seen that both hybrid and hierarchical communication medium shows an increase in average data throughput. Average throughput of shared bus and crossbar still remains the same as in scenario 3. The hierarchal bus and hybrid medium also outperforms the shared bus. However, the crossbar medium still gives the best average throughput.

This simulation shows that for an optimised utilisation of resources, placement and communication order of the connected cores play an important role. Just by re-arrangement of communication order, the performance of the hierarchical bus is increased by 33%.

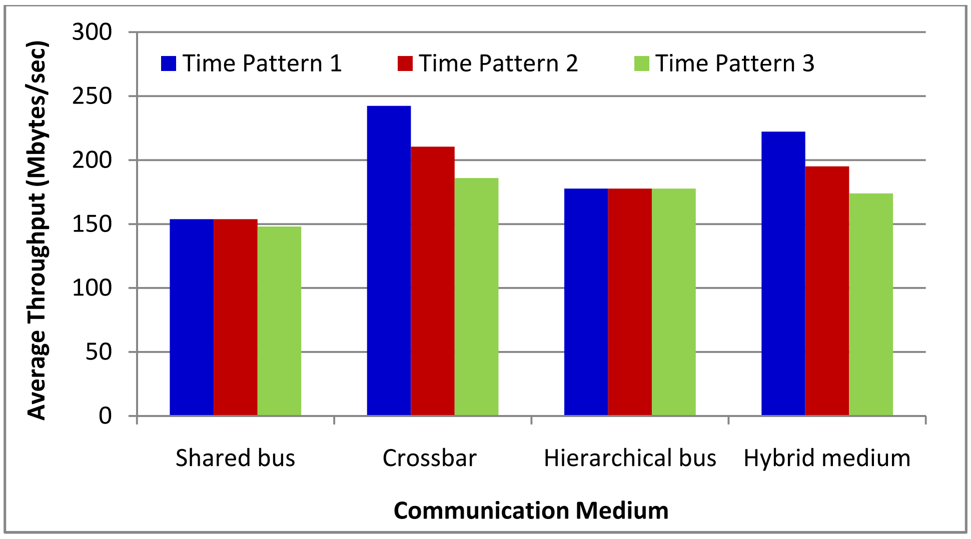


Figure 4-29 Scenario 4 - Simulation results

Scenario 5: Multiple masters with multiple slaves – fully simulated platform

In order to analyse the performance of the proposed platforms under a high data throughput scenario, all four masters were made to communicate with the slaves in a burst pattern. Each master-slave pair transmits two bursts of 20 word data at an interval of 400 ns between the bursts. The arrangement of cores on platforms is left the same as per previous scenarios. Table 4-6 show the master-slave pairs involved in communication.

Table 4-6 Scenario 5 - master-slave pairs

	Master (initiator)	**Slave (target)**
Link 1	Master 1	Slave 4
Link 2	Master 2	Slave 1
Link 3	Master 3	Slave 2
Link 4	Master 4	Slave 3

Figure 4-30 shows the simulation results of scenario 5. It can be seen that as per last simulations, the crossbar still outperformed its counterparts followed by the hybrid communication medium. The shared bus performed better than the hierarchical bus. This brings us to an important conclusion about the proposed hybrid communication medium. The hybrid medium performed better than the hierarchical bus in all but one case. Thus combining the advantages of two different communication media has given us a medium with performance better than the shared and hierarchical bus based medium.

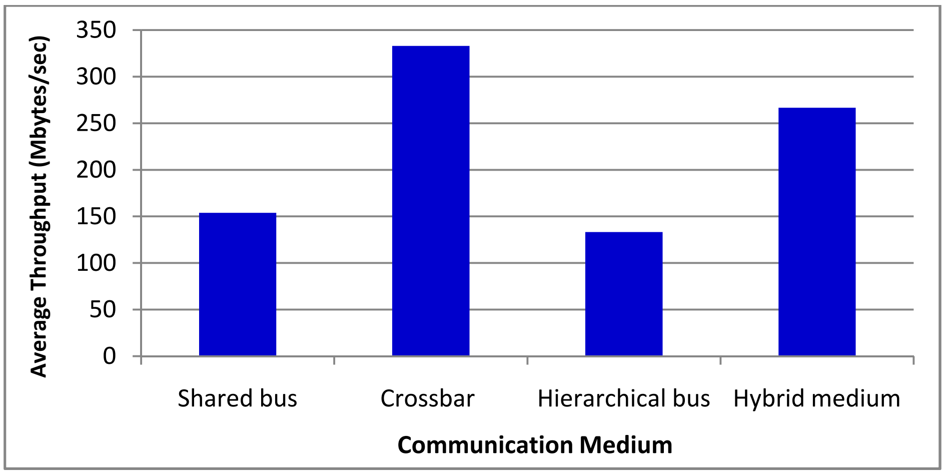


Figure 4-30 Scenario 5 - Simulation results

In the simulation results performed, the master cores were expected to initiate communication at a given time. During the simulations, it was found that these deadlines were seldom missed showing the degradation in QoS performance of the platform. In order to compare the delayed start-ups by different communication media, the missed deadlines were counted from scenario 2, 3, 4 and 5.

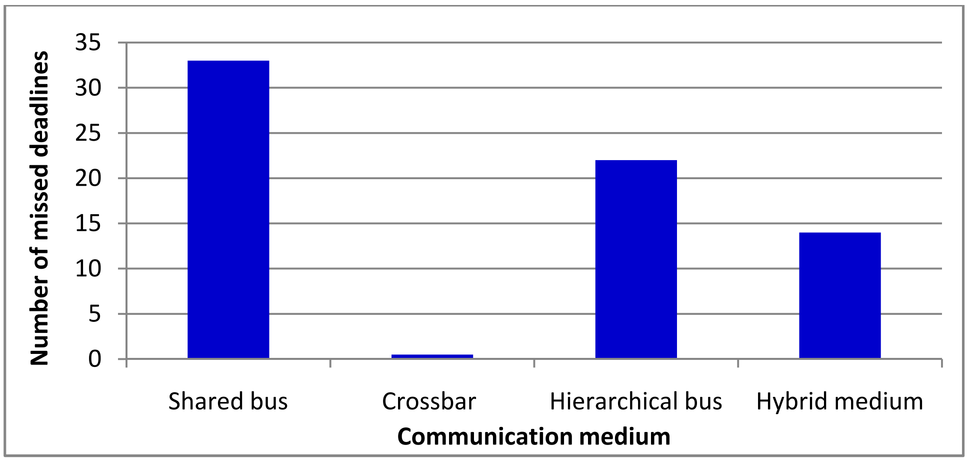


Figure 4-31 Delayed start-up deadlines of different communication media

Figure 4-31 displays the results of the comparison. It can be seen that the crossbar based platform missed the least deadlines followed by the hybrid medium. Shared bus performed the worst by missing the most start-up deadlines. This is due to the simultaneous availability of the links in the case of crossbar medium. Communicating cores didn’t have to wait for prior communication channels to be available to start communication, thus, not missing the start-up deadlines. The hierarchical bus performed better than the shared bus in average. However, under certain scenarios discussed above, when the communicating cores were placed on different part of bus medium and the bridge had to be used, the hierarchical bus performed similar to a shared communication medium.

4.5.5 Explanation of Results

The main aim of the simulations was to analyse the throughput characteristics of the developed platforms and to explore the factors affecting the throughput. From the simulations, it can be concluded that under no data contention on the interconnect architecture; all four types of communication media depict similar performance as dedicated point-to-point link. However, with the increase in traffic load, the platforms show different characteristics. Crossbar communication medium provides the best performance in terms of data throughput and on-time delivery of data. The performance of shared communication medium degrades with the increase in number of communicating cores. Due to the availability of only one link shared by all the connecting cores, one master-slave pair can effectively communicate with each other at any one time thus, causing other masters to wait till controller grants them bus access resulting in missing the start-up deadlines.

Analysing hierarchical and hybrid communication medium reveals the importance of placement of cores on the communication medium. By exploiting locality, and thus limiting the communication through the bridge, performance of the hierarchical and hybrid platforms is greatly improved. The hybrid communication medium gives a compromised performance between bus based (shared/hierarchical) and crossbar based platform.

Most of the communication centric platforms discussed in chapter 2 offer a hierarchical bus approach, as shown by simulation results of the proposed platforms; the hierarchical bus does not provide an optimal solution when employed for high data intensive multi-core architectures. As seen above by the simulation results, the crossbar based platform gives the best performance, this is in accordance with the conclusion of bus comparison conducted in [KYE-001]. This increased performance comes at a cost of increased area, power and design complexity (implementation results in section 4.6).

The shared bus implemented in the proposed platform follows FIFO arbitration. The performance of the shared medium can be improved by using priority based or even time sharing arbitration, by using pipelined transactions and even using asynchronous channels (chapter 2), but if the above mentioned performance tuning is carried out in the other types of implemented platforms, the performance is likely to improve proportionally. Thus, in order to prove the hypothesis of having the need of different communication centric platforms for future multi-core architectures, the platforms were developed without any performance enhancement technique.

4.6 Real World Example

To evaluate the proposed platforms and SystemC communication models, WiMAX (Worldwide Interoperability for Microwave Access) is chosen as a real world application for demonstration purposes. The WiMAX technology, based on the IEEE 802.16-2004 Air Interface Standard, is rapidly proving itself as a technology that will play a key role in fixed broadband wireless metropolitan area networks [WIM-001]. The scalable architecture, high data throughput and low cost deployment make Mobile WiMAX a leading solution for wireless broadband services.

The physical layer of WiMAX provides the wireless means of transmitting raw bits among devices. At present, the WiMAX standard has been implemented on DSP or ASIC technologies such as the WiMAX solution based on the Freescale MSC8 126 DSP [FRE-001] and Intel NetStructure WiMAX Baseband Card [INT-001]. However, as part of an embedded system, the WiMAX application demands a stricter low power requirements and more frequent updates since the standard keeps changing. In DSP and ASIC implementations, achieving high performance and power efficiency is not possible and a compromise has to be reached.

In the WiMAX physical layer, apart from the usual functions such as randomization, forward error correction (FEC), interleaving, and mapping to QPSK and QAM symbols, the standard also specifies optional multiple antenna techniques. This includes space time coding (STC), beam forming using adaptive antennas schemes, and multiple input multiple output (MIMO) techniques which achieve higher data rates. The OFDM modulation/ demodulation is usually implemented by performing FFT and inverse FFT on the data signal.

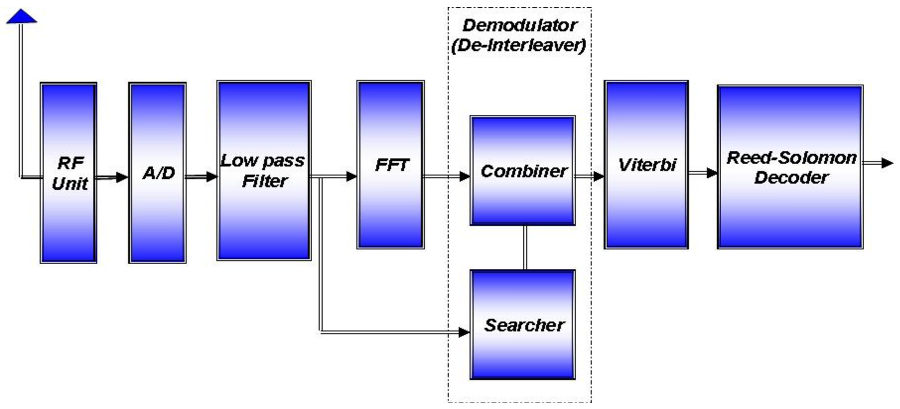


Figure 4-32 Block diagram of WiMAX reciever

Consequently, FFT and Viterbi are considered in this work to be modelled in SystemC as hardware accelerators which are compute intensive modules in WiMAX computing chain. The WiMAX application runs as a software core on the ARM7 [ARM-003] processor. Figure 4-32 shows the block diagram of a WiMAX receiver. The reconfigurable FFT encoder and a Viterbi decoder modelled in systemC are integrated into the WiMAX chain thus co-simulating modules from different levels of abstraction (Viterbi and FFT in SystemC and WiMAX running as software of ARM7). Figure 4.33 shows the communication links in the WiMAX receiver.

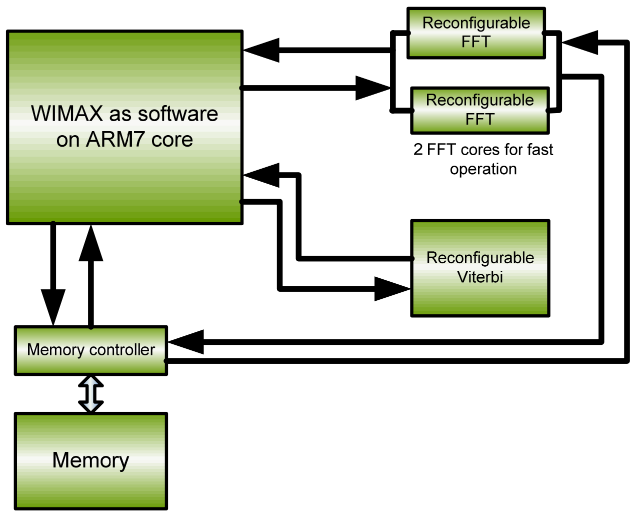


Figure 4-33 Communication links in the WiMAX receiver

To investigate power and area characteristics of the main blocks of WiMAX receiver, RTL-version of these blocks are used. A 0.13 micron technology library is used for ASIC synthesis. Gate level simulation is carried out and Synopsys design compiler is used to calculate power and area results. Functionality evaluation was performed through the complete model. This is demonstrated as received bit rate performance of the overall receiver.

Figure 4-34 demonstrates impact of communication infrastructures on the WiMAX application for a range of FFT sizes: 128-2048 points. The implemented reconfigurable Viterbi module decodes for different constraint lengths (K=3, K=5, and K=7). Since, in Mobile WiMAX [IEE-001], the FFT size of OFDM part is scalable from 128 to 2,048. A Configuration Mode is defined;

$$Configuration\ Mode = \ Log(FFT - Size) + \left(\frac{1}{Viterbi{-}code{-}rate}\right) - 8.$$

An interesting result is that the used code rate has nearly no influence on the hardware complexity of the Viterbi decoder structure, just one more code word has to be added in the same clock cycle. Therefore all metric values have to be increased by one bit. However, when it is increased considerably, its effect on area, power and timing issues becomes of notable importance.

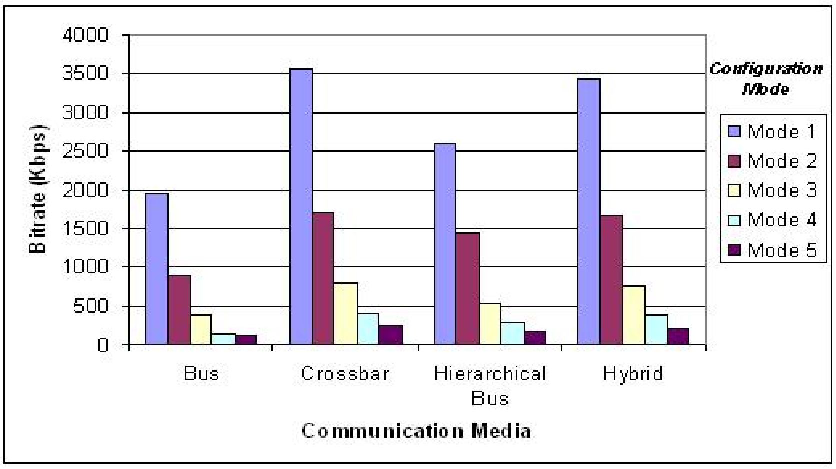


Figure 4-34 Impact of Communication Centric System-on-Chip Design on WiMAX

The received bit-rate performance of the overall WiMAX receiver is measured with various communication platforms, as shown in Figure 4-34. It can be seen that the crossbar based implementation gives the best bit-rate, followed by hybrid communication medium. This result is in accordance with the results obtained by simulations in 4.5.4, by providing simultaneous links for cores to communicate a higher throughput is achieved. Figure 4-35 illustrates the change in power and area by different communication media. It can be seen that we get the least power when global bus is employed as a communication medium. This is because in bus based system only one core is effectively communicating with any other core in the system at any given time. Also the bus arbiter is simple in operation and also occupies less area on silicon. This however, comes at a cost of least data throughput. Hierarchical bus performance and power/area results are between hybrid and bus based implementations.

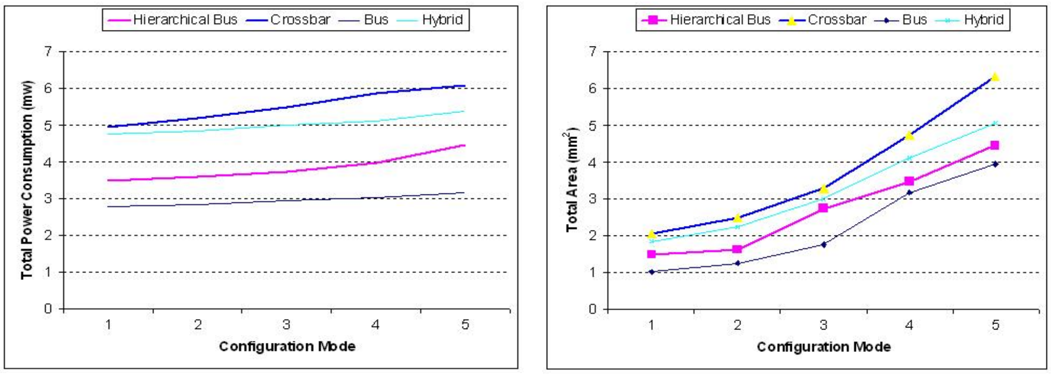


Figure 4-35 Impact of Communication Centric System-on-Chip Design on WiMAX Power and Area Consumption

A compromise has to be reached between power, area and performance figures when choosing the best performer. The crossbar did provide high throughput but it is not area and power efficient. The bus based implementation provided the best power and area performance but is not efficient when it comes to the throughput. This brings us to a conclusion that there should be a communication media that provides a compromised data throughput, power and area figures between the global shared bus based and the crossbar based implementations. This is the concept behind a hybrid medium implementation. From the above results, it can be seen that the hybrid medium provides reduced power and area figures when compared to the crossbar based implementation. Hybrid medium also provide better throughput characteristics than the hierarchical bus and shared bus based platforms. The increase in power of hybrid medium is due to the increase in switching needed for communication between the cores not connected locally i.e. exist on different channels. The increased area in the hybrid medium is because of the bridge implementation and complex controller functionality. In hybrid medium implementation, placement of cores play an important part in the throughput and power characteristics. High communicating cores can be placed together on crossbar part to exploit the advantages of crossbar matrix, and by placing low communicating cores on a shared bus; we get the advantage of quicker bus access. In short we can conclude that hybrid medium provides a good trade-off between power, area and performance.

4.7 Summary

This chapter took the notion of platform based design and a novel communication centric platform based design methodology was proposed. Four different communication centric platforms are developed each with different throughput, area and power characteristics. Thus, for any given application, the platform most suited to the design constraints is used for that application. The proposed platforms are designed with a socket based approach so cores can be integrated in these pre-verified platforms in plug and play fashion without the need of communication controller programming.

The concept of hybrid communication medium is also introduced in this chapter, two different types of communication media co-exist in a system, combining advantages of both communication media. A high level modelling strategy is proposed and the proposed platforms are modelled for quick verification and simulations. Simulation results and implementation of WiMAX receiver using the developed platforms demonstrates the effectiveness of the proposed platform approach.

Chapter 5
SOCCAD Tool

5.1 Introduction

The present day consumer electronic market is heavily driven by time to market constraints. As mentioned earlier in chapter 2, platform based design approach is seen as a way to rapidly develop SoC architectures by exploiting the concept of component re-use. Looking at the fact that SoC design methodologies are moving to a communication centric design flow, in chapter 4, four communication centric platforms were developed. Each of the developed platforms has different area, power and throughput characteristics associated with them, making them suitable for different types of applications.

Bringing together the communication centric platforms, systemC communication models and systemC reconfigurable cores model [AHM-006][AHM-007][AHM-008][AHM-009][AHM-010][AHM-011], a tool has been developed that generates the complete SoC architecture with optimised communication media for reduced power and area characteristics. This chapter deals with the development of the tool for automated generation of SoC architectures targeting custom reconfigurable cores.

This chapter begins with a brief introduction of some existing SoC generation tools developed commercially and by reputable research institutes. The proposed SOCCAD (SoC communication architecture development) tool is then discussed followed by the design flow and finally the demonstration of the developed tool.

5.2 Current SoC architecture development tools

New tools are emerging in the market to provide designers the capability to rapidly create SoC architectures. These tools allow the designers to specify the whole system design using drag and drop and graphical user interface. Most of the commercially prepared tools target company's own products and thus integration of third party components still remains a time consuming job. Targeting at different level of programming, some tools work at systemC level and some at low level HDL levels, while some translate high level programming languages into HDL files.

System Vision™ by Mentor graphics [MEN-001], coreAssembler™ and Galaxy™ Design Platform by Synoposis [SYN-001][SYN-002], Platform Architect by CoWare [COW-001], Xtensa Xplorer by Tensilica [TEN-002], Nx-Bilder by Philips [PHI-002] and SonicsStudio by Sonic Inc. [SON-002] are just some of the industry leading tools in this area that provide the designers the capability to rapidly create SoC architectures by allowing faster design exploration. Aimed at system architects, platform and processing core developers, most of these tools are based on high level language environments that allows embedded software developers to validate their software on a model of the silicon.

Catering for the need of reconfigurable architectures and the importance of communication centric designs, research institutes have come up with several SoC communication network generation and reconfigurable architecture development. Carrabina et al. have given a good review of bus centric architecture generation tools in their paper bus-centric architecture generation tools [CAR-001].

Jalabert et al. proposed an advanced NoC architecture called Xpipes and proposed a tool called XpipesCompiler, targeting high performance and reliable communication for on-chip multi-processors. It consists of a library of soft macros (switches, network interfaces and links) that are composable and tuneable at run time so that domain specific heterogeneous architectures can be instantiated and synthesised. The proposed XpipesCompiler tool automatically instantiates a customised NoC from the library of network components [JAL-001].

CHAIN (Chip Area Interconnect) is a delay insensitive SoC interconnect created by AMULET group at Manchester University [APT-001]. It uses GALS methodology to connect devices together. A CHAIN network is claimed to reduce power due to power being dictated by traffic load and not by clock rate. CHAINworks™ is tool suite by Silistix (a spin-out of the University of Manchester) that is used for design and synthesis of CHAIN fabrics [SIL-001].

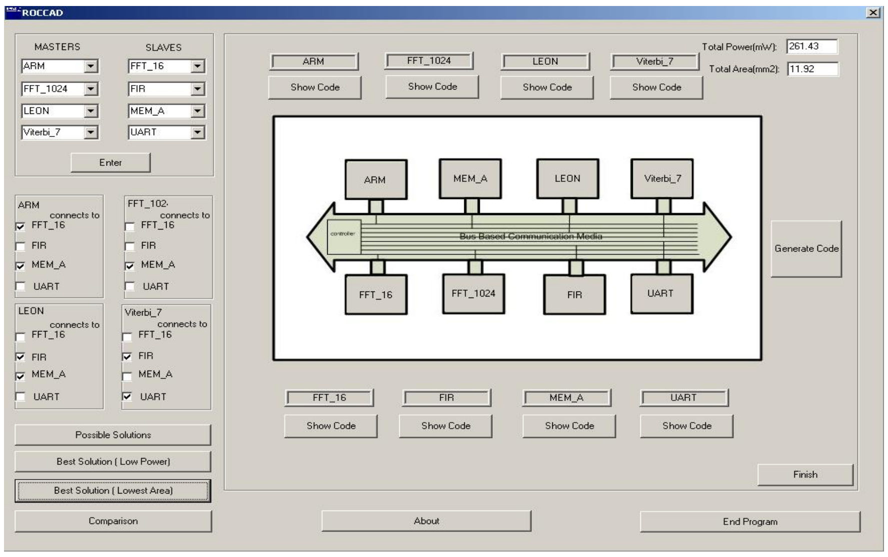
Dealing with the automated generation of application specific architecture for a heterogeneous multi-processor system on chip, Lyonnard et al. presented a design flow where architectural parameters are first extracted from a high-level system specification. These parameters are first used to create communication network instances. For each module, a node processor instance is then created with an interface to integrate with the communication network, thus, developing a cycle accurate MPSoC architecture. [LYO-001]

Platune was proposed by T. Givargis and F. Vahid as a framework for performance and power tuning of SoC platform. Platune is used to simulate an embedded application that is mapped onto the SoC platform and output performance and power metrics for any configuration of the SoC platform. Thus, it can be used as a tool to aid the system designer in selecting appropriate architectural parameter values, for a given application that is to be mapped on the parameterized SoC platform, in order to meet performance and power goals. [GIV-001].

5.3 SOCCAD Tool

The third design challenge addressed in this thesis is the automated generation of complete multi-core architectures. A tool called SOCCAD has been developed. The main aim of the SOCCAD tool is to automate the development of multi-core architectures, incorporating custom reconfigurable components, conventional RISC based processors, hardware accelerators and memory blocks etc. Depending on the application required, components can be chosen from a built in library. The tool automatically generates the communication media and gives area and power figures for implementations with different communication media. The best solution based on power and area can also be generated by the tool.

SystemC models of communication centric platforms, described in chapter 4, along with systemC models of custom reconfigurable and fixed cores implemented as part of the SOCCAD project are used as the base unit for the tool. The tool comprises a library of the four communication centric platforms (explained in chapter 4) and processing cores, memories etc. An interface of the prototype SOCCAD tool is shown in Figure 5-1.



MASTERS: ARM, FFT_1024, LEON, Viterbi_7

SLAVES: FFT_16, FIR, MEM_A, UART

ARM connects to: [✓ FFT_16] FIR [✓ MEM_A] UART

FFT_102· connects to: FFT_16 FIR [✓ MEM_A] UART

LEON connects to: FFT_16 [✓ FIR] [✓ MEM_A] UART

Viterbi_7 connects to: FFT_16 [✓ FIR] MEM_A [✓ UART]

Total Power(mW): 261.43

Total Area(mm2): 11.92

Figure 5-1 Interface of the SOCCAD tool

The components needed to implement the system are chosen from the drop down menu captioned “MASTER” and “SLAVE”. If the components needed for the required multi-core architecture is not available in library, transaction level modelling needs to be done for that component. After the selection of components, their connections need to be established. This is done by selecting the components that are supposed to communicate with the integrated Master components (shown in Figure 5-1).

The tool gives options of “possible solutions” where it shows the implemented system with different communication platforms and also displays the area and power characteristics. The “Best Solution” option displays the system with lowest power and area characteristic and “Comparison” option displays the information of all the generated Systems with different communication architectures.

The “Show Code” option displays the systemC code of the required component and “Generate Code” option displays the systemC code for the complete SoC architecture ready to be simulated. The “comparison” option also generates an excel file of area/power characteristics of system that can be used for analysis purposes.

5.4 Tool Design Flow

Figure 5-2 shows the design flow of the developed SOCCAD design methodology. It can be divided into three main parts:

- Modelling of processing cores and communication platforms
- Automated communication centric placement of cores on communication centric platforms
- Generation of complete systemC code for system verification

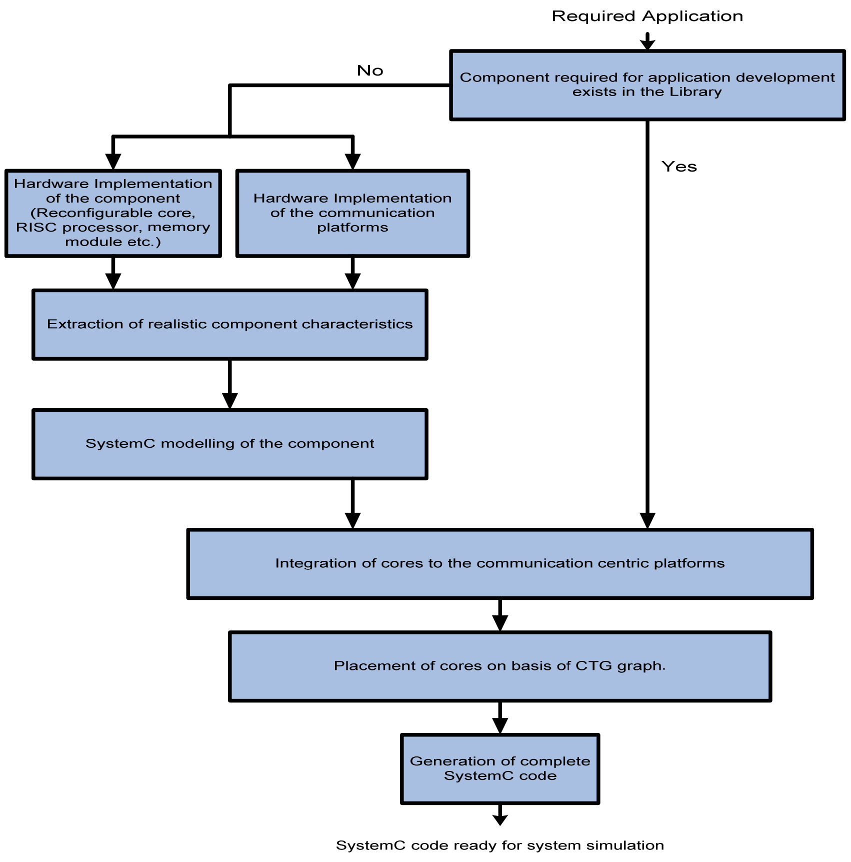

Required Application
Component required for application development exists in the Library
No
Yes
Hardware Implementation of the component (Reconfigurable core, RISC processor, memory module etc.)
Hardware Implementation of the communication platforms
Extraction of realistic component characteristics
SystemC modelling of the component
Integration of cores to the communication centric platforms
Placement of cores on basis of CTG graph.
Generation of complete SystemC code
SystemC code ready for system simulation

Figure 5-2 Design Flow of SOCCAD tool

As mentioned earlier, the tool incorporates a library of SystemC components including the communication centric platform described in chapter 4, reconfigurable cores [AHM-006][AHM-007][AHM-008][AHM-009][AHM-010][AHM-011], fixed cores, RISC based processors and memory blocks. The components in the library are also implemented in hardware to extract realistic power and area characteristics.

With the increasing number of cores being integrated on a single chip, communication media is fast becoming a major design bottleneck. As concluded in chapter 4, the placement of cores on communication platforms heavily affects system performance. In a traditional communication architecture design flow, components are manually placed in the SoC designs and their effects are analysed untill an ideal placement scenario is reached. In order to solve this problem, an automated communication centric placement of components is proposed.

In the proposed communication centric placement, a communication task graph (CTG) is used. A *CTG* G' = G'(T,D) is a directed acyclic graph, where each vertex represents a computational module. Each directed arc between modules characterises either data or control dependencies.

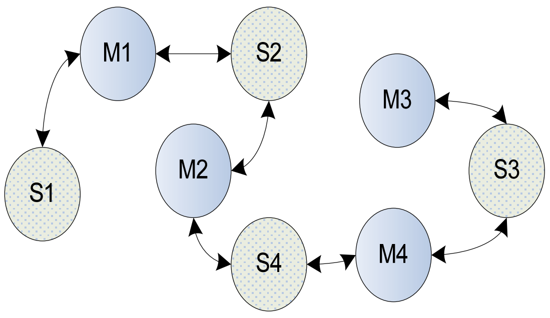


Figure 5-3 Example of a CTG

CTG simplifies the process of mapping components to communication media by dividing the components on basis of their dependencies. CTG can then be simplified to realise shared bus based, hierarchical bus based or hybrid communication medium. Figure 5-3 shows an example of a CTG with M1, M2, M3, M4 as the masters or communication initiators and S1, S2, S3 and S4 as the slave components.

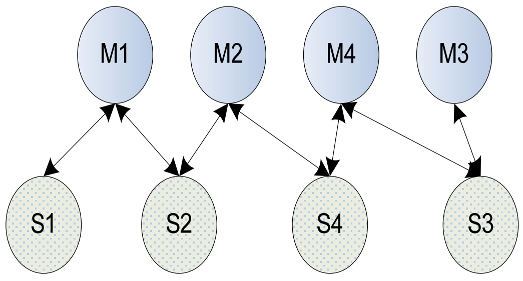


Figure 5-4 Simplification of CTG for shared bus

In the shared bus based communication medium, one global bus is shared by all the attached components. The CTG in Figure 5-3 translates to a shared bus mapping shown in Figure 5-4. It can be seen that the components shared by more than one component are placed closer to them. Simplified bus is a rather trivial example of using CTG. However, considering the same example implemented for a hierarchical bus, we have now two shared buses connected together via a bridge. Correct placement of cores in this scenario is more significant for system performance e.g. S1 only communicate with M1 so making it communicate through the bridge will cause it to wait for the availability of bridge and thus will affect the performance of component M1. Figure 5-5 shows the hierarchical bus implementation of the CTG in Figure 5.3.

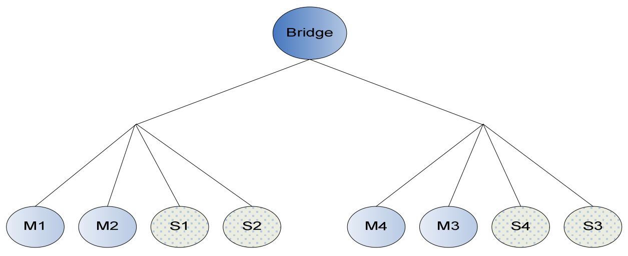


Figure 5-5 Simplification of CTG for Hierarchical bus

When considering the hybrid communication architecture, as the designer has two different communication media with different number of components attached to each part, and communicating to each other through the bridge, the use of CTG becomes even more desirable. Considering the above example, since each of M2, M4 needs to communicate with S2, S4, and S3. They are all placed on the crossbar part of the hybrid media, and M1, M3 and S1 are placed on the bus based part of the hybrid medium. This also reduces the communication through the bridge. For example, just having placed S3 instead of S1 in Figure 5-6 below, M1 and M4 have to wait for bridge availability for any communication they waiting on from S1 and S3 respectively causing the overall system performance to decrease.

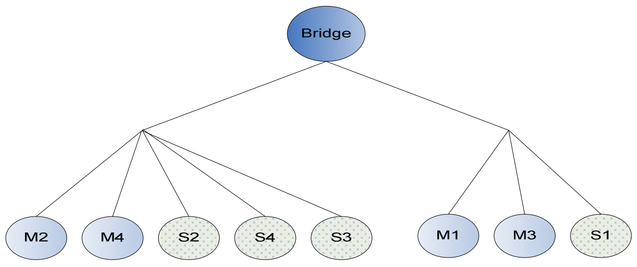


Figure 5-6 Simplification of CTG for Hybrid medium

In order to increase the complexity of the above discussed CTG and to see its effect on communication media mapping, consider an additional link between S3 and M2. This additional link changes the shape of CTG, and M2 is now connected to three slave components S2, S3 and S4. This affects the placement of M2 and M4 on the communication media. Figure 5-7 shows the CTG graph where M1 is connected to S1 and S2, M2 connected to S2, S3 and S4. M4 connected to S3 and S4 and finally M3 connected to S3.

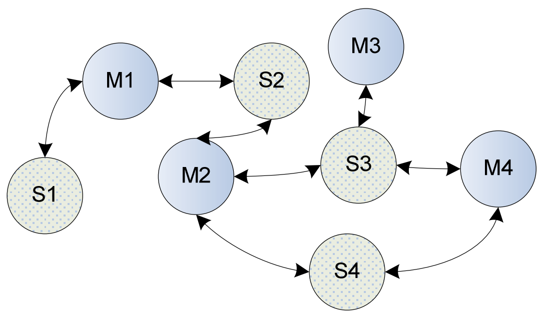


Figure 5-7 Example 2 CTG

Figure 5-8 shows the placement of the CTG of Figure 5-7 on a shared bus. The shared bus placement looks similar to the previously discussed scenario due to the communication medium being a single shared bus. However, for this scenario, shared communication medium will be highly inefficient for system performance due to the components having to wait for the bus access to communicate with the required components.

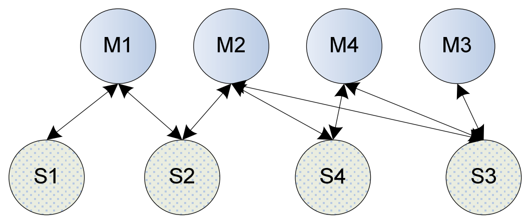


Figure 5-8 Simplification of Example 2 CTG for Shared bus

The difference in component placement becomes visible when dealing with the hierarchical bus and hybrid bus. Since M4 communicates with S4 and S3, and M2 communicates with S4, S3 and S2, when dividing the bus to accommodate equal number of cores, M2, M4, S4 and S3 are placed on one bus and M1, M3, S2 and S1 placed on the other bus. The bridge is accessed only when M2 wants to communicate with S2 and M3 wants to communicate with S3. On the other hand, M1 can communicate freely with S1 and S2. Figure 5-9 shows the placement on hierarchical bus.

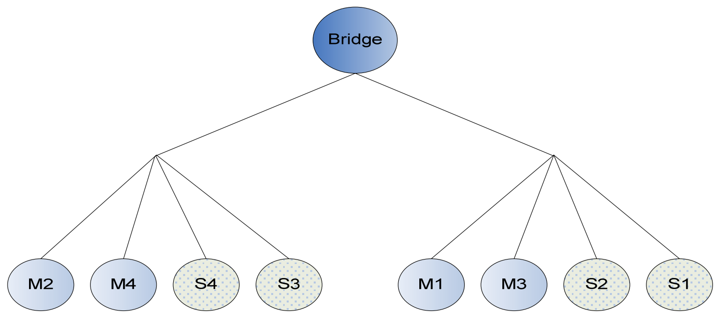


Figure 5-9 Simplification of Example 2 CTG for Hierarchical bus

The same system, when implemented on hybrid system displays an ideal placement scenario where all the main communication components are placed on the crossbar section. This placement drastically reduces the communication through the bridge making the overall system performance more efficient. The bridge is only used when M3 wants to communicate with S3. Figure 5-10 below shows the placement figure of hybrid communication medium.

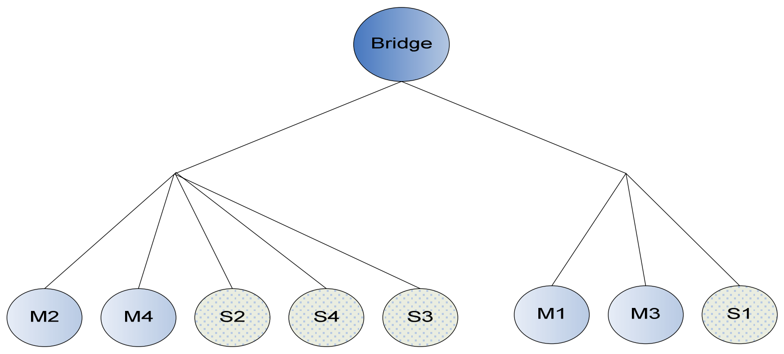
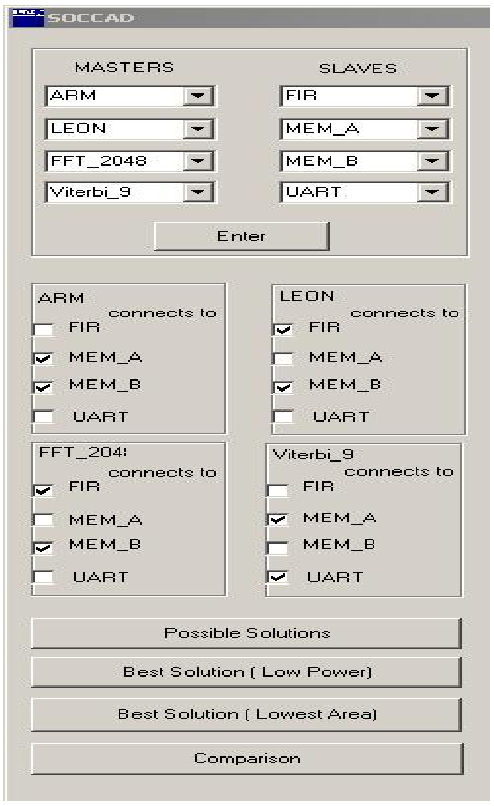


Figure 5-10 Simplification of Example 2 CTG for Hybrid media

The generation of systemC code for the whole system becomes rather a trivial task after the placement has been done and it has been established which component is connected to which interface block on the platform. This is due to the interface being based on the OCP socket standard. A high level systemC file is created by the tool with all the components integrated. Thus, simply by compiling the code the designer can simulate the system and by using efficient test benches verify the implemented functionality.

5.5 Tool Demonstration with Real life example

In order to demonstrate the working of the SOCCAD tool, let us consider an example of a system incorporating ARM, Leon, Viterbi and FFT as master cores. Reconfigurable Viterbi selected is of constraint length 9 and FFT size is of 2048-points. The slave cores selected from tool drop down menu are FIR filter, two memory modules MEM_A and MEM_B and a UART. First step after the selection of cores to be integrated is to establish the connections between the master and slave components. The checkboxes on the SOCCAD tool interface are used to input the communication dependencies. The communication input interface and the respective CTG is shown in Figure 5-11.


SOCCAD

MASTERS	SLAVES
ARM	FIR
LEON	MEM_A
FFT_2048	MEM_B
Viterbi_9	UART

ARM connects to
- [ ] FIR
- [✓] MEM_A
- [✓] MEM_B
- [ ] UART

LEON connects to
- [✓] FIR
- [ ] MEM_A
- [✓] MEM_B
- [ ] UART

FFT_204! connects to
- [✓] FIR
- [ ] MEM_A
- [✓] MEM_B
- [ ] UART

Viterbi_9 connects to
- [ ] FIR
- [✓] MEM_A
- [ ] MEM_B
- [✓] UART

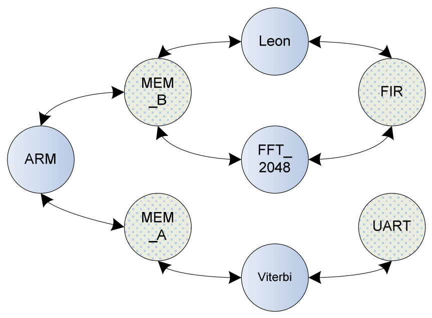


Figure 5-11 SOCCAD communication input interface and respective CTG

As seen in the CTG in Figure 5-11, all components have two cores connected to them apart from the UART, which is only connected to Viterbi. Figure 5-12 shows the shared bus and crossbar based communication media generated by the tool.

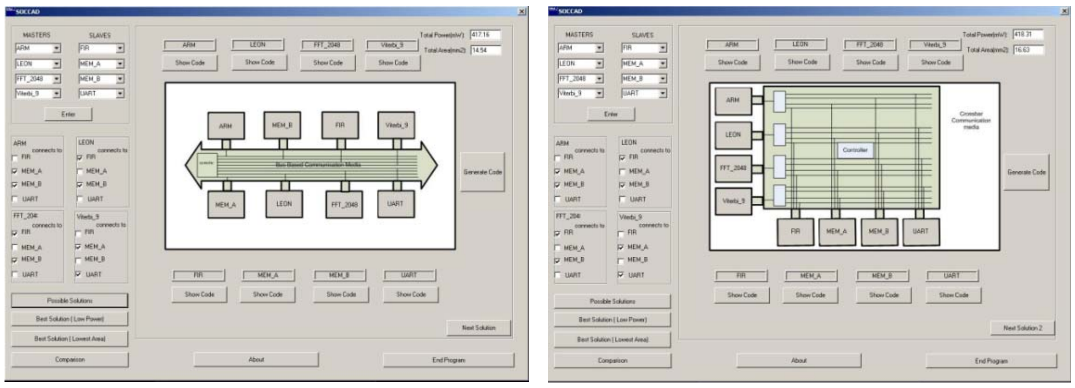


Figure 5-12 Shared bus and Crossbar based media implementation by SOCCAD tool

As mentioned in section 4.5.4, when implementing a hierarchical bus based system, the placement plays an important role. The placement graph obtained from the CTG and the system generated by the SOCCD tool for hierarchical implementation is shown in Figure 5-13. The aim of placement is to minimize communication through the bridge. Hence, Viterbi, ARM, UART and MEM_A are placed on one bus and Leon, FFT, MEM_B and FIR on the other bus. The only time when communication through the bridge takes place is when the ARM processor has to communicate to MEM_B.

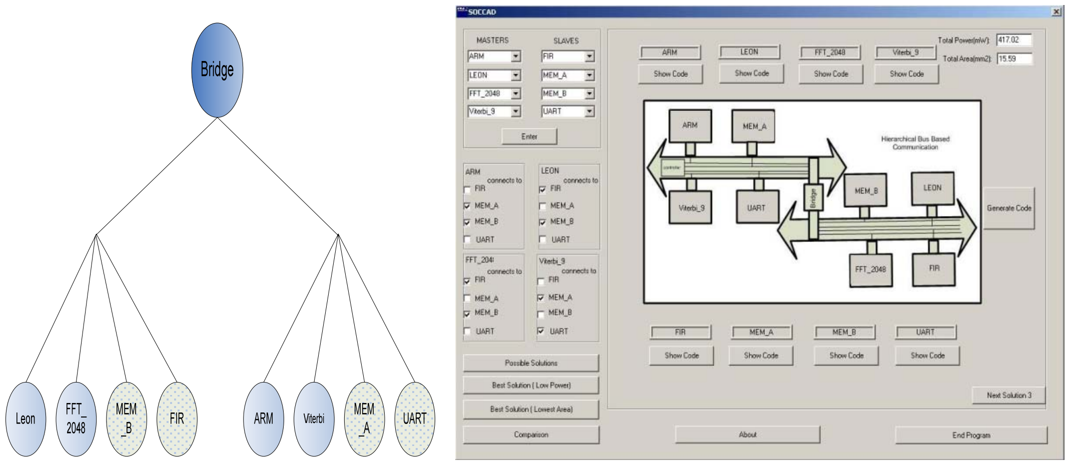
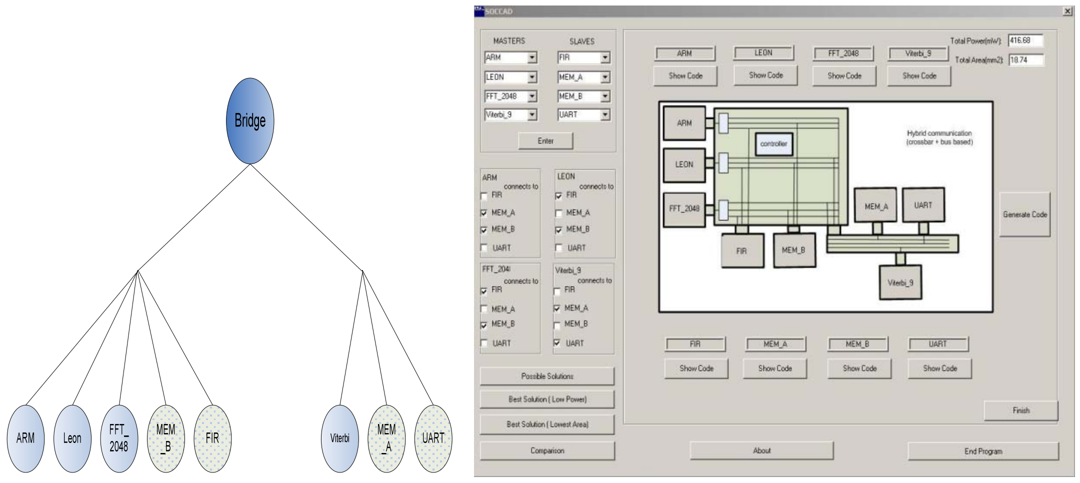


Figure 5-13 Simplified CTG for hierarchical bus and SOCCAD implementation of system

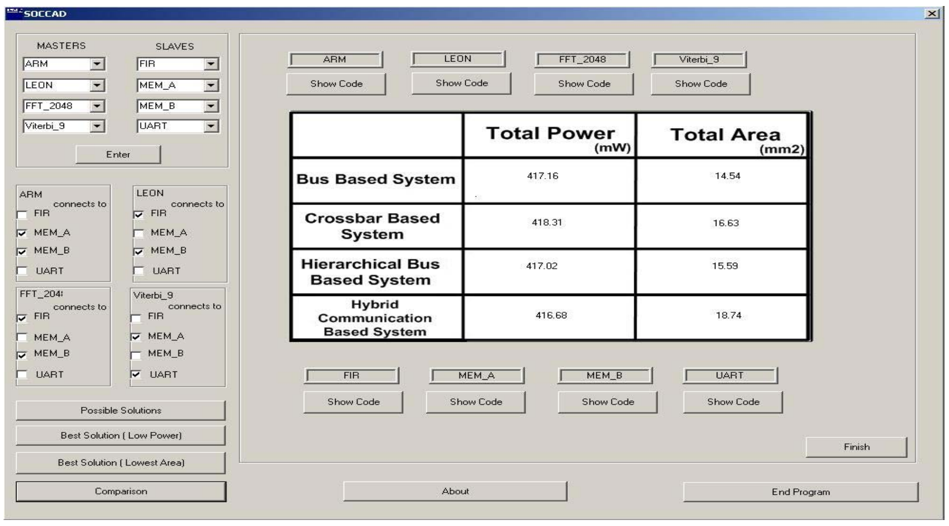
The system when implemented for hybrid medium, shows the ARM processor moved to the crossbar part of the platform. However, the communication through the bridge is still done by the ARM when it is communicating with MEM_A. Figure 5-14 shows the CTG simplified for the hybrid communication medium and the corresponding hybrid medium based system generated by the SOCCAD tool.



Total Power(mW): 416.68
Total Area(mm2): 18.74

Figure 5-14 Simplified CTG for hybrid media and SOCCAD implementation of system

An important feature of SOCCAD tool is to display total power and area of the generated system for different communication media platforms. Figure 5-15 shows the SOCCAD tool, displaying the Power and Area figures for the generated system. It can be seen that the shared bus based system provides the lowest area implementation while hybrid media based implementation provides the lowest system power. Crossbar implementation yields the highest system power making it unsuitable for a low power implementation of the system under consideration.



MASTERS	SLAVES
ARM	FIR
LEON	MEM_A
FFT_2048	MEM_B
Viterbi_9	UART

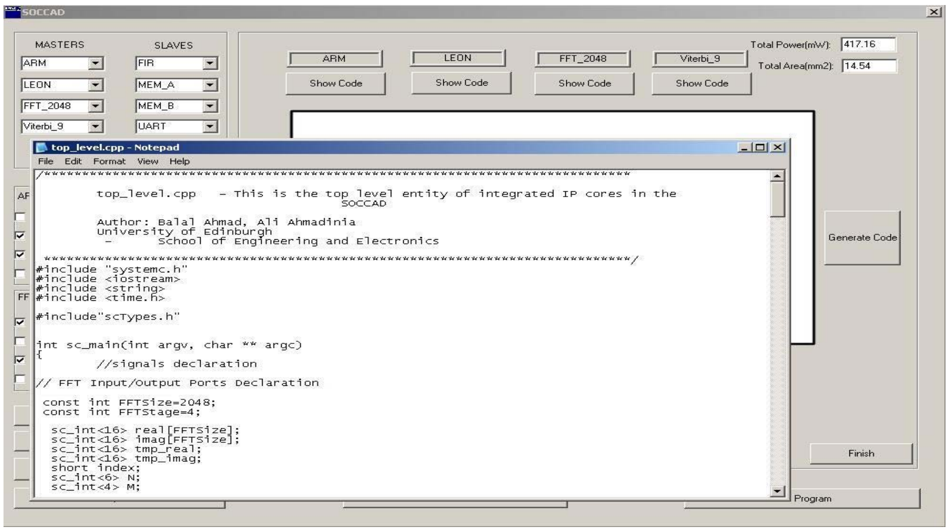
ARM connects to: ☐ FIR, ☑ MEM_A, ☑ MEM_B, ☐ UART
LEON connects to: ☑ FIR, ☐ MEM_A, ☑ MEM_B, ☐ UART
FFT_204 connects to: ☑ FIR, ☐ MEM_A, ☑ MEM_B, ☐ UART
Viterbi_9 connects to: ☐ FIR, ☑ MEM_A, ☐ MEM_B, ☑ UART

	Total Power (mW)	Total Area (mm2)
Bus Based System	417.16	14.54
Crossbar Based System	418.31	16.63
Hierarchical Bus Based System	417.02	15.59
Hybrid Communication Based System	416.68	18.74

Figure 5-15 Power & Area results of system generated by SOCCAD tool

In chapter 4, it was concluded that the crossbar provides the best throughput followed by the hybrid medium. Thus in order to achieve the compromise solution, a hybrid medium based system with second best performance figure and lowest power figure comes out as an attractive solution for low power and high throughput system. However, for a system with area constraints, hybrid communication medium will tender unsuitable, and choice has to be made between a crossbar and hierarchical bus based platform.

For system verification, overall systemC code is required. SOCCAD tool provides an option for the generation of systemC code thus facilitating the rapid simulation and verification of system. Figure 5-16 shows the top level systemC file generated by SOCCAD tool.



Total Power(mW): 417.16
Total Area(mm2): 14.54

```
/*********************************************************************************

        top_level.cpp   - This is the top level entity of integrated IP cores in the
                                        SOCCAD

        Author: Balal Ahmad, Ali Ahmadinia
        University of Edinburgh
          -      School of Engineering and Electronics

 *********************************************************************************/
#include "systemc.h"
#include <iostream>
#include <string>
#include <time.h>

#include"scTypes.h"


int sc_main(int argv, char ** argc)
{
        //signals declaration

// FFT Input/Output Ports Declaration

 const int FFTSize=2048;
 const int FFTStage=4;

  sc_int<16> real[FFTSize];
  sc_int<16> imag[FFTSize];
  sc_int<16> tmp_real;
  sc_int<16> tmp_imag;
  short index;
  sc_int<6> N;
  sc_int<4> M;
```

Figure 5-16 Overall SystemC code generated by SOCCAD tool

5.6 Summary

In this chapter the SOCCAD tool was introduced. SOCCAD tool is developed to automate the generation of complete systemC code of the system for quick simulation and verification. The communication centric platforms developed in chapter 4 along with systemC models of reconfigurable cores, RISC processors, memory modules etc. are added in the tool’s library. Based on the application, components can be integrated from the library. SOCCAD tool generates the system for different communication centric platforms and gives power and area calculations for comparison purposes. SystemC code for the complete system can be automatically generated by SOCCAD tool either on low power constraint or low area constraint.

Chapter 6

Dynamically Reconfigurable NoC

6.1 Introduction

SoC opens up the feasibility of a wide range of applications making use of massive parallel processing and tightly interdependent processes, some adhering to real-time requirements, bringing into focus new complex aspects of the underlying communication structure [BJE-001]. Traditional bus based communication medium will prove to be a bottleneck in achieving the high processing requirements in future SoC, especially with the on-chip communication requirements rising over 9 GB/s. NoC is considered as a solution to this communication bottleneck.

In chapter 2, the concept of NoC and a review of current NoC architectures was introduced. The NoC architecture is generated by choosing a network topology, one of the routing and switching schemes and fixing a packet size. Many different NoC architectures have been proposed in the past few years, as described in chapter 2. Network parameters like topology, routing algorithm, switching scheme and packet size are set at the generation time, mostly to deal with the worst case communication scenario. However, in the real world, the cores connected are not utilising all the available bandwidth, and some cores utilise more bandwidth then others. This results in some of the allocated resources being not fully utilised, causing waste of resources. In an invited paper by [ATI-001], it is suggested that 74% improvement in power can be achieved by using an application specific NoC against the traditional NoC approach.

In this chapter a dynamically reconfigurable NoC (drNoC) architecture is proposed as a high data throughput communication centric platform for future multi-core systems [AHM-002][AHM-003][AHM-004]. Exploiting the notion of reconfigurability, the proposed drNoC allocates the resources and parameters suitable for the desired application domain at run time, thus, not having to deal with effects of unfavourable selection of parameters. In order to develop a generic multi-domain communication centric platform, the proposed NoC has been given a socket based approach as per the platforms in chapter 4 to allow for ease of integration.

This chapter begins with the description of the proposed drNoC. The description of different layers and the router design is then explained. The layered structure makes it easy for designers to understand the different aspects of the proposed architectures. Simulation results are finally discussed to establish the effectiveness of the proposed network.

6.2 Proposed dynamically reconfigurable NoC

Keeping in view the success of busses and increased communication requirements in multi-core SoC, a dynamically reconfigurable NoC (drNoC) has been proposed that combines the advantages of bus based systems and the advantages of NoC. The network changes its characteristics with the changing communication requirements of the system. The design inspiration comes from the fact that different cores connected in a system have different bandwidth requirements. e.g., in an advanced PDA with 3G communication capabilities, the on-chip communication can vary depending on the user application, thus a network with fixed communication design parameters is not the optimal solution in terms of power and data throughput.

The allocation of network resources depends on network topology, switching and routing decisions. Packet size also affects the network performance both in data throughput and overall system energy consumption [BEN-001]. In the proposed design, the network configures itself in terms of its routing, switching and packet size to maintain the QoS requirement of the system.

This intelligent network has its kernel in the form of a micro network stack called smart network stack (SNS) of the node processor. Depending on the data about to be transferred, the SNS makes the decisions about the packet size, switching and routing, required for the data and includes this information in packet header. This information is read by the router and packets are processed as desired. Thus, in the case of a processing core with high bandwidth requirements, packet size would be increased, also switching would change from packet switching to circuit switching. These changes increase data throughput and decrease the switching power and timing delays. Distributed control of a network has the advantage that if a single node failure occurs, the network continues to perform its functions.

The router is the main building block of the network. It serves two main functions. Firstly, it acts as the interface between the core and network. To allow for heterogeneous component integration, the router is built with a bus like interface based on OCP socket standard which allows cores to integrate with less effort. Secondly, the router routes the data packets and control signals to the right path.

SNS is a modified version of the micro network stack proposed by L. Benini [BEN-002]. The SNS is composed of five layers, application, transport, network, data link, and physical layer (shown in Table 6-1).

Table 6-1 Smart Network Stack

Software	Application Layer
Architecture & Control	Transport Layer Network Layer Data Link Layer
Physical	Wiring

The application layer provides the user interface to the communication system. Thus the applications running on the processor core does not have to worry about the complex communication network facilitating its communication with other applications connected to other nodes.

6.3 Network architecture and Control

The network architecture refers to the topology and physical organization of the interconnect network. The protocols specify the use of these network resources during system operation. Network control is taken care of by transport, network and data link layer.

6.3.1 Transport Layer

Unlike the bus based system, the data transported over NoC is in the form of small packets. This packetisation of data is dealt with by the transport layer. All packets have same format, “packet header” followed by the “payload”. Packet header carries information necessary for the routing of packets to its destination.

Some important subfields of the packet header include:

Type: Indicates the start of packet and specifies if the packet is a broadcast packet (meant for all connected cores) or is meant for only a specific destination.

Destination Address: Indicates the destination address. In case of broadcast packets, the destination bits are not read by the router for routing the packets.

Switching Type: Instructs the router to switch between packet switching and circuit switching (set by the Network Layer).

Packet type: Specifies if it’s a data packet, acknowledgment of received packet, or request for retransmission in case of not received or receipt with error packet.

Source Address: Source address is read by the destination node and is used in replying to the packets.

Dataoffset: This indicates where the data begins in the packet. It can also be referred to as the sequence number of packet useful in assembly of data from the received packets at the destination.

Checksum: Used for error detection.

Header bits are followed by the payload. The size of the payload depends on the type of data transmitted. As mentioned earlier, the size of packet changes for different types of communication. Three types of data thresholds are defined, normal, medium and high. Normal threshold refers to any communication where data throughput is under 100 Kbits/sec e.g. interrupt handling etc., high data threshold refers to any communication above 10Mbits/sec e.g. CPU cache to main memory, compressed or uncompressed video. Medium data threshold is referred to as any communication between these two thresholds.

In the prototype, these values are fixed. However, data threshold values can be added by users when defining the network parameters. Figure 6-1 shows a conceptual data packet structure.

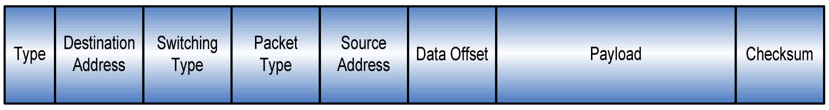


Figure 6-1drNoC data packet structure

6.3.2 Network Layer

The network layer deals with switching and routing aspects of the packetised data. Switching can be packet switching and circuit switching. If only a small amount of data is to be transmitted, then setting up a circuit to transfer data is inefficient. Thus, the network layer transmits data by packet switching. The advantage of packet switching is that the network is only used when there is information to be sent, also a single path can be used by packets from different sources at the same time thus utilising bandwidth more effectively.

In the proposed network, circuit and packet switching co-exist in the same network and only the paths needed for high data throughput are converted to circuit switching, the rest of the network keeps working on packet switching and excludes the circuit switched paths from their routing decision. The decision to change switching depends on the amount of data to be transmitted. A data threshold value defined in section 6.2 is used to determine the type of switching for a specific path. This is also coupled with an increase in the packet size by transport layer for maximum data throughput. The routers are notified about the decision by the “switching type” in data packet header. Thus the routers in the path modify themselves for the new switching type. The circuit switched path formed between the two nodes can be looked upon as an existence of bus as the two communicating cores are connected via a dedicated path for the length of data transmission. Figure 6-2 shows the formation of a circuit switching path between node 6 and 11. Once the circuit switched path is established, it appears invisible to neighbouring nodes and cannot be used as part of their packet switching.

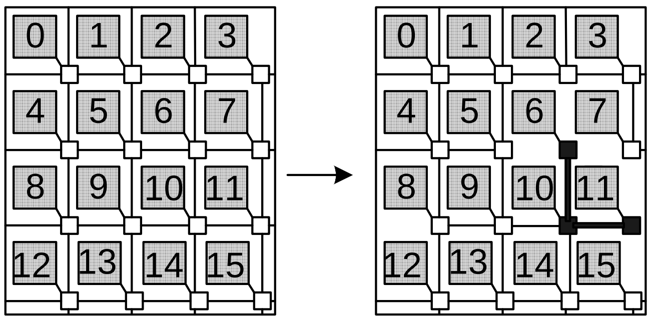


Figure 6-2 change of packet switching to circuit switching between node 6 and 11

5.3.2.1 Routing Algorithm for packet switching

Routing is performed in the proposed drNoC based on concept of wormhole routing. As mentioned in sub-section 3.4.2.1, wormhole routing is prone to live-lock and dead-lock. Three virtual channels per port are employed in the proposed design in order to deal with live-lock and dead-lock situations. The proposed adaptive routing algorithm works in two steps:

Step I: Output paths are selected for control flits by choosing the most profitable channel from the available free channels. Profitable paths correspond to ones that will take the packet closest to its destination. The path, once established by the control flit, is utilised by the data flits since it is based on wormhole routing. If all channels are busy or there is a faulty neighbouring router, the packet is stored in the buffer. The packets following the blocked/stored packets are stopped at the destination by a backward signal from router to source.

Step II: Once the output channel is available for routing, packets are injected from the buffer or the processing core into the network. If the channel is not available, packets are kept stored in the buffer. This process is repeated for all the packets in the buffer before any other packet can be injected into that channel. Once the entire packet from buffer is sent, packets from other incoming ports are injected into the network.

To explain the routing further, consider the network in Figure 6-3. Router "R" has eight neighbours represented by a, *b, c, d, e, f, g*. Router communicates with its neighbours through its input ports W_{i1}, E_{i1}, N_{i1} and S_{i1} and output ports W_{i2}, E_{i2}, N_{i2} and S_{i2} . Let the virtual channels be VC_1, VC_2, and VC_3. Each router tests the state of its neighbours to establish the availability of channel.

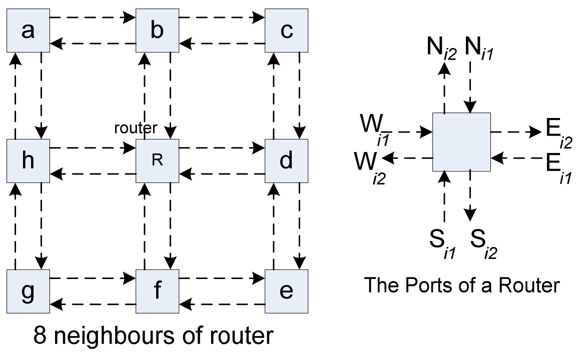


Figure 6-3 Eight neighbours and ports of a router

The message routings according to the current router R (x_s , y_s) and the destination (x_d , y_d) are divided into four types: WE, EW, SN and NS routings, where WE (from west to east) routing is taken if $x_s < x_d$, EW (from east to west) routing if $x_s > x_d$, NS (from north to south) routing if $x_s = x_d$ and $y_s < y_d$, and SN (from south to north) routing if $x_s = x_d$ and $y_s > y_d$. In presence of an unblocked outgoing channel, the router directs the header flit on the shortest path according to Table 6-2. However, if the output channel is busy, then router checks for path with penalty of two more packet hops. In absence of even router's second choice of output channel, the packet is stored in the buffer till the shortest path or the second choice path becomes available.

Table 6-2 Available channel for message routing

Routing Type	**Channels available**
WE routing	In VC_1
EW routing	In VC_1 or VC_2
SN routing	In VC_1, VC_2 or VC_3
NS routing	In VC_1, VC_2 or VC_3

6.3.3 Data Link layer

As mentioned earlier, the data link layer abstracts the physical layer and treats it as a medium with a non-zero probability of errors in the transmitted bit stream. Error correction with retransmission is employed in the proposed network to keep silicon cost low. A checksum is calculated (one's complement sum of all 16 bit words in the header and payload) at the transmitter and the checksum bit is sent with the packet to the receiver where the checksum is calculated again and compared with transmitted checksum to detect any error. At the present stage of network implementation, the checksum is implemented but is not used in the simulations due to the processing overhead associated with it.

6.3.4 Physical Layer

Keeping in view the abundance of wiring resources, separate wires are used for control signals and data transfer. Using separate wires for control signals facilitates transmission of information between routers to avoid network congestion. More details of signals are discussed in next section.

6.4 SystemC Modelling of drNoC

The proposed systemC modelling level described in section 4.3.1 is applied to drNoC. The router is the main building block of NoC. It acts as an interface point of the processing cores to the system. In the proposed design, each router is connected to four neighbouring routers and the interfaced core. The proposed NoC is also given a bus-like interface as mentioned above, which helps to eliminate the use of wrappers and any component designed to be integrated to a bus based platform can be integrated in the proposed NoC in plug and play fashion. The proposed router can be divided into four components (shown in Figure 6-4):

The Input Controller that manages the routing table also known as look-up-table (LUT) and determines the fate of arrived packets after header inspection. The input controller of a router is connected to the input controller of its neighbouring routers. This connection is to update routing tables and pass control signals. Thus when a router is instructed to change mode to circuit switching, it informs its neighbours to exclude the specified path from their routing tables, and remembers the path established till it receives the end of transmission packet. This passage of control signal to neighbouring routers makes it possible to avoid the need for big buffers to store the packets as once a packet is blocked, a control signal is sent backwards till it reaches the source to stop packet injection. The input controller checks the arrived packets at each input port in a round robin pattern to establish the output path.

Input Port that is the point of entry of the incoming packet, it has a buffer to store one packet that is getting inspected for its header contents. Information extracted from an incoming packet includes its destination address and type of switching.

An *Interface module* has been added to deal with the packetisation of data. Each router has one interface module and is present at the port where the component is connected to the router. Its main function involves dealing with handshaking with the connected component and once data has arrived, packet assembly with the correct destination and source address. De-packetisation of packet is also done by interface module. The layered SMS responsible for control of the dynamically reconfigurable NoC is implemented in the interface module.

Finally the *Switching* Logic that connects the input ports to the output ports depending on the instructions from the Input Controller.

The implementation of the system started with modelling a basic NoC router and adding the reconfigurable features into the system. The modelled basic NoC also served in evaluating the performance of the system.

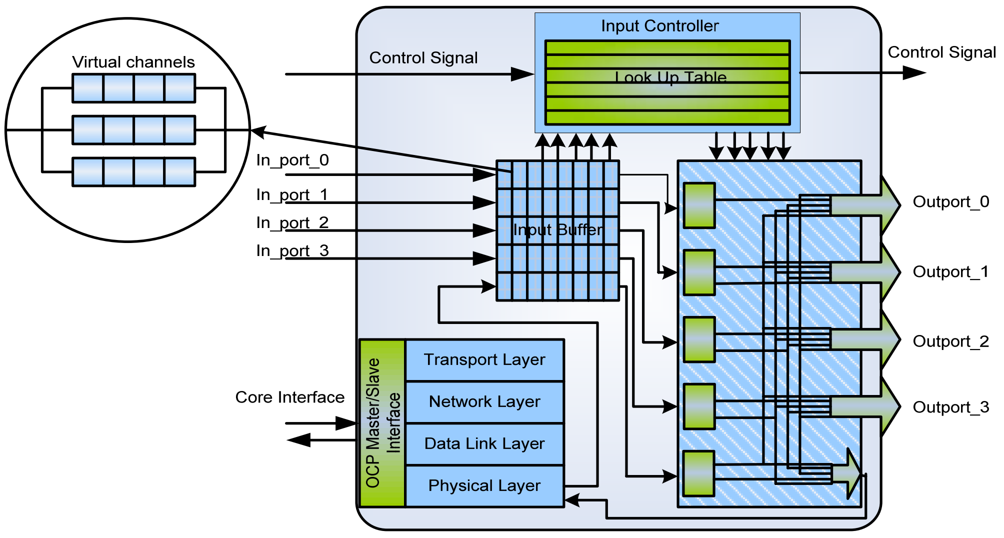


Figure 6-4 Conceptual model of drNoC Router with OCP interface

The node is implemented as an FSM with different states representing different tasks carried out by the interface node or router. Figure 6-5 shows the FSM of the first stage of implementation where a simple router is designed. At the start, the router waits for a packet to arrive at its input nodes. On packet arrival the destination address is read by the input controller. The LUT is then checked to determine the output port for the packet. Every packet can be routed to any of the four output ports, with each route carrying a penalty in terms of packet hop or packet delay. The entries in LUT are according to the routing algorithm explained in 6.3.2.1.

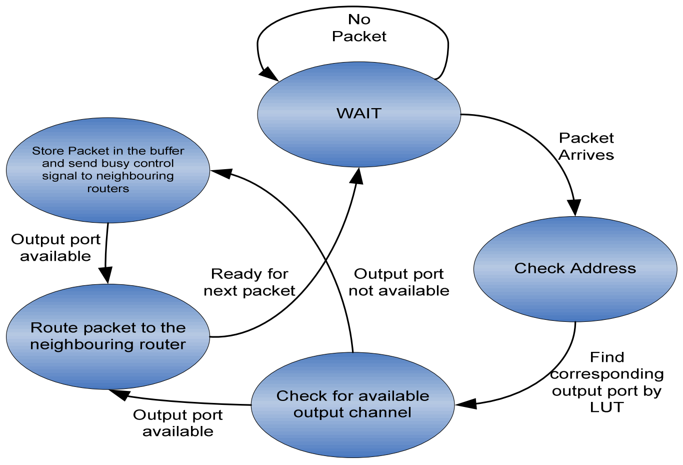

No Packet
WAIT
Packet Arrives
Store Packet in the buffer and send busy control signal to neighbouring routers
Check Address
Output port available
Ready for next packet
Output port not available
Route packet to the neighbouring router
Find corresponding output port by LUT
Check for available output channel
Output port available

Figure 6-5 FSM showing basic NoC flow

If the output port is available for transmission of packet, the packet is routed to the neighbouring router connected to that port, otherwise, the packet is stored in the buffer and a control signal is sent to the packet source to stop data production. The first packet is sent to determine the path and is called a control packet. Once a complete path is discovered, the following packets (data packets) follow the same route.

After implementation of the basic structure of NoC router, the novel features of the proposed NoC router were added in the model. Figure 6-6 shows the FSM of the proposed drNoC router. It starts like the normal NoC waiting on packet arrival. However, once the packet has arrived and its address is decoded, the router checks the data threshold set in the packet. Data threshold is the option included in the proposed drNoC that determines the packet size, switching and routing of the packet and is described in section 6.3.1.

The input controller, after establishing an output channel, acts on the data threshold and sends control signal to neighbouring routers about the routing/switching information. If the core is connected through the OCP socket, all the features of SMS are performed by the router. However, in case the packets are being generated by the connected core, issues like packetisation, routing, threshold decisions are made by the core.

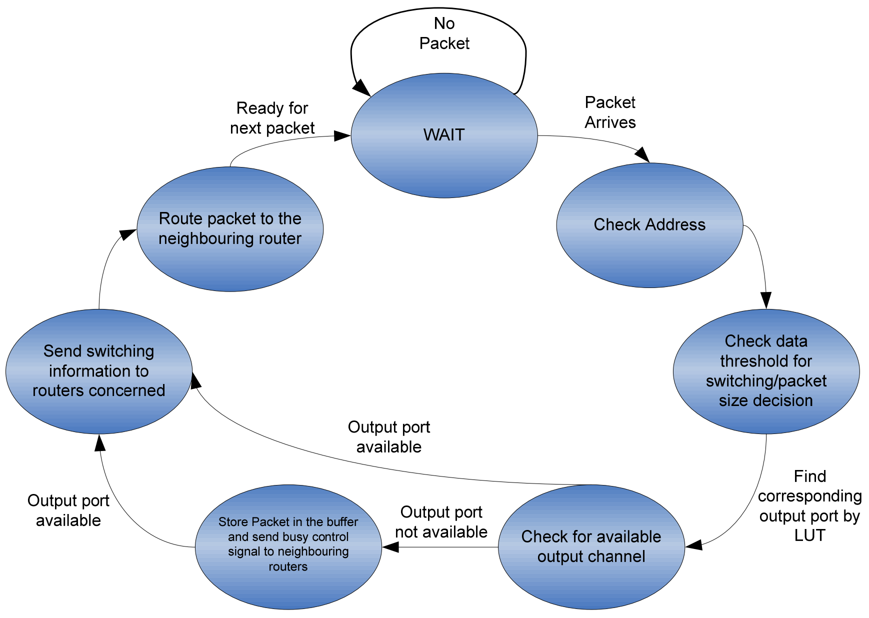

No Packet
Packet Arrives
Ready for next packet
WAIT
Check Address
Route packet to the neighbouring router
Check data threshold for switching/packet size decision
Send switching information to routers concerned
Output port available
Find corresponding output port by LUT
Output port available
Store Packet in the buffer and send busy control signal to neighbouring routers
Output port not available
Check for available output channel

Figure 6-6 FSM showing drNoC flow

When implementing the complete NoC system, different topologies are considered on the basis of degree, maximum distance between two nodes, average distance between nodes and wire cost. 2D-mesh is the most simple 2D network structure consisting of a grid of horizontal and vertical lines with nodes placed at their intersections. 2D Mesh is considered as a preferred topology for NoC architectures due to its simple addressing scheme and predictable inter-node delay. Zhong [ZHO-002] argues that torus theoretically outperforms the 2D-mesh. However due to the complexity involved in keeping the wire lengths the same, the minor performance improvement of torus can be neglected compared to the profit of placement optimization. Network properties for 16 node 2D mesh can be given by [BIJ-001];

Degree = 3-5

Average Distance = $\frac{2}{3}\sqrt{N}$ = 2.6

Wire Cost = $2\left(N - \sqrt{N}\right)$ = 24

A prototype router design is implemented in Verilog. Each input port has a fixed link width of 32 bits. When analysed under 0.18µm technology, router area is found to be

66362.3 µm^2, this area is without the implementation of OCP adapter interface. Area of drNoC is considerably higher than a router for the circuit switched presented by [PHI-002], but when compared with the 32-bit link router for packet switched network in [KIM-001], it is found to be only 0.9% more in area. This increase is area is due to the complex circuitry involved in the implementation of smart network stack.

6.5 Network Analysis

SystemC modelling, introduced in section 4.3.1, is employed to model a network with 16 nodes. The aim of these simulations is to investigate the effect of dynamic reconfiguration of network parameters on the proposed drNoC. Network delay and data throughput is taken for different simulation scenarios to evaluate the proposed drNoC. Traditional NoC is taken as the one with fixed network parameters i.e. switching is packet switching, routing is adaptive (section 6.3.2.1), packet size is fixed. Clock frequency in these simulations is taken to be 100MHz.

Network delay is taken as the round-trip delay for a data packet within the network. Network delay comprises the sum of transmission delays and queuing delays experienced by a packet travelling through the collection of routers. Delay has huge impact on the processing capabilities of the concerned processing element waiting on data from the other processing core. Data throughput on the other hand is taken as the amount of data transferred from one processing core to another through the network over a particular period of time.

In the simulations the terms normal, medium and high data throughput refers to the data threshold defined in section 6.3.1. Thus, in order to mimic this traffic pattern, normal data threshold traffic will be simulated as a small transaction of data injected to the network at a rate of 60 packets/sec or 9.6 Kbits/sec. In the case of medium data threshold packets are injected at a rate of 5000 packets/sec or 800 Kbits/sec. In case of high data threshold, packets are injected at a rate of 65000 packets/sec or 10.4 Mbits/sec. Another difference between high data threshold and medium data threshold is the way data is injected. Medium data threshold follows a bursty traffic burst where data will be injected for only a small period of time, while, high data threshold refers to continuous data getting injected in the network. Data packet size is taken as 160 bits. The 4x4 network used for simulation is shown in Figure 6-7. A 16 core size for NoC is a realistic simulation for near future applications. Different master-slave pairs considered are mentioned in different simulation scenarios.

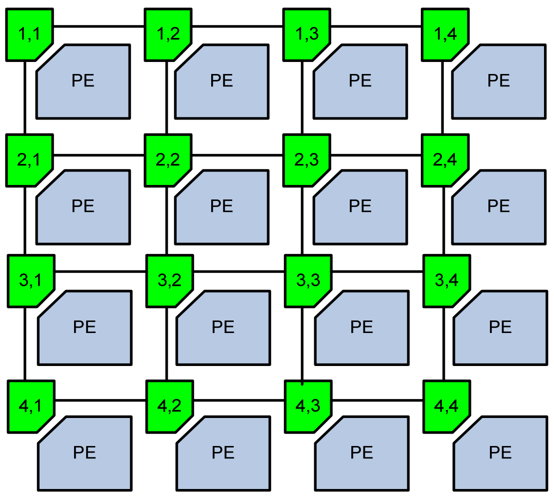


Figure 6-7 Simulated 4x4 2D Mesh network

Scenario 1: Network delay - Traditional NoC vs. drNoC - no contention in the network

The aim of this simulation is to monitor the network delay when there is no contention in the network i.e. routers/channels are only utilised for one master-pair communication. Table 6-3 lists the mater-slave pair scenario considered. 6 cores (3 master-slave pairs) are involved in communication. Link 1 and Link 2 are medium data threshold links i.e. the packet injection will be bursty. Link 3 is high data threshold link and will be injecting packets continuously, hence the reason, the total number of packets is not specified. The simulation is run for 20 seconds.

Table 6-3 Scenario 1: Resource configuration

	Source	**Destination**	**Traffic Type**	**No. of Packets**
Link 1	2,3	3,4	Medium	80000
Link 2	1,3	3,2	Medium	50000
Link 3	1,1	4,2	High	Continuous

Figure 6-8 shows the communicating cores and the simulation results. Under the medium data threshold, the links perform the same way. However, under high data threshold, network delay is decreased by 35.7% when drNoC is used. As mentioned above, network delay is the sum of transmission delay and queuing delay. In the case of Link 1 and link 2, the difference in network delay is due to the number of intermediate nodes involved.

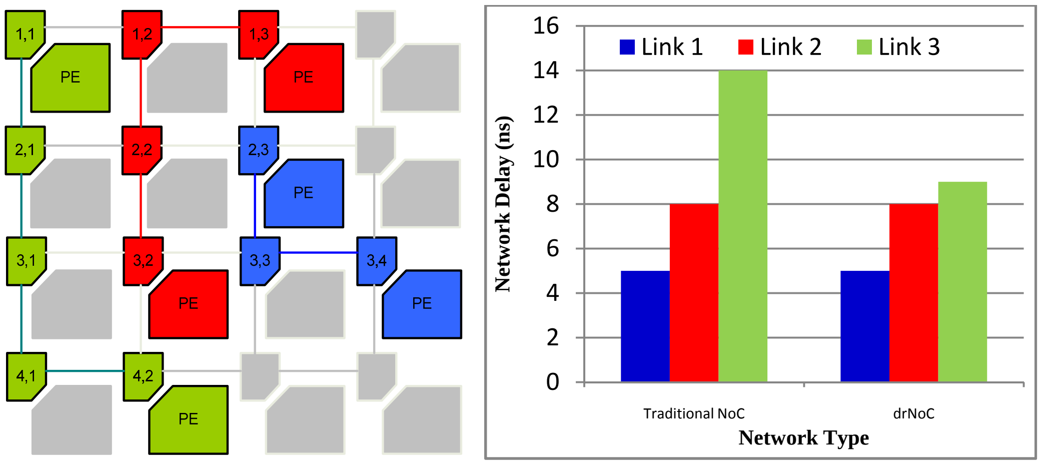


Figure 6-8 Scenario 1: Network delay results

Scenario 2: Network delay - Traditional NoC vs. drNoC - in network with contention

Table 6-4 Scenario 2: Resource configuration

	Source	**Destination**	**Traffic Type**	**No. of Packets**
Link 1	1,3	2,4	Medium	80000
Link 2	1,2	3,2	High	Continuous
Link 3	1,1	3,4	High	Continuous

The aim of this simulation is to monitor the network delay in network with resource contention. Table 6-4 lists the mater-slave pair scenario considered. 6 cores (3 master-slave pair) are involved in communication. Link 1 is medium data threshold links i.e. the packet injection will be bursty. Link 2 and Link 3 are high data threshold link and will be injecting packets continuously, hence the reason, total number of packets are not specified. The simulation is run for 30 seconds. Contention is taken as the sharing of router (3,2) by the data packets of Link 2 and Link 3.

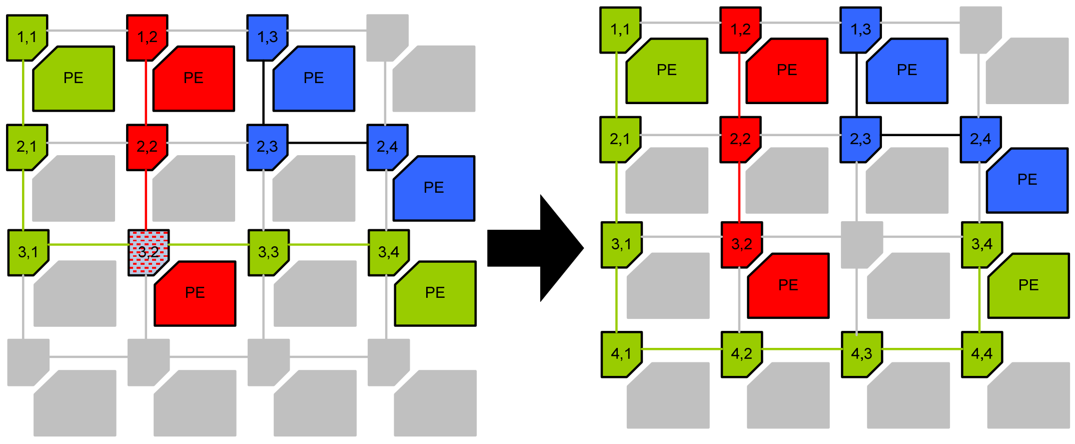


Figure 6-9 Scenario 2 - Path formation in drNoC

Figure 6-9 shows the formation of two circuit switched paths in the network. In the case of traditional NoC, Link 3 takes the shortest possible route with the least resource contention. Link 2 takes the shortest possible route. Both these routes share a common router (3,2). On the establishment of circuit switched network by Link 2, the router (3,2) becomes invisible to the other network and hence Link 3 takes a two hop penalty forming a new route. As Link 3 carries high threshold data, a circuit switched path is formed between (1,1) and (3,4).

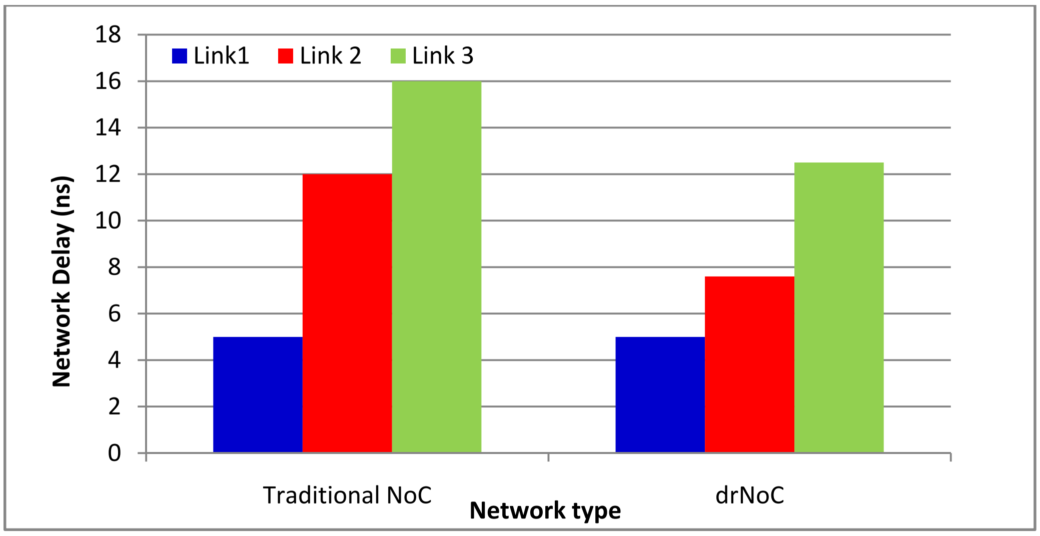


Figure 6-10 Scenario 2 - Network delay results

Figure 6-10 shows the simulation results of scenario 2. The network delay is decreased by 22% in Link 3, and 37% in Link 2. As per the last scenario, the difference in delay decrease ratio is due to the number of intermediate nodes involved.

Scenario 3: Network delay - Traditional NoC vs. drNoC – Fully simulated network

Table 6-5 Scenario 3: Resource configuration

	Source	**Destination**	**Traffic Type**	**No. of Packets**
Link 1	1,1	3,3	Medium	50000
Link 2	2,1	4,2	Medium	100000
Link 3	3,1	2,4	High	Continuous
Link 4	4,1	2,3	High	Continuous
Link 5	1,2	1,4	High	Continuous
Link 6	2,2	4,3	Medium	150000
Link 7	3,2	3,4	Medium	200000
Link 8	1,3	4,4	Normal	200

In order to analyse network delay in a complicated fully simulated network, eight master-slave pairs are made to communicate with different type of data thresholds. Table 6-5 lists the master-slave pairs. This network was simulated for 60 seconds.

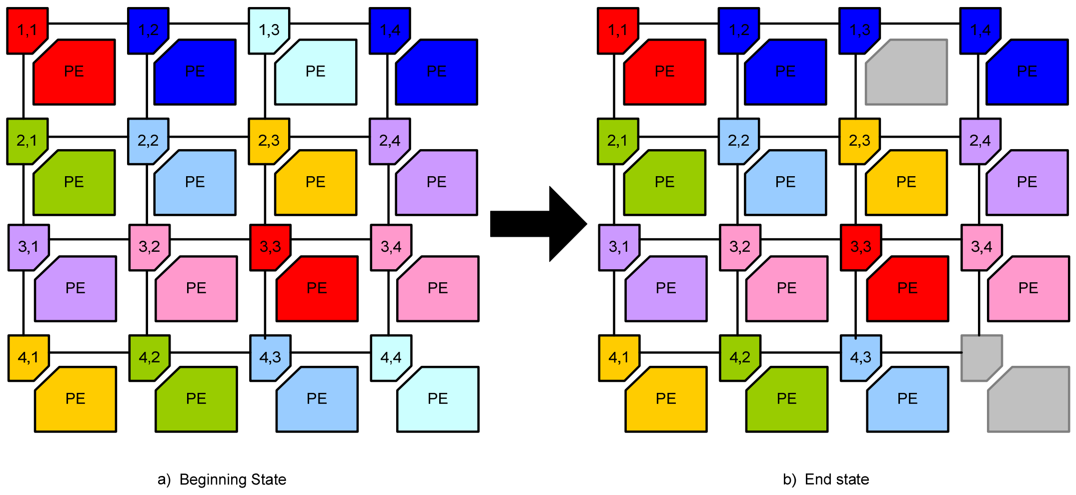


Figure 6-11 Scenario 3 - Path formation in drNoC

The aim of this simulation is to analyse network delay in an environment where the proposed drNoC is not able to perform effectively due to its limitations. Figure 6-11 shows the paths that were established in the considered master-slave scenario. Due to the locality of master-slaves pairs, it became impossible for drNoC to form circuit switching paths in the high data threshold links. There was one circuit switched path formed in Link 5, however, it was formed after the communication ended in Link 8. The other two high data threshold links; Link 3 and Link 4, are forced to continue communication as packet switched due to not being able to establish a path that is not used for any other communication. Even after the medium data threshold links ended communication, due to Link 4 having no alternative path to route its data, Link 3 was forced to communicate as packet switched network.

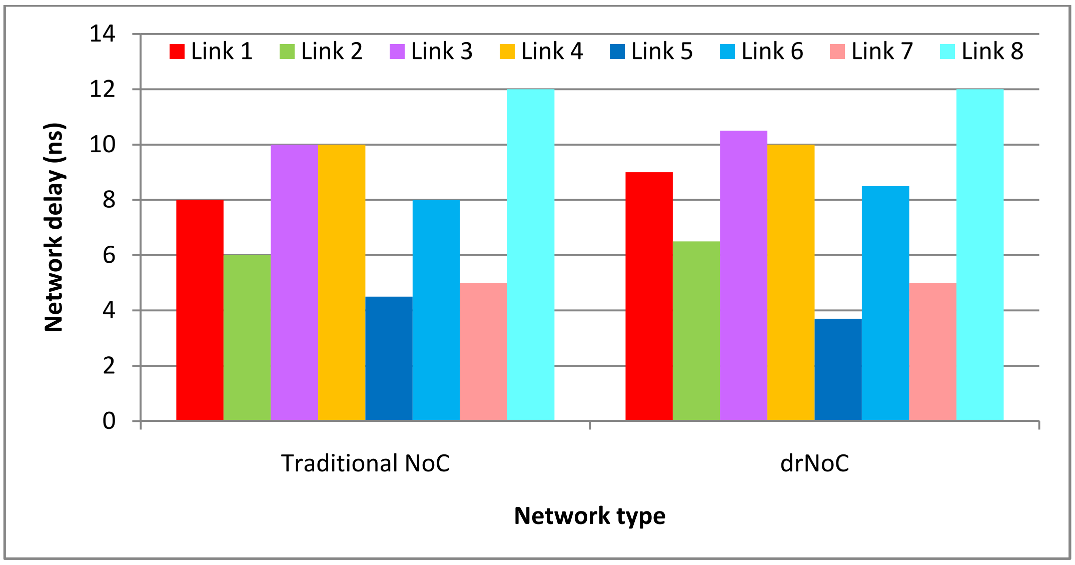


Figure 6-12 Scenario 3 - Network delay results

Figure 6-12 displays the network delay results of the simulation. Unlike the previous scenarios, disappointingly, network delay of Link 1, Link 2, Link 3 and Link 6 increased. This is due to the additional burden on the routers by re-routing of Link 1 packets due to formation of a circuit switched path by Link 5. drNoC did decrease the network delay by 17.7 % for Link5, but overall, there is a 2.7 % increase in network delay. This situation can be avoided by communication centric placement of cores in the network.

Scenario 4: Throughput comparison - Traditional NoC vs. drNoC

The aim of this simulation is to compare the throughput and network delay of a fully simulated drNoC with that of traditionally implemented NoC with fixed parameters as mentioned above. A network with master-slave pairs as listed in Table 6-6 is simulated for 120 seconds.

Table 6-6 Scenario 4: Resource configuration

	Source	Destination	Traffic Type	No. of Packets
Link 1	1,1	1,4	High	Continuous
Link 2	2,1	4,2	Medium	100000
Link 3	3,1	2,4	High	Continuous
Link 4	4,1	4,4	High	Continuous
Link 5	1,2	3,2	Medium	500000
Link 6	2,2	4,3	Medium	200000
Link 7	3,3	3,4	Medium	300000
Link 8	1,3	2,3	Normal	2000

Figure 6-13 shows the formation of circuit switching paths once the medium threshold links have ended communication. Figure 6-14 shows the percentage increase in throughput and percentage decrease in average network delay. It can be seen that over a period of 100 seconds, throughput is increased by 32%, and network delay is decreased by 37%.

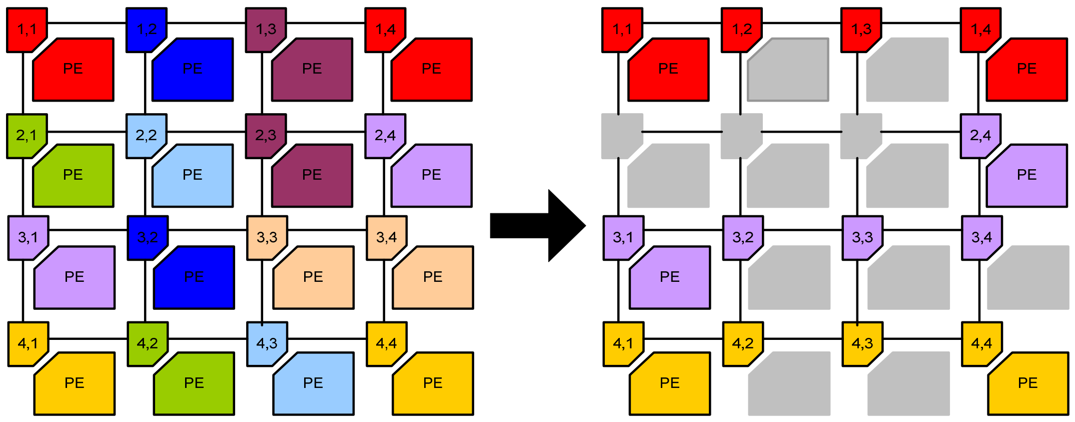
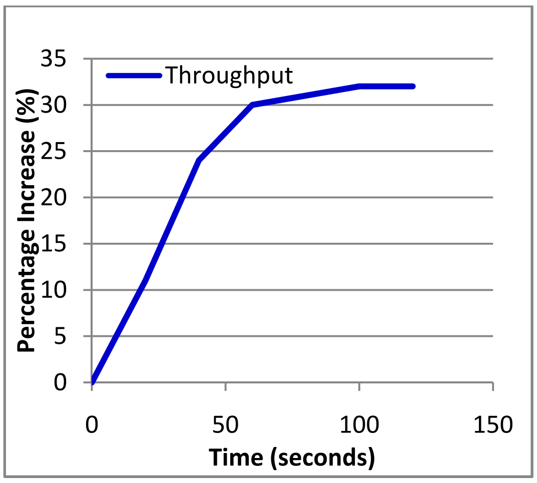


Figure 6-13 Scenario 4- Path formation in drNoC

Data throughput is taken as the amount of data transferred from one processing core to another through the network in a particular time. Increase in throughout in the simulation is due to the establishment of three circuit switched paths once the medium data threshold links have completed transactions. Most of the medium data threshold links ended communication after 60 seconds, hence achieving the peak in data throughput curve around that time. After the paths were established, the high threshold data getting communicated by drNoC increased as the traditional was still using packet switching. The same applies in case of network delay.



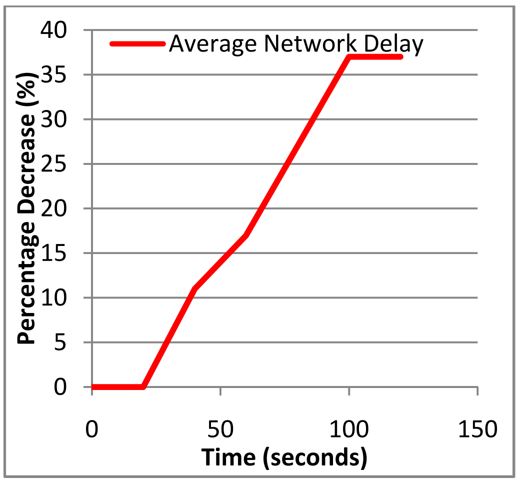


Figure 6-14 Scenario 4 – Percentage Increase in data throughput and decrease in average network delay.

As mentioned in scenario 3, when changing one part of the drNoC, the load from that part is shifted to the packet switched part of the network. Figure 6-15 shows the effect on data throughput of the links by load shifted to normal data threshold and medium data threshold links.

It is noticed that the links can accommodate a 39% increase of shifted load without any degradation to data throughput in case of normal data threshold links and 23% in case of medium data threshold links. An interesting thing to note is the increase in data throughput of the links when the network load is shifted. The decrease in data throughput of the links is compensated for by the increase in delivery time for that certain link.

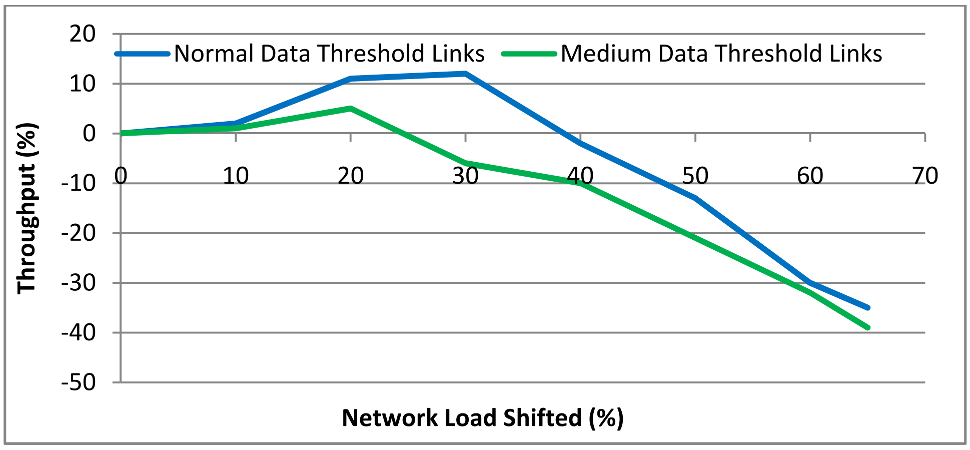


Figure 6-15 Effect of drNoC path formation on rest of the network

Figure 6-16 shows the increase in time required to accommodate the decrease in throughput due to network load shifted. In the case of a normal data throughput link, the delivery time increase by 2% in case of 60% increase in network load causing a 30% decrease in data throughput for that link. In the case of medium data threshold links the delivery time is increased by 3.3% to accommodate the 60% increase in network load that caused a 33% decrease in data throughput for the medium data threshold link. Increase in delivery time affects the QoS requirement of the system. One way to deal with this is to eliminate the cause of decreased data throughput of these links by proper placement of cores so the formation of paths for high data threshold links does not affect the medium and normal data threshold links.

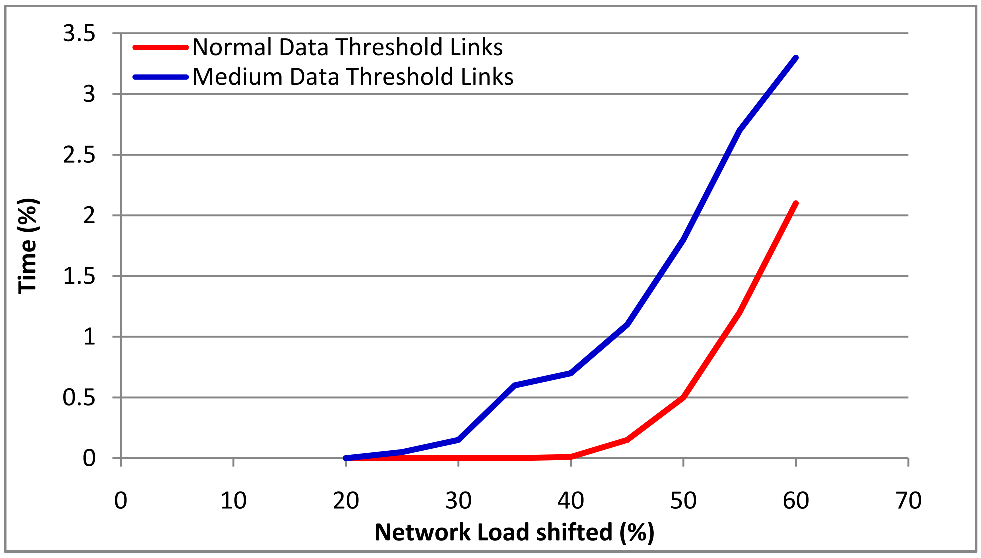


Figure 6-16 Effect of network load shifting on time

Scenario 5: Suitability of routing algorithm

In order to monitor the effectiveness of the proposed drNoC routing algorithm versus traditional wormhole and XY routing algorithms, a simulation was carried out under mixed traffic pattern and different master-slave pairs communicating at different times. Table 6-7 lists the resource configuration for this simulation. Unlike the scenarios above, the continuous nodes were made to stop and start after some time in order to check the adaptability of the routing algorithm.

Table 6-7 Scenario 5: Resource configuration

	Source	**Destination**	**Traffic Type**	**No. of Packets**	**Start Time (second)**
Link 1	1,1	1,4	Medium	30000	0
Link 2	2,1	4,2	Medium	100000	0
Link 3	3,1	2,4	High	Continuous	100
Link 4	4,1	4,4	High	Continuous	30-45
Link 5	1,2	3,2	Medium	50000	0
Link 6	2,2	4,3	Medium	20000	0
Link 7	3,3	3,4	Medium	30000	45
Link 8	1,3	2,3	Normal	2000	60
Link 9	2,2	3,4	Medium	100000	60
Link 10	2,1	2,3	Medium	80000	0
Link 11	3,3	4,2	High	Continuous	100

Figure 6-17 shows the comparison of routing algorithms. It can be seen that by employing XY routing, which is deterministic in nature, the reliability of network, in terms of delivery of packets has decreased. However, wormhole routing increased in delivered packets (decrease of non-delieverd packets in the Figure 6-17). On the other hand, the proposed drNoC routing algorithm which is based on wormhole routing but with virtual channels, the percentage of delivered packets is higher than that of womhole routing. This is due to the highly adaptive nature and the use of virtual channels as a guard against livelocks and deadlocks. On the change of a link to circuit switching due to a high data threshold link, it appears invisible to the rest of the network. Thus, routing that is deterministic in nature cannot cope with the traffic. Comparing the graph with the resource configuration table, whenever a circuit switched link is formed, the percentage of undelivered packets increases. Leading up to a maximum of 50% when network is left only with high data threshold links.

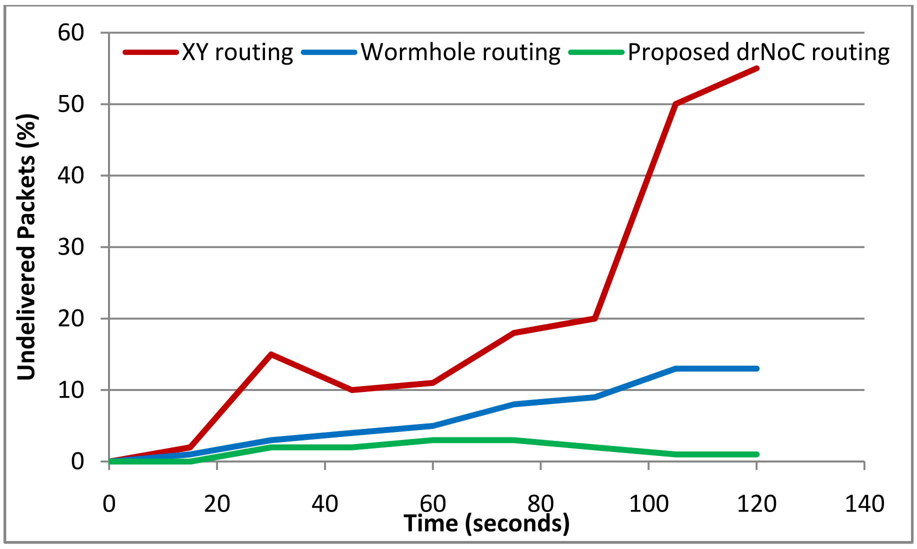


Figure 6-17 Scenario 5- comparison of routing algorithms on drNoC

6.6 Explanation of Simulations

The aim of these simulations was to establish the effectvieness of proposed drNoC for data intensive applications. For comparison reasons, a traditional NoC was first implemented in systemC based on the concepts of NoC architecture proposed by [BEN-002] and [SUN-001]. In these application specific architectures, the architectural parameters, switching, routing and packet size is fixed at design time. This approach is effective for SoC architectures of a particular application domain, but for future SoC architctures with the existence of multi-domain traffic on the communication network, this will result in a waste of resources and valuable bandwidth.

In the proposed drNoC, the switching, routing and packet size in the network changes depending on the traffic getting transmitted, thus, allowing effective utilisation of available bandwidth. As seen in the simulation results, the proposed drNoC outperforms the traditional approach of fixed parameters. With the formation of a circuit switched path, the network load is shifted to the packet switched links. The advantage of this load shifting is the optimal utilisation of link bandwidth, but it also comes at a cost of affecting the performance of packet switched links if the load increases over 23% in the case of medium data threshold links (data throughput requirement between 100Kbits/sec to 10Mbits/sec) and over 39% in normal data threshold links (data throughput requirement under 100Kbits/sec).

Analysing the network delays in a fully simulated network reveals the importance of placement of cores on the network. Due to the formation of circuit switched links, it appears invisible to the remaining routers i.e. these links cannot be used for switching data packets of other communicating cores. Formation of circuit switching paths only occurs if there is an alternative path available for the other data on the network. Thus, as seen in the simulations, if the placement of cores on network is not done effectively, this can cause degradation of the overall performance (scenario 3 in section 6.5).

With the abundance of wiring resources in SoC, the control signal mechanism implemented for adaptive routing does not require a big overhead in resources. However, the formation of different data paths (circuit switching/packet switching) carry with them an overhead in terms of latency, which again, is negligible in high data intensive links. The only significant overhead is in terms of area of the router, especially with support for OCP socket standard. However, the advantages of drNoC exceed its disadvantages.

6.7 Summary

In this chapter a dynamically reconfigurable NoC was proposed. In a traditional NoC design, the architectural parameters, like, switching, routing and packet size are fixed. With communication in future heterogeneous SoC architectures, especially with reconfigurable cores, this will prove highly in-efficient due to the network resources not getting utilised effectively, causing wastage of bandwidth. The proposed drNoC endeavours to solve this problem, by dynamically reconfiguring its nodes to match the required bandwidth. On-chip traffic from high data throughput links is shifted to links with low data throughput and a circuit switched path is formed for the high data throughput links. Simulation results have shown the effectiveness of this approach in cases where cores are placed according to their throughput requirement.

The router in drNoC, that also acts as an interface point for integration of cores, is given an OCP socket based interface, allowing cores to be integrated without having the need for wrappers. This makes drNoC a very attractive communication centric platform for future high data throughput applications. The increase in area is compensated by the increase in bandwidth provided by drNoC.

Chapter 7

Conclusion and Future Work

7.1 Introduction

The focus of this thesis has been to investigate the on-chip communication architectures suitable for future high data intensive multi-core architectures. Particular emphasis has been given to communication centric platform based designs ranging from shared bus based to complex on-chip packet based networks. This has paved the path to establish the requirements for future interconnect architectures.

This chapter begins with the summary of material presented in each chapter of this thesis. This is followed by conclusions drawn from the work presented and evaluates the extent to which the desired aim of this thesis has been accomplished. Finally, areas of future work are outlined.

7.2 Summary of Thesis

In today's electronic industry, rapid development of SoC architectures plays an important role in the success of the product. Platform based designs favour design re-use and are seen as an efficient way to reduce design time.

With the latest DSM technologies, designers not only have to keep in mind the data throughput, but low power and reduced area is of prime importance as well. Communication media are becoming a bottleneck in developing high throughput, low power heterogeneous SoC architectures, especially when targeting custom reconfigurable cores. In recent years, many on-chip interconnect architectures have emerged to deal with this communication bottleneck. Many commercial communication centric platforms have also emerged, addressing communication issue as well as the issues of design reuse.

Chapter 2 introduced the concept of a platform based design approach that led to the emergence of communication centric platform based designs. It discusses the basics of on-chip communication and gives an overview of evolution from a simple shared bus to commercially developed communication framework, which aims to reduce the SoC design time under the area, power and throughput constraints. Chapter 2 also lists the architectural parameters that are developed to improve the interconnect performance in the DSM era.

NoC is seen as a solution to the communication bottleneck arising from the scaling down of device size and scaling up of design complexity. Chapter 3 gives an overview of the concept of NoC. It explains the layered approach taken in designing NoC by discussing different design aspects of the different layers and the architectural tuning parameters associated with them. Chapter 3 also summarises the current work done by different researchers in the field of NoC.

Four different communication centric platform based designs, shared bus based, crossbar based, hierarchical bus based and hybrid communication medium based are proposed to deal with the different communication requirements of different application domains. Chapter 4 describes the developed platforms. The steps taken to develop a platform architecture where components, i.e. reconfigurable cores, RISC based processors, fixed cores, memory modules etc. can be integrated in a plug and play fashion without the need of programming the verified communication centric platforms, was discussed in this chapter. Chapter 4 also introduced the novel concept of hybrid communication medium where two different interconnect architectures co-exist to realise the complete communication architecture of the system. The modelling strategy taken in the development of systemC models of these platforms are also described in Chapter 4, along with the simulation results and porting a real life example of WiMAX on the proposed platforms.

A tool called SOCCAD is proposed to automate the development of SoC architectures. Chapter 5 gives an overview of some of the tools developed by research institutes and by industry to ease the process of SoC architecture development and quick simulation. It describes how communication centric platforms, proposed in chapter 4 are used to automate the development process. Automated communication centric placement of cores is then discussed. The steps involved in the development and working of the tool are described and demonstrated by a real life example.

Chapter 6 describes the proposed dynamically reconfigurable packet based on-chip network called drNoC. The novelty of the proposed NoC architecture lies in its ability to reconfigure itself with the changing communication requirements of the system. It describes the layered approach taken to realise the architecture and control of drNoC, focusing on the parameters like switching, routing, topology etc. Chapter 6 also argues how drNoC can be an ideal candidate for a communication centric platform. It lists the steps involved in systemC modelling of the drNoC and the simulations carried out to analyse the proposed dynamically reconfigurable network.

7.3 Summary of achievements

The main achievements of this work is the development of four communication centric platforms, development of hybrid communication medium where two different interconnect architecture co-exist in a system and development of dynamically reconfigurable NoC architecture. The four communication centric platforms include, shared bus based, crossbar based, hierarchical based and hybrid communication medium based. As demonstrated by simulation results in section 4.5.4 and the real life example of WiMAX in section 4.6, each developed platform has different throughput, area and power characteristics, thus making them suitable for different application domains. An OCP socket based approach is taken when designing the interface for platform. The platforms have built in controller dealing with the addressing, communication and control of the integrated cores in the platform. This has a benefit that cores (reconfigurable, processors, memory modules etc.) can be integrated in a plug-an-play fashion without the need of reprogramming the controller.

The proposed hybrid communication medium combines the advantages of both crossbar medium (high throughput) and shared bus based medium (low power). Simulation results and implementation of WiMAX receiver as a real-life example shows a 65% increase in data throughput than shared bus based communication medium, 13% decrease in area and 11% decrease in power than crossbar based communication medium. As concluded in chapter 4, this proves to be a good compromise for applications with power and throughput constraints.

In chapter 4, it was also concluded that placement of cores on the communication platform plays an important role in the overall communication media performance. To automate the generation of SoC architectures with optimised communication media centric placement of cores, a tool called SOCCAD is developed. The cores needed to implement the required SoC architecture can be chosen from the library, the SOCCAD tool automatically generates the communication architecture with optimised placement of cores. The tool also gives overall power and area figures of the system implementation with different communication platforms for comparison reasons. Finally, the tool can generate the overall systemC code for the system to allow quick simulation and verification.

The proposed drNoC changes its characteristics with the changing communication requirements of the system. As demonstrated in section 6.5, the reconfigurable characteristics have the advantage of low network delay thus prove to be suitable for high data intensive applications. The proposed drNoC has also been given an OCP socket based interface to allow for cores to be integrated into drNoC with ease. This makes the proposed drNoC a suitable candidate for the communication centric platforms. Simulation results have shown a 32% increase in data throughput and 22-35% decrease in network delay when compared with a traditional NoC with fixed parameters.

7.4 Conclusions

As mentioned earlier, the aim of the thesis was to review the current on-chip communication architectures and to establish the requirements for future on-chip communication medium.

In during this process, a thorough review of traditional shared bus based communication medium was conducted. Shared bus based medium provides a low power communication architecture and occupy less area on silicon. However, it has its limitation when it comes to data throughput. To tackle this problem, the concept of hierarchical bus originated. SoC components are placed at appropriate level in the hierarchy according to the performance level they require. A bridge is use to connect the parallel buses. Hierarchical buses provide an increase in data throughput over the shared buses due to decreased load per bus and the potential for transactions to proceed in parallel on different buses.

Crossbar switch bus architecture connects multiple inputs to multiple outputs in a matrix manner. The crossbar switch provides simultaneous multiple connections between its inputs and outputs. It provides the highest data throughput amongst the other bus based architectures at a cost of highest power and occupies the highest area on silicon. In order to find a compromise between the two extremes (shared bus and crossbar), a hybrid communication medium is developed in this thesis that provides better throughput than shared and hierarchical bus and lower power and area than crossbar based communication architectures.

Hybrid communication medium provides a shared bus and a crossbar matrix integrated together via a bridge. Analysis of hybrid medium and hierarchical bus has revealed the importance of placement of cores on the communication medium. A tool called SOCCAD has been developed that deals with the automated generation of SoC architectures. The SOCCAD tool provides a communication centric placement of cores on the communication medium for optimal utilisation of the interconnect architecture.

In future data intensive application, with data throughput requirements of over 9GB/s, bus based communication architectures will not suffice. A notion of NoC is considered as a viable solution to provide the required performance. A research into current NoC architecture was also conducted as part of this thesis. Developed with fixed architectural parameters like packet size, switching and routing, current NoC architectures do not utilise the network resources efficiently.

A NoC architecture is developed to overcome the short comings of the current NoC architectures. The developed drNoC, dynamically reconfigures its characteristics to utilise the network resources efficiently and to provide an increased QoS requirement in terms of data throughput. Simulation results have shown its effectiveness when simulated under different traffic patterns expected from future applications.

7.5 Future Work

This thesis has endeavoured to provide a rigorous investigation in the field of future on-chip communication and it opens doors for future research and development.

The communication centric platforms are developed to prove the hypothesis that having different communication media platforms can facilitate the development of SoC architectures optimised for different application domains. This could not be achieved by utilising the commercially available single communication medium based platforms since a compromise has to be reached in terms of throughput, power and area.

The developed platforms need to be further optimised in terms of their architectural parameters e.g. GALS clocking, pipelining, handshaking and different arbitration schemes to mention a few. The concept of hybrid medium can be further exploited by combining more communication architectures e.g. hierarchical bus with crossbars. Similarly, drNoC can be further improved by doing research in clocking strategies, optimising routing schemes, optimising the hardware implementation for area etc. For making the developed architectures more reliable, research is needed in QoS aspects, thorough testing and verification methodologies are also to be investigated.

In terms of development aspects, the SOCCAD tool can be improved by including more components in the library, including the tool’s ability to implement the SoC architecture for more communication architectures like drNoC, employing advanced placement algorithms e.g. genetic algorithms for optimised placement of the cores on the platform. Giving the SOCCAD tool drag and drop features and the ability to find the best SoC architecture on basis of their throughput is also to be developed.

There is a need for benchmarking software for analysing the developed drNoC. Unfortunately, at present, there isn’t any benchmarking tool for NoC that can test the performance of drNoC under realistic traffic patterns. Thus, development of such bench mark is also intended in future.

In short, combining the communication centric platforms and drNoC with the current communication media optimizing schemes can help develop optimised interconnects for future multi-core SoC architectures.

References

[AHM-001] Balal Ahmad, Ali Ahmadinia, and Tughrul Arslan, "Hybrid Communication Media for Adaptive SoC Architectures". Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), 5-8 August 2007. Edinburgh, U. K., pp. 373-378.

[AHM-002] Balal Ahmad, Tughrul Arslan, "Dynamically Reconfigurable NoC for Reconfigurable MPSoC", Proceedings of IEEE 2005 Custom Integrated Circuits (CICC 2005), 18-21 September 2005. San Jose, USA. Pages: 277-280.

[AHM-003] Balal Ahmad, A. T. Erdogan, S. Khawam, "Architecture of a Dynamically Reconfigurable NoC for Adaptive Reconfigurable MPSoC". Proceedings of IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2006), 15-18 June 2006. Istanbul, Turkey. Pages: 405- 411.

[AHM-004] Balal Ahmad, Ali Ahmadinia, Tughrul Arslan, "Dynamically Reconfigurable NoC with Bus Based Interface for Ease of Integration and Reduced Design Time". Proceedings of IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2008), 22-25 June 2008. European Space Agency, Noordwijk, Netherlands. Pages: 309-314

[AHM-005] Balal Ahmad, Ali Ahmadinia, Tughrul Arslan, "Communication Centric Platform Based Designs for Future MPSoC". 6th International Bhurban Conference on Applied Sciences and Technology 2008. January 09. CESAT, Pakistan

[AHM-006] Ali Ahmadinia, Balal Ahmad, A T Erdogan, Ahmet T, Tughrul Arslan, "SystemC-based Reconfigurable IP Modelling for System-on-Chip Design". Proceedings of IEEE NASA/ESA Conference on Adaptive

Hardware and Systems (AHS 2008) ,22-25 June 2008. European Space Agency, Noordwijk, Netherlands. Pages: 362-367

[AHM-007] Ali Ahmadinia, Balal Ahmad, Ahmet Erdogan, Tughrul Arslan, “SystemC-based Custom Reconfigurable IP Cores for Wireless Applications”. Proceedings of Engineering of Reconfigurable Systems and Algorithms (ERSA 2008), 14-17 July 2008. Las Vegas , USA.

[AHM-008] Ali Ahmadinia, Balal Ahmad, Tughrul Arslan, “High-Level Power Estimation for Multi-Standard Wireless Systems.”. Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2008), 7-9 April, 2008. Montpellier, France, pp. 275-280.

[AHM-009] Ali Ahmadinia, Balal Ahmad, and Tughrul Arslan, “Communication Centric Modelling of System on Chip Devices Targeting Multi-Standard Telecommunication Applications”. Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 7-9 April, 2008. Montpellier, France, pp. 209-214.

[AHM-010] Ali Ahmadinia, Balal Ahmad, and Tughrul Arslan, “System Level Reconfigurable FFT Architecture for System-on-Chip Design.”, Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), 5-8 August 2007. Edinburgh, U. K., pp. 169-175.

[AHM-011] Ali Ahmadinia, Balal Ahmad, Tughrul Arslan, “System-level Modelling and Analysis of Embedded Reconfigurable Cores for Wireless Systems.”. Proceedings of International Conference on Field-Programmable Logic and Applications (FPL 2007). 27-29 August 2007. Amsterdam, Netherlands, pp. 757-760.

[ALT-001] Altera's Excalibur, www.altera.com/products/devices/excalibur/exc-index.html

[ALT-002] Altera Avalon interface specification. Altera corporation. http://www.altera.com

[APT-001] APT Research group, http://intranet.cs.man.ac.uk/apt/publications/

[APT-001] The Advanced Processor Technologies group at Manchester University http://intranet.cs.man.ac.uk/apt/projects/processors/amulet/

[ARC-001]	ARC International www.arccores.com
[ARM-001]	AMBA System Architecture - www.arm.com/products/solutions/AMBAHomePage.html
[ARM-002]	ARMS PrimeXsys: www.electronicstalk.com/news/ank/ank000.html
[ARM-003]	ARM. ARM7 TDMI Datasheet. ARM. http://www.arm.com.
[ATI-001]	David Atienza, Federico Angiolini, Srinivasan Muralia, Antonio Pullinid, Luca Benini, Giovanni De Micheli,"Network-on-Chip design and synthesis outlook", The VLSI journal, Received 26 November 2007; received in revised form 23 December 2007; accepted 27 December 2007
[AYA-001]	J. Ayala, M. Lopez-Vellejo, D.Bertozzi, and L. Benini, “State-of-the-art soc communication architectures,” in Embedded Systems Handbook, R. Zurawski, Ed. Boca Raton: CRC Press, 2006, pp. 20.1–20.22.
[BAI-001]	William J. Bainbridge- PhD Thesis, "Asynchronous System-on-Chip Interconnect",http://intranet.cs.man.ac.uk/apt/publications/thesis/bainbridge00_phd.php
[BAK-001]	*Bakoglu, H.B.* “Circuits, Interconnects and Packaging for VLSI*”*. MA:Addison-Wesley, 1990
[BEN-001]	Luca Benini, Giovanni De Micheli, "Powering Networks on Chips: Energy-Efficient and Reliable Interconnect Design for SoCs," ISSS,pp.33-38, Proceedings of the 14th international symposium on Systems synthesis (ISSS '01), 2001
[BEN-002]	Luca Benini, Giovanmi De Micheli,"Networks on chip: a new paradigm for systems on chip design"In Proceedings of Conference on Design, Automation and Test in Europe 2002.
[BEN-003]	*Luca benini, Giovanni De Micheli* , “Energy Reliability Trade off for NoCs”, A. Jantsch and H. Tenhunen, Network on Chip, 107 – 129.

[BEN-004] T.T. Ye, L. Benini, G. De Micheli, “Packetization and routing analysis of on chip multiprocessor networks”, Journal of Systems Architecture, vol. 50, 2004, pp. 81-104.

[BER-001] *D. Bertsekas, R. Gallager*, “Data Networks”. Prentice Hall, 1991.

[BIJ-001] Bas Bijlsma, "Asynchronous Network-on-Chip Architecture Performance Analysis",Master’s thesis, Department of Electrical Engineering, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology.September 2004-September 2005.

[BJE-001] T. Bjerregaard and S. Mahadevan. “A survey of research and practices network-on-chip”. ACM Computing Survey, 38(1):1–54, 2006.

[CAR-001] Jordi Carrabina et al "Bus-centric SoC Architecture Generation Tools", found at http://cephis.uab.es/resources/pdf/papers/GSPX_2004_UltraWizard.pdf

[CHA-001] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, Lee Todd, “Surviving the SOC Revolution”, p. 71, Kluwer Academic Publishers, Boston, 1999.

[CHI-001] A.A. Chien and J.H. Kim, “Planar-adaptive routing: Low-cost adaptive networks for multiprocessors,” Proc. 19th Ann. Int’l Symp. Computer Architecture, pp. 268–277, May 1992.

[COW-001] CoWare Platform Architecture, www.coware.com/products/platformarchitect.php

[DAL-001] *Dally W. and Towles B.*, “Route packets, not wires: Onchip interconnection networks”. ProcDAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

[DAL-002] W. J. Dally, “Performance Analysis of a k-ary n-cube Interconnect Networks” ,. IEEE Transactions on Computers, June 1990, pp. 775-785.

[DAL-003] W. J. Dally, “ Virtual channel flow control”. IEEE Trans on parallel and Distributed Systems, 3(2):194-205 Mar. 1992.

[DAL-004]	W. J. Dally, H. Aoki, “Deadlock -free adaptive routing in multicomputer networks using virtual channels”, IEEE Trans. on Parallel and Distributed Systems, April 1993.
[DAL-005]	W. J. Dally and B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufman Publishers, 2004.
[DUA-001]	*J.Duato, S. Yalamanchili, L. Ni*, “Interconnection Networks, an Engineering Approach”, IEEE Computer society press, 1997.
[DUA-002]	J. Duato, “ A necessary and sufficient condition for deadlock free adaptive routing in wormhole networks,” Proc. Int’l Conf parallel processing, Aug 1994.
[FEI-001]	U. Feige, P. Raghavan, “Exact analysis of hot-potato routing”, Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, October 1992.
[FEL-001]	S. A. Felperin, L. Gravano, G. Pifarre and J. Sanz, “Routing Techniques for massively parallel communication”, IEEE proceedings, 79(4):488-503, 1991.
[FRE-001]	Freescale Semiconductor. Implementing WiMAX PHY Processing Using the Freescale MSC8126 Multi-Core DSP. Freescale Semiconductor. http://www.freescale.com.
[GAJ-001]	D. Gajski et al., “SpecC: Specification Language and Methodology”, Kluwer Academic Publishers, 2000
[GIV-001]	T. Givargis,F. Vahid,"Platune: a tuning framework for system-on-a-chip platforms"IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 21, Issue 11, Nov 2002 Pages: 1317 - 1327
[GLA-001]	C.J Glass and L.M. Ni, “ Fault tolerant wormhole routing in meshes”, Proc. IEEE 23rd Int’l Symp. Fault tolerant Computing pp. 240-249, 1993.
[GOO-001]	K. Goossens, J. Dielissen, J. Meerbergen, P. Poplavko, A. R˘adulescu, E. Rijpkema, E. Waterlander, and P. Wielage. Networks on Chip,

chapter Guaranteeing The Quality of Services. Kluwer Academic Publisher, 2003.

[GRO-001] T. Grötker, S. Liao, G. Martin, S. Swan. “System Design with SystemC”. Kluwer Academic Publishers, 2002

[GUE-001] P. Guerrier and A. Greiner. A generic architecture for on-chip packetswitched interconnections. In Proceedings of the Design, Automation and Test in Europe Conference, pages 250–256, March 2000.

[HAN-001] Han Ke; Deng Zhongliang; Shu Qiong,"Verification of AMBA Bus Model Using SystemVerilog" 8th International Conference on Volume Electronic Measurement and Instruments, 2007. Issue , Aug. 16 2007-July 18 2007 Page(s):1-776 - 1-780

[HEM-001] Ahmed Hemani, Axel Jantsch, Shashi Kumar, Adam Postula, Johnny Öberg, Mikael Millberg, Dan Lindqvist."Network on a Chip: An architecture for billion transistor era" Proceeding of the IEEE NorChip Conference, November 2000.

[HUA-001] Huang, R.; Vemuri, R., Analysis and evaluation of a hybrid interconnect structure for FPGAs. Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on 7-11 Nov. 2004 Page(s):595 - 601

[IBM-001] The CoreConnect™ Bus Architecture , white paper - http://www-01.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture

[IEE-001] IEEE 802.16e: Air interface for fixed and mobile broadband wireless access systems. 2005. http://standards.ieee.org/getieee802/802.16.html

[IMP-001] Improv Systems www.improvsys.com

[INT-001] Intel Corporation. Intel NetStructure WiMAX Baseband Card Product Brief. Intel Corporation. http://www.intel.com.

[ITR–001] International Technology Roadmap for Semiconductors web site http://public.itrs.net.

[JAL-001] A. Jalabert, S. Murali, L Benini, G. De Micheli "×pipesCompiler: a tool for instantiating application specific networks on chip".Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2004. Volume 2, Issue , 16-20 Feb. 2004. Pages: 884 - 889.

[JAR-001] Jari Nurmi, Hannu Tenhunen, Jouni Isoaho, Axel Jantsch, "Interconnect-centric Design for Advanced SoC and NoC", Published by Springer, 2004.

[JON-001] H. Jonathan Chao, Necdet Uzun, “An ATM Routing and Concentration Chip for a Scalable Multicast ATM Switch,” IEEE Journal of solid state circuits, Vol. 32, No. 6, June 1997, pp. 816-828.

[KAR-001] *F. Karim, A. Nguyen, S. Dey*, “On-chip Communication Architecture for OC-768 Network Processors” Proceedings of 38th Design Automation Conference, June 2001.

[KEU-001] *K. Keutzer*, “ Chip level assembly (and not integration of synthesis and physical) is the key to DSM Design”, Proceeding of the ACM/IEEE International Workshop on Timing Issues in the specification and Synthesis of Digital Systems, March 1999.

[KEU-002] K. Keutzer et al. “System-level design: Orthogonalization of concerns and platform-based design,” In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Dec. 2000

[KIM-001] Daewook Kim, Manho Kim and Gerald E. Sobelman,"Design of a High-Performance Scalable CDMA Router for On-Chip Switched Networks" nternational SoC Design Conference,October 20-21, 2005,Korea.

[KYE-001] Kyeong Keol Ryu, Eung Shin, V. J Mooney, "A comparison of five different multiprocessor SoC bus architectures" Proceedings of Euromicro Symposium on Digital Systems, Design, 2001.page(s): 202-209.Warsaw, Poland.

[LAH-001] Lahiri, K.; Raghunathan, A.; Dey, S.”Evaluation of the traffic-performance characteristics ofsystem-on-chip communication architectures” Fourteenth International Conference on VLSI Design, 2001.Volume , Issue , 2001 Page(s):29 – 35

[LAH-002] K.Lahiri, S. Dey, A. Raghunathan, Design of Communication Architectures for High-Performance and Energy-Efficient System-on-Chips, in Multiprocessor Systems-on-Chips , Morgan Kaufmann, September 2004. pp. 187–222.

[LIN-001] *S.Lin and D.J Costello*. " Error Control Coding : Fundamentals and applications". Prentice Hall Inc.,1983.

[LYO-001] D. Lyonnard,S. Yoo, A. Baghdadi,A.A. Jerraya,"Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip" Proceedings of Design Automation Conference, 2001.pages: 518- 523.

[MAR-001] J. Hu and R. Marculescu, "DyAD - Smart Routing for Networks-on-Chip", DAC 2004, June 2004,San Diego, California, USA.

[MCK-001] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks", computer, vol. 26, 1993, pp. 62-76.

[MEN-001] SystemVision by Mentor Graphics, www.mentor.com/products/sm/system_integration_simulation_analysis/systemvision/

[MIL-001] Milica Mitic and Mile Stojceev,"An overview of on-chip buses",published in book "Digital Systems and Applications" Published by CRC Press, 2007.

[NIL-001] E. Nilsson, *M. Millberg, J. Oberg, A. Jantsch* ,"Load distribution with the proximity congestion awareness in a network on chip", DATE 2003, pp 1126-1127.

[OCP-001] Open Core Protocol Specifications, http://www.ocpip.org/socket/ocpspec/

[OSC-001] Open SystemC Initiative (OSCI) - http://www.systemc.org

[PAL-001] Palmchip Corporation, www.palmchip.com

[PAL-002] Palmchip, overview of coreframe architecture, white paper. Palmchip Corporation. http://www.palmchip.com

[PAS-001] Sudeep Pasricha, Nikil Dutt, "On-Chip Communication Architectures: System on Chip Interconnect", Published by Morgan Kaufmann, 2008

[PAS-002] Sudeep Pasricha, Nikil Dutt, Mohamed Ben-Romdhane, " Using TLM for Exploring Bus-based SoC Communication Architectures",Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05)

[PHI-001] Philips nexperia : www.nxp.com/products/nexperia/

[PHI-002] Nx-builder by Philips, http://www.design-reuse.com/articles/9769/philips-semiconductors-next-generation-architectural-ip-reuse-developments-for-soc-integration.html

[PHI-003] Pham Phi-Hung,Y. Kumar,Kim Chulwoo,"High Performance and Area-Efficient Circuit-Switched Network on Chip Design", The sixth IEEE International Conference on Computer and Information Technology, 2006. CIT apos;06.Pages:243 - 243

[POR-001] PORTO, Marcelo S.; SILVA, Thaísa L. da.; PORTO; Roger E. C.; GÜNTZEL, José Luís; AGOSTINI, Luciano. "Design and Comparison between PVCI, BVCI and OCP Hardware Reuse Interfaces Mapped to FPGA." In: STUDENT FORUM ON MICROELECTRONICS, 3. aceito para apresentação oral e publicação nos anais. Available online at http://minerva.ufpel.edu.br/~guntzel/publicacoes.html.

[QUI-001] Quick Logic, http://www.quicklogic.com/

[RAH-001] Rahul Bhatt, Giovanni De Micheli,Daniel D. Gajski,Christopher K. Lennard,Stan Liao,John Sanguinetti,Patrick Schaumont, Rajesh K. Gupta and Kaushik Roy."System-on-Chip Specification and Modeling Using C++: Challenges and Opportunities", In IEEE D&T May/ June 2001

[RIJ-001] E. Rijpkema et al., "Tradeoffs in the design of a router with both guaranteed and best effort services for network on chip", IEE Proceedings : Computers and Digital Techniques, vol. 150, 2003, pp

294-302.

[SAL-001] E. Salminen, V. Lahtinen, K. Kuusilinna, and T. Hämäläinen, "Overview of bus-based system-on-chip interconnections," in *ISCAS*, vol. 2, Scottsdale, AZ, 2002, pp. 372-375.

[SAN-001] Alberto Sangiovanni-Vincentelli and Grant Martin, A Vision for Embedded Systems: Platform-Based Design and Software Methodology, IEEE Design and Test of Computers, Volume 18, Number 6, November-December, 2001, pp. 23-33.

[SAN-002] Michael Santarini, "Sonics boosts silicon backplane IP network; ARM lands licensing pair" www.eetimes.com/news/design/columns/core_competency/showArticle.jhtml?articleID=17407787

[SHA-001] Shashi Kumar, et. al, "A Network on Chip Architecture and Design Methodology", IEEEComputer Society Annual Symposium on VLSI, Pittsburgh,Pennsylvania, USA, April 2002.

[SHI-001] C.C. Su and K.G. Shin, "Adaptive Fault tolerant dead lock free routing in meshes and hypercubes", IEEE Trans. Computers, 45(6): 666-683. June 1996.

[SHI-002] K. G. Shin and S. W. Daniel. Analysis and implementation of hybrid switching.IEEE Tran. on Computers, 1996.

[SIL-001] Silistix - http://www.silistix.com/chainworks.php

[SON-001] Sonics network technical overview. Sonics inc. www.sonicsinc.com

[SON-002] Sonic Studio by Sonic Inc. , http://www.sonicsinc.com/sonicsstudio.htm

[SUN-001] Yi-Ran Sun, Shashi Kumar, Axel Jantsch,"Simulation and Evaluation for a Network on Chip Architecture Using Ns-2",NOrChip 2002

[SYL-001] *Sylvester D. and Hu C.* "Analytical modelling and characterization of deep submicron interconnect". Proceedings of the IEEE, May 2001.

[SYN-001] Design ware - coreAssembler by synopsis, www.synopsys.com/products/designware/ipreuse_tools.html?BAC-HP&Link=Product_Pulldwn_Home

[SYN-002] Galaxy Design Platform by synopsis, www.synopsys.com/products/solutions/galaxy_platform.html

[TEN-001] Tensilica Technologies www.tensilica.com

[TEN-002] Xtensa Xplorer by Tensilica , www.tensilica.com/products/devtools/hw_dev/sw_xplorer.htm

[TEX -001] TI OMAP www.mistralsoftware.com/html/company/partners/ti_omap_center.htm

[THE-001] *T. Theis*, “The future of Interconnection Technology,” IBM Journal of Research and Development, Vol. 44, No. 3, May 2000, pp. 379-390.

[UMI-001] Umit Y. Ogras and Radu Marculescu,"Prediction-based Flow Control for Network-on-Chip Traffic"DAC 2006, July 24-28, 2006, San Francisco, California, USA.

[VIS-001] Virtual Socket Interface Alliance's http://www.vsia.org/

[WAR-001] *J. Walrand, P. Varaiya*, “High-Performance Communication Networks”. Morgan Kaufman, 2000.

[WIJ-001] Wijetunga, P.,"High-performance crossbar design for system-on-chip" The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, 2003. Proceedings, 30 June-2 July 2003 Page(s):138 – 143.

[WIM-001] WiMAX Forum http://www.wimaxforum.org/home/

[WIN-001] D. Wingard and A. Kurosawa, “Integration Architecture for System-on-a-Chip Design,” IEEE Custom Integrated Circuits Conference 1998, pp. 85-88.

[WIS-001] Wishbone system-on-chip interconnection architecture for portable IP cores, revision: B.3. http://www.opencores.org/ projects.cgi/web/wishbone/wishbone

[WUJ-001]	J. Wu, “A deterministic fault-tolerant and deadlock-free routing protocol in 2-D meshes based on odd-even turn model” Proceedings of the 16th international conference on Supercomputing, 2002.
[XIL-001]	Xilinx Virtex, http://www.xilinx.com/products/devices.htm
[YIM-001]	Joon-Seo Yim et al. “A C-Based RTL Design Verification Methodology for Complex Microprocessor”, In Proc of DAC, 1997
[YOO-001]	Hoi-Jun Yoo, Kangmin Lee, Jun Kyong Kim, "Low-Power NoC for High-Performance SoC Design" Published by CRC Press, 2008.
[ZHO -002]	M. Zhong, “Evaluation of deflection-routed on-chip networks,” Master’sthesis, KTH, Stockholm, Sweden, June 2005.
[ZHO-001]	J.P. Zhou and F.C.M. Lau, “Fault-tolerant wormhole routing in 2D meshes,” Proc. 2000 International Symposium on Parallel Architectures, Algorithms and Networks, Dec. 2000.