1-1-2008

Network-On-Chip Topology Generation and Analysis For Transaction-Based Systems-on-Chip

Victor. Dumitriu
Ryerson University

NETWORK-ON-CHIP TOPOLOGY GENERATION AND ANALYSIS FOR TRANSACTION-BASED SYSTEMS-ON-CHIP

by

Victor Dumitriu
Bachelor of Engineering
Ryerson University, 2006

A thesis
presented to Ryerson University
in partial fulfilment of the
requirements for the degree of
Masters of Applied Science
in the Program of
Electrical and Computer Engineering

Toronto, Ontario, Canada

I hereby declare that I am the sole author of this thesis or dissertation.

I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

Victor Dumitriu

I further authorize Ryerson University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Victor Dumiţriu

Network-on-Chip Topology Generation and Analysis for Transaction-Based Systems-on-Chip

Victor Dumitriu
Masters of Applied Science, 2008
Program of Electrical and Computer Engineering
Ryerson University

Abstract

The Network-on-Chip concept is emerging as a promising new method of addressing the communication requirements of complex Systems-on-Chip. However, network design at this level must take into consideration the specific communication protocols of on-chip components. This thesis presents a topology analysis and design method for networks-on-chip based on the transaction-oriented protocols common to on-chip systems. The generated topologies target the latency of critical links in the system, while the analysis method can predict the degree of contention in a system prior to the simulation phase. The proposed topologies are tested using various applications, including an MPEG4 Decoder, and are found to perform the same or better than regular topologies, while using less network resources. The contention prediction method is found to be accurate to within 27% in the worst case scenario.

Acknowledgment

I would like to thank my supervisor, Dr. G. N. Khan, for his guidance and support throughout my masters studies. I also wish to acknowledge the financial support provided by the National Science and Engineering Research Council of Canada (NSERC), the Canadian Microsystems Corporation (CMC), and Ryerson University.

Contents

List of Figures

List of Tables

Chapter 1

Thesis Introduction

1.1 Introduction

The constant stream of advances made in the field of IC manufacturing, coupled with the ever increasing demand for high-performance embedded systems has led to the adoption of the System-on-Chip (SoC) design process. At the high end of the SoC scale are to be found the Multi-Processor Systems-on-Chip (MPSoC), which integrate multiple processing elements, such as Digital Signal Processors (DSP) and general purpose processors as well as dedicated hardware units. Such systems are primarily used in the communications and multimedia fields, where large information volumes are handled. To address communication performance and efficiency on the chip, the Network-on-Chip (NoC) concept has been proposed as a new approach to communication infrastructure, replacing traditional systems such as shared buses or crossbars [1].

The NoC concept has its roots in the field of parallel computing. However, the migration to on-chip communications handling requires a different communication performance model than that found in the field of parallel computing. The difference in complexity between a processing node in a parallel computer and a processor in an embedded systems requires

that NoC design be tackled at a lower abstraction level. In addition, MPSoCs often hav different requirements than high-performance parallel computers; the easy addition of addi tional processing elements is not as important as the area and power characteristics of th system. In fact, the majority of current on-chip systems cannot change their internal struc ture once implemented in silicon. Regular topologies are not always as desirable as irregulai application-specific topologies, since they may include circuitry that remain unused.

The current work presents a topology generation and analysis method for NoC desigi based on the transaction-oriented communication methods of on-chip systems. The topology generation process creates dedicated topology descriptions based on application requirement and network structural parameters. The aim is to provide the required communicatio infrastructure for an application using a minimum of resources, so that efficient application specific systems are generated. In addition, to accelerate the design process of such systems the proposed method incorporates a performance prediction process referred to as Contentio Analysis. The process attempts to predict interference effects in the system, and based o this makes a recommendation as to the needed operating frequency of the network. The method minimizes complex computation by considering network components in isolation and uses a simplified form of simulation rather than queue analysis. This approach can b less accurate than complete modelling of the whole system, but it reduces the complexity and execution time of the analysis method.

1.2 Motivation and Contribution

The main motivation for the proposed work is to address a perceived gap in the field of NoC automated design and analysis. This gap refers to the current trend of modelling on-chip communication behavior at a high level of abstraction similar to that used for wide-area computer networks or parallel super computers. Such approximations rarely match on-chip

system behavior, in particular in cases where component interfacing is accomplished using transaction-based protocols, thus making them inappropriate models. The work presented in this thesis represents an attempt to address the topic of network-on-chip design while keeping in mind the behavioral aspects of current on-chip components.

For this reason, the main focus of the work will be the analysis of performance requirements.

The main contribution of the proposed work is three-fold. First and foremost, the thesis formally establishes the conditions that must be met in order to meet performance requirements in transaction based on-chip systems (which represent the majority of such systems). The second main contribution is the development of two algorithms aimed at the generation of irregular NoC topologies. Both algorithms are aware of the limitations of transaction-based systems and incorporate this information into their structure. Finally, the thesis presents an analysis method for the estimation of packet behavior in an active network, and incorporates this method into the topology generation process. The analysis method uses Petri Net representation of the system and a partitioned approach to system analysis which allows it to execute faster than in-depth analysis methods such as those based on queue theory.

1.3 Thesis Organization

Before examining networks-on-chip, and their design, it is important to establish what types of systems will use such communication infrastructure, and what their characteristics are. Chapter 2 will briefly discuss the field of on-chip systems, and in particular the topic of Multi-Processor Systems-on-Chip (MPSoC). The chapter will conclude with an analysis of the communication protocols generally used by such systems, and their characteristics. Chapter 3 will fully introduce the Network-on-Chip concept, and present some examples of existing NoC

designs. It will also survey the available research in the field of NoC design and optimization since this is the main discussion topic of the Thesis.

Once the NoC concept has been described, along with the fundamental characteristics o on-chip systems, the main topic of the Thesis is presented. Chapter 4 describes the topology generation and analysis processes being proposed. Because the primary aim of the topology generator is to achieve a desired level of performance, a method is needed to determine if a particular network with given characteristics can achieve said level of performance. Chapte 5 introduces a simulation environment designed specifically to model NoC communication and describes the characteristics and implementation of the simulation models. Chapter 6 presents tests and results used to verify the proposed methods. The results section can be broadly divided into two sections: a comparative section, where the generated topologies are compared with regular ones to determine their benefits and short-comings; and an analysis section which examines the accuracy of the proposed analysis methods, by verifying whether the recommended NoC operating frequency allows the generated topology to meet application communication requirements. Finally, Chapter 7 concludes the Thesis report, and is followed by the Appendix and Bibliography sections.

Chapter 2

Systems-on-Chip: Characteristics and Methodologies

Before delving into the subject of Network-on-Chip and their design, digital Systems-on-Chip (Soc) should be explored, as they are the reason for the emergence of the NoC concept. This section briefly presents the emergence and rise in popularity of the SoC concept, culminating with the Multi-Processor System-on-Chip (MPSoC). The section also discusses the progress made in design methods, and the emergence of standardized interfacing in on-chip components. Finally, once the details of on-chip protocols are established, it is possible to derive some characteristics of on-chip communication, particularly as it relates to achievable performance. This last portion of the chapter also analyzes the main differences between on-chip systems and larger networks, from the communication point of view.

2.1 The Rise of the Systems-on-Chip Concept

The improvements made in the field of integrated circuit manufacturing have led to a constant increase in the number of transistors that can be integrated into a single chip. This fact,

coupled with new requirements for embedded systems, such as reduced power and area, have led designers to integrate various types of components into one chip [2]. Systems that would have consisted of multiple ICs can now be implemented as Systems-on-Chip (SoC). Such systems could incorporate a processor, memory and interfaces to other components, as well as an on-chip interconnect. This approach yields savings not just in terms of area (since only one chip is now used) but also power, since chip boundaries are no longer crossed as often, and special drivers are no longer needed. Finally, having most system components on-chip allows a designer more freedom in certain instances; for example, pin-count limitations are largely eliminated in the case of on-chip connections, meaning that serialization is not always necessary when connecting components.

As the demand for increased performance in such systems continued to increase, designers began to adopt more complex SoC structures. The operating frequency could not be increased indefinitely to provide adequate performance, due to physical constraints as well as power consumption considerations. As a solution, multiple processing elements were introduced on the chip. Such a solution yields added performance, but the power requirement is not as large as in multi-chip systems, since chip boundaries are not crossed as often, reducing the need for added drivers off-chip. Such processing elements include instruction-set processors, Digital Signal Processors (DSPs) and dedicated logic circuits. Such systems are currently known as Multi-Processor Systems-on-Chip (MPSoCs), and are being adopted for use in the multimedia and communications fields, where large data volumes are processed. An example of such a system is shown in Figure 2.1, where the Samsung S3C6400 platform is shown [3]. This SoC is aimed at the mobile telephone market, as well as media players, and incorporates special dedicated processing elements in the form of video and audio encoders and decoders. It is important to note that, unlike chip multi-processors or multi-core processors, the components that make up an MPSoC are heterogeneous, both in their structure and their behavior. Examples of components one might find in such SoCs include general

purpose processors, digital signal processors, direct memory access (DMA) units, memory or Ethernet controllers; such components have different behaviors, and can lead to different communication patterns.

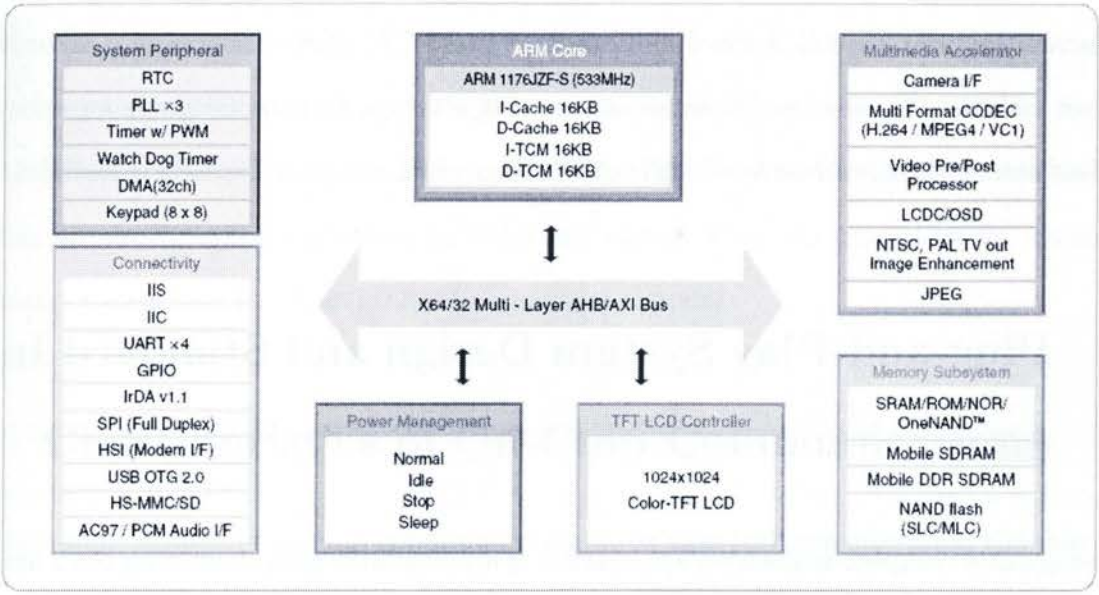

Figure 2.1: Example of a Current-Generation SoC

Unfortunately, the increased number of on-chip components, and the increased volume of information that has to be handled are bringing to light a new problem area within complex SoCs: the on-chip interconnect. Early on-chip systems worked quite well using only a shared bus, primarily because the bus was used by one component (the processor), while the remainder of on-chip systems were passive. Unfortunately, increasing the number of components that can actively use the communication medium leads to congestion; in such instances, the shared bus quickly becomes a bottleneck of system performance [4, 5]. As a solution to this problem, the fabric switch has been adopted as the on-chip interconnect for complex SoCs; examples include the AMBA Multi-Layer Protocol [6] and the Avalon interface [7]. The fabric switch can be thought of as a collection of shared busses with associated arbitration

units, each of which is associated with one of the system slave interfaces. Because of this, the fabric switch is the ideal interconnect, minimizing latency and maximizing throughput for multiple communicating components. However, as the number of communicating components increase, the size of the interconnect quickly explodes, not just in terms of silicon area, but also in the use of metal layers for interconnection [4, 5]. Networks-on-chip are emerging as a new solution to on-chip communication, and attempt to provide similar performance with less associated area overhead and complexity, when compared to fabric switches.

2.2 Plug-and-Play System Design and Standard Interfaces

The design of a complex embedded application is a formidable task, involving both hardware and software components that must interact with each other. Traditionally, such a design would take considerable work, both in the design of the various components as well as their interfacing. In situations where the available time-to-market is short, it is not always possible to design all system components specifically for the current design. The solution being proposed and used with increasing regularity is to re-use earlier component designs, and allow them to be easily interconnected in various configurations [8]. A new industry section is emerging whose only role is to design Intellectual Property (IP) digital components for use in on-chip systems, often referred to as *cores*. Such IP cores can be licensed for a particular design, and can be customized to a varying extent. This allows system designers to concentrate on the design of the application, and on the efficient interfacing of components, rather than on the design of each circuit. This approach has become very wide-spread, with available cores ranging form processors (ARM [9] and NIOS II [10]) to dedicated processing elements and I/O controllers.

To be truly useful, the concept of re-useable design must allow designers to build systems quickly and efficiently. The interfacing of components in particular is an important aspect of re-usable design [11]. Standard protocols and interfaces have emerged in conjunction with embedded cores to allow efficient interconnection. Examples include the AMBA Bus [12], as well as the Avalon Interface [7]. More recently, the move has been towards specialized, point-to-point oriented protocols, which hide all characteristics of the interconnect from the cores themselves. The Open Core Protocol (OCP) [13] and AMBA AXI protocol [14] both take this approach; neither protocol includes any signals that are related to the underlying interconnect structure (such as arbitration request signals).

2.3 Characteristics of On-Chip Communication

Any discussion regarding on-chip communication must take into consideration the characteristics that such communication has. Since most on-chip components rely on specific on-chip protocols for communication, examining these protocols is mandatory for forming a model of communication that can then be used for design purposes (as this Thesis attempts to do). This final section will present some of the characteristics common to a number of on-chip protocols, as well as some advanced features that have been introduced for high-performance designs. Based on this behavior, a set of observations and specifications will be derived for on-chip components. Finally, on-chip communication will be briefly compared with communication methods found in larger systems such as parallel computers.

2.3.1 On-Chip Protocols

All on-chip protocols emerged as a result of the need to allow logic circuits to communicate with each other. At their core, most protocols are built around information transactions, which are themselves divided into request and response phases [7, 12, 13, 14]. In all cases,

the transaction occurs between a master device that can initiate transactions, and a slav device that is passive and responds to transactions. Components with both master and slav characteristics incorporate both master and slave interface logic (an example is a DMA unit) During the request phase, master components assert signals to indicate that a transaction i required, and drive transaction information on dedicated ports (address, data, transactio type, etc). In the case of older protocols, such as the AMBA AHB protocol, the master woul also assert dedicated request signals used by the interconnect controller for arbitration. Th slave component responds to the transaction when it is ready, using a number of handshak signals, in the response stage. During the time of the transaction both interfaces are engage and cannot undertake other tasks. Figure 2.2 shows a transaction in the AXI protocol. Th master components initiates the read transfer by asserting the ARVALID signal, and write address and transaction information on the address channel [14]. The slave latches thi information and asserts the ARREADY signal; the slave then responds when it is ready b asserting the RVALID signal, and the transaction completes (and read data is transferred when the master component asserts the RREADY signal.

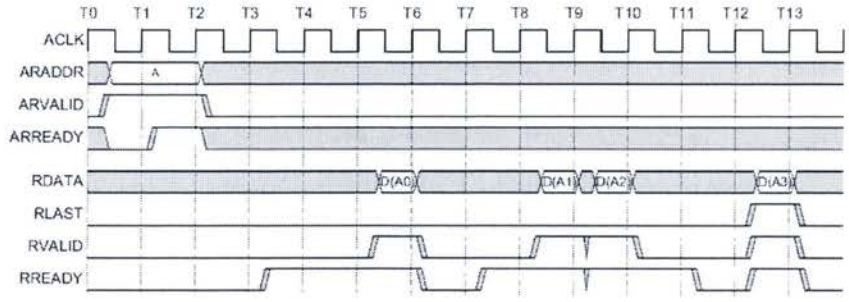

Figure 2.2: AXI Read Transaction

A number of features were added to on-chip protocols to increase performance. One o the first, and certainly the most wide-spread feature is the burst transfer, which was born in shared bus systems, where multiple transactions could be grouped under one arbitration

session, increasing throughput. The burst transfer is now a common feature of all on-chip interconnects, including fabric switches [7]. In addition, more advanced features have been implemented in an attempt to increase the achievable performance. A number of protocols support the concept of split transfers, where a slave device can suspend the transfer while it prepares for the transfer. This allows other master devices to use the interconnect while a slow slave processes a transaction. Once the slave completes processing, it can signal the interconnect to inform it that it is ready to complete a transaction.

The latest generation of on-chip protocols include the support for multiple outstanding transaction to different slave components, and the completion of transactions out of order [13, 14]. In these cases, transactions can complete in or out of order, depending on the configuration and capabilities of the master and slave devices. This feature guards against the case of components being stuck waiting for one slow slave device, and can drastically increase the over-all performance of the system. However, it should be noted that only complex components, such as large embedded processors (ARM 11J series, for example) can take advantage of such features. Simpler, or dedicated component do not have the behavioral support to issue transaction requests to multiple slave devices, and often do not need to do so. However, regardless of how they are grouped or arranged, each transaction still consists of the request and response stage, which must be completed.

Some exceptions exist to the above rules, which are used in certain situations. One example used in processing streaming information is a serial arrangement of components, connected using special, simplified, one-way interfaces. One such example is the Avalon Streaming Interface, which is built around source and sink interfaces [7]. In this case, the processed data is flowing in one direction, and is passed from one component to the next. In such cases, data transfers occur without the complex hand-shaking seen in the protocols described above. The Avalon Streaming interfaces supports flow control through pack-pressure, where sinks can temporarily stop incoming data, if they become saturated. Such interfaces,

however, are used primarily for dedicated applications.

2.3.2 On-Chip Communication Characteristics

The aim of the above discussion and this entire chapter is to establish some characteristics and formal requirements for on-chip communications. As the previous section shows, most on-chip protocols are transaction-based and go through specific stages, which makes it possible to establish a relationship between the latency of a transaction (the time taken for a transaction to complete) and the achievable throughput of an interface such as [15, 16]. The following paragraphs will describe this relationship and how it is derived. Since most existing on-chip protocols specify that both interfaces are engaged during a transaction, this means that latency becomes the dominating factor affecting throughput [16]. In particular, because the interface blocks during a transfer, latency and throughput become inverses of one another. This statement holds in all cases except streaming interfaces and advanced components that support multiple outstanding transfers to separate slaves. However, in the case of advanced components, the analysis presented here can be considered a worst-case scenario.

Based on the blocking nature of on-chip protocols and interfaces, the following definitions can be made. First, the latency of a transaction can be defined as the time that a master interface is engaged during a transfer. In situations where any transaction (read or write) has a request and response phase, this delay is composed of the transport delay (due to the interconnect) and the processing delay of the slave involved. Equation 2.1 defines this relationship:

$$Lat_t = Lat_{int} + Lat_{slave} \tag{2.1}$$

Secondly, because the interface blocks during the transaction and cannot be used for alternative transfers, transactions occur serially, and the achievable throughput becomes the inverse

of the transaction latency Lat_t. In this case, if a specific throughput T_{req}, in transactions per unit time, is required, then the following condition must hold:

$$Lat_t \leq \frac{1}{T_{req}} \tag{2.2}$$

Note that the above requirement does not take into account the processing time of the master interface itself. The latency per transaction can be computed in various ways, to account for burst transfers and encoding options, but the condition specified by Equation 2.2 must ultimately hold, in situations where interface blocking occurs.

The above equations contrast heavily with the behavior of large networks, such as those found in parallel computers and general computer networks, where latency and throughput are unrelated. It is not the networks themselves that cause this difference, but rather the behavior of the communicating components that access said networks. In both these cases, the elements accessing the network are complex entities, consisting of at least a processing element, memory and some sort of network interface. An example of such a system is the BlueGene L computing node, which is itself an SoC [17]. The node consists of two processing cores, a dedicated memory hierarchy including multiple levels of cache and external memory and an integrated router. What this means is that the node itself is capable of operating autonomously while sending messages. In addition, many such systems rely on message-passing rather than transactions for communication, meaning that response stages are not required and the node can continue with other work after the message is sent. In such cases, the communication requirement can be abstracted to an information stream being sent in one direction from the source to the destination (the messages, essentially). Such a stream is often represented as a number of bits per second, and communication constraints are met if the following inequality holds:

$$Data_{ij} \leq BW_{ij} \tag{2.3}$$

Above, $Data_{ij}$ represents the information volume (bits per second) that must be sent between nodes i and j, and BW_{ij} represents the available bandwidth of the network between nodes i and j. Note that the bandwidth does not refer only to the links in the system, but also to any interim elements such as routers or switches, which must process and forward the data being transported by the network. The above equation can only be used in on-chip systems in specific situations. Specifically, it holds in the case of write transfers, provided that the protocol supports posted write transfers, meaning that a response phase is not required for write transactions. The AXI interface, for example, is not accurately modelled by the above inequality.

2.4 Conclusion

The chapter covers a relatively wide range of topics in very short form, primarily for the purpose of introducing the concept of MPSoCs, re-usable designs and standard interfaces and protocols. Given that these are the characteristics of current embedded systems, it is important to establish the underlying ground work before moving specifically to the area of networks-on-chip. Essentially the chapter presents the environment that NoCs are used in, as well as the specific characteristics of the communication protocols used by on-chip components.

Arguably the most important section of the chapter is the last, which formally specifies certain criteria for meeting throughput requirements when components with standard interfaces (such as AMBA AXI or OCP) are used. These criteria have to be considered when designing any interconnect for a specific application, irrespective of the actual interconnect structure. This information is presented first primarily because of chronological considerations. The standard interfaces used on-chip pre-date the Network-on-Chip concept, and their associated behavior and requirements exist independent of the actual interconnect used.

Chapter 3

Networks-on-Chip

Having established the behavior and requirements of on-chip systems in the area of communication, it is now time to fully introduce the subject of Networks-on-Chip (NoC), as well as the field of NoC design automation. The chapter starts with a brief introduction to the concepts of NoCs, and their roots in parallel computing communication infrastructure. Examples of proposed NoC architectures are presented to establish what direction the field is taking, and to establish the types of features one would reasonably expect to find in such systems. Finally, a review of automated NoC design is presented, which analyzes many of the works proposed in this field, and identifies some underlying areas requiring further refining and development.

3.1 Networks on Chip: An Introduction

As already specified in the introduction, the NoC concept has its roots in the field of parallel computing communications. Dedicated communication elements are used to manage the passing of messages such as data transfers of process synchronization flags between various processing elements. In most such systems, each processing element consists of at least one

processor, dedicated memory and communication interface (router), and the messages are addressed to other, similar components.

The core component of all such communication systems is the router, a dedicated component that manages the communication aspects of the system. Many parallel systems, especially when using distributed memory architectures, deploy one router per processing node. Such a node has one dedicated interface to its own processing element, as well as additional interfaces used to communicate with neighboring routers [18]. In contrast to this, other systems allow multiple processing elements to connect to a single router. In these systems, the router does not differentiate between connection to other routers and connections to processing elements. To accommodate such designs, each processing element has its own dedicated Network Interface (NI), which converts messages to a format appropriate for the communication network [18].

Networks further distinguish themselves by the method in which information is transmitted. Sent messages are converted into packets for transmission across the network. Circuit switched networks then reserve a connection between the source and destination nodes, and transmit the message packets once a path is set [18]. On the contrary, packet switched networks send packets out as soon as they are formed, and rely on individual routers to direct them to their destination [18]. In addition, the way each packet is transferred through a router can vary. Store-and-forward networks buffer an entire packet before sending it one, while wormhole networks subdivide packets into smaller flow control units (flits) and deal individually with these [18].

Finally, two other important parameters of a network are the topology of the network and the routing strategy used by individual routers. These two aspects are presented together because they are strongly dependant on each other. A network with a regular topology, where the position of each node is exactly specified by its network ID , can use dedicated hardware to determine the route of a component (see dimension ordered routing in two-dimensional

meshes). On the other hand, irregular networks often make use of look-up tables to determine how to forward a packet or a flit. Some networks rely on the source NI or router to determine the path sequence and append it to the packets or flits being transmitted (know as source routing), while other networks store look-up tables locally within each router, in the form of memories. Finally, some networks implement adaptive routing strategies which monitor the status of the network, and make routing decisions based on this as well as the destination of a packet.

As the above discussion shows, many aspects characterize a given network design. Although NoC systems are intra-chip infrastructures rather than inter-chip, they nonetheless exhibit some of the characteristics listed above [5]. The following reviews will briefly describe some existing NoC designs, paying particular attention to their topology, switching and routing method and intended structure. This will identify the potential design aspects that have to be tackled by the design flows which will be reviewed in Section 3.

3.1.1 The XPipes NoC

The XPipes NoC system is presented by Bertozi and Benini as a collection of macros which can be used to specify the communication infrastructure of a given SoC system [19]. The XPipes NoC is a best-effort (BE) network, and uses wormhole switching to transfer data across the system. Because of this, each router output has to buffer only a few flits, rather than the entire packet (the XPipes system uses output buffering). To further reduce the size of individual routers, the network uses source routing; each packet has routing information appended to its header flit. The routing information consists simply of a numerical indicator of which output port the flit should be forwarded to (referred to as street sign routing).

To allow communication between various cores, the XPipes libraries contain Network Interfaces and Link components in addition to switches. The Network Interface is responsible

for converting the transactions issued by the system cores into flits to be transferred by the network. Most notably, the NIs contain look-up tables with the routes to various destinations. The links themselves are designed to allow the network to be insensitive to wire latency. This is accomplished by designing the links as a series of interconnected registers rather than simple wires. This pipelining of the links allows fast clocks to be used in the network, at the cost of increased cycle delay. The proposed network implements error control at the link level, and allows the use of various error correcting or detecting codes. The error detection in this case is used to determine if transmission errors occurred during the transfer process, and if a flit has to be re-transmitted. A go-back-N retransmission policy is used in the case where an error is detected, to allow for the fact that multiple flits may have been sent by the time a transmission was detected.

Because the network components are specified as macros, they can be instantiated in multiple ways, to suit a given application. The network is built to accommodate both regular and irregular topologies, which allows a system designer more freedom. This is primarily accomplished through each core's NI which stores the routes of transaction targets, and the fact that the routing is look-up table based rather than being hard-wired in the switches.

3.1.2 The Aethereal NoC

Similar to XPipes, the AEthereal NoC is built to accommodate various topologies and configurations [20]. However, unlike the XPipes network, the AEthereal system combines both best effort and Guaranteed Service (GS) support. The BE service assumes the form of a wormhole-switched, input-buffered network using source routing, similar to that used by XPipes. Guaranteed Service operation is provided via time-division multiplexing of virtual circuits over the same physical link. This guarantees that no contention can occur in the system, although bandwidth can be wasted in this way, if a given time-slot is reserved but no

data is available to forward. To alleviate this problem, each router combines capabilities for both GS and BE operation. Best effort packets are forwarded in time-slots that are unused by any of the GS connections.

The Aethereal NoC provides a programming model for programming the GS connections into the slot-tables of each router. Such programming occurs when the nature of the application changes, leading to new communication requirements that must be met. At this time, different time-slot reservations would be made, based on the new communication requirements between components. Two models are proposed by Goossens et al.: a distributed programming model, as well as a centralized model. In the distributed model, GS circuits are established using set-up and tear-down packets to reserve specific time-slots in each switch in the path. This way, the cores themselves could control the existence of various circuits. The centralized programming model makes use of a central control unit to program the slot tables of all routers in the network; such a solution would primarily be used in small systems.

3.1.3 Regular Topology NoCs

While the above designs allow complete freedom in the selection of a network topology, other works have been presented which target a specific topology for the network structure. The CLICHE system, proposed by Kumar et al. targets two-dimensional mesh architectures [21]. The proposed network is aimed primarily at distributed memory systems, where each component consists of a processing element, local memory and a network interface to handle packetization of the sent messages.

The SoCBUS interconnect also presents a mesh based topology, but the system employs a hybrid of circuit and packet switching for its operation [22]. The system uses control packets to reserve system resources and establish a connection; however, once the connection exists, circuit switching is used, which reduces the latency inherent in packet switched systems.

As with most other NoCs, a wrapper component is used to interface between the cores in the system and the network itself. The wrapper also handles the circuit set-up phase of the transaction process. Finally, routing is only required in the set-up phase, and simple shortest-path routing is used.

The SPIN network on chip makes use of fat-tree topologies, targeting 4-child trees [23]. The network is wormhole-switched, and makes use of 32 bit flits for communication. Messages have no pre-set size, and can be any arbitrary length, terminated by a tail flit. Hardware routing is used, with each router capable of implementing adaptive routing to alleviate congestion in the system. The system makes use of wrapper components (acting as network interfaces) which conform to the Virtual Component Interface [23] protocol to allow cores to be connected to the network.

Finally, the Octagon NoC presented by Karim et al. proposes the use of a specialized network based on ring networks with bisecting links [24]. The Octagon is based on 8 nodes arranged in a ring, but with the addition of diagonal links connecting each diametrically opposite pair of nodes. This approach ensures that any pair of nodes in such a structure can reach each other in a minimum of two hops. In the case of more than 8 nodes being present, the system can be expanded by connecting multiple octagons topologies together. The network supports either best effort, store and forward packet-switching, or circuit switching operation. Routing in the packet switched mode of operation is based on network addresses, and is hardwired into each router. Circuit switched operation relies on a central arbiter to establish connections between components, and permits multiple connections to exist provided they do not overlap.

3.1.4 The Asynchronous NoC: The MANGO Clock-less Network

As a response to the ever increasing problem of clock distribution in deep sub-micron environments, the MANGO NoC is presented as an example of asynchronous design [25]. The MANGO NoC employs clock-less operation, and relies on the Network Adapter component to act as a boundary between the synchronous region where the core resides and the asynchronous network. The Network Adapter conforms to the OCP protocol [13], and can support any OCP compliant core. This approach allows various components in the system to have different clock domains, and alleviates the problems of clock distribution networks.

The MANGO router supports both GS and BE operation, similar to the AEthereal design. The best effort service is provided by means of wormhole switching, and source routing is used. The guaranteed service is based on virtual circuits between source and destination. The virtual connections are established through the use of a centralized System Programming Unit, which uses the best effort router components to program the circuits in the system. Both the BE and GS operation modes employ virtual channels, allowing them to multiplex connections onto one physical channel.

3.1.5 Commercial NoC Solutions: The Arteris Danube NoC Library

Finally, an example of a commercial NoC solution is presented to complete the list of potential Noc systems. The system in question is the Danube component library offered by Arteris Inc [26]. The Danube NoC library incorporates a number of parameterizable network components, divided into two large categories: Network Interface Units (NIUs) and Packet Transport Units (PTUs). As with all other networks, the interface units convert various socket formats to a network-specific format. The Danube NIU can interface AHB [12], AXI [14] and OCP [13] components to the underlying network.

The Library also contains network transport units of various types, which transmit information through the network. These transport units include switches which are used to multiplex packets over physical links, various types of buffers for congestion alleviation, as well as clock conversion units. These components allow a wide range of variation in generated networks. It is interesting to note that this proposed system incorporates request and response networks for the connected components, both constructed from the available NTUs.

3.2 NoC Design Research

The previous section gives an overview of some of the design aspects that have to be considered when an NoC is selected for a given application. This section examines various proposed methods for automatically, or semi-automatically generating NoC systems for given applications. Some of the research works analyzed were developed by the same research group, and for this reason are presented here together.

3.2.1 XPipes-Related Design Methods

A series of design methodologies were developed in conjunction with the XPipes NoC. The SUNMAP system, presented by Murali and DeMicheli has the aim of mapping a given application onto various regular topologies [27]. The application is represented by its core graph [28], a data structure showing all the physical cores that compose an application and their communication requirement, and it is mapped onto a topology graph [28]. Topologies are analyzed based on area, bandwidth, power consumption and average delay, and the best topology is selected. Similarly, various routing strategies, including shortest path and split path routing are analyzed to determine which yield the best throughput. Once a topology is selected, the XPipes Compiler [29] is used to instantiate SystemC models of the given

network.

The SUNMAP method supports various topologies, including mesh, torus, clos and butterfly topologies; as well, the topology library can be expanded to support other topologies. The mapping process goes through three phases for any given topology. In the first phase, a greedy algorithm is used to associate core graph vertices with topology graph vertices. The core with the largest communication requirement is placed at the node with the most available links, and subsequent cores are placed according to their communication requirements with the cores already selected. In the second phase, routes are generated for the various communication requirements in the system. During this phase the bandwidth requirements in the system are verified, by ensuring that the bandwidth of mapped communications on a given link does not exceed that link's maximum capacity. As well, power and area estimates are obtained using power models of the NoC components, and a built-in floor-planner. The power models are based on the architecture of NoC components, and is obtained using the ORION power modelling tool [27]. The final phase attempts to improve the results of the first phase by swapping pairs of cores in the topology graph, and evaluating the new topology as described in phase two.

To handle irregular topology generation, Murali et al. present a method for generating irregular topology networks based on floor-plan information of the application [30]. The proposed method has two primary design objectives, minimizing power, hop count or a combination of the two. As well, it considers constraints such as wire-length, hop count, power consumption and area. Once again, the core graph is used, although the edges of the graph now consist of the information flow multiplied by their criticality. The proposed method generates a topology graph for a given application, as well as setting the link width and frequency of operation of the generated network. The algorithm itself iteratively selects discreet frequency and link widths, and proceeds to build topologies based on the selected parameters. Topologies vary between one which has all cores connected to a single switch,

and one where all cores are connected to separate switches. Cores are placed relative to each other based on their bandwidth requirements, as specified by the core graph. All topologies in the range are then rated according to power consumption, area, wire-length, and hop delay (as with the SUNMAP method, device models are used to obtain area and power estimates). Bandwidth constraints are treated the same as in the SUNMAP system; however, the link capacity here is defined as: $frequency\_of\_operation \times link\_width$.

Although both proposed methods target aspects of embedded system design, by analyzing power consumption and area of potential NoC designs, neither dedicates enough attention to the information throughput requirements of on-chip systems. Both methods propose to meet a core's throughput by ensuring that links on a communication path have an equal or larger capacity. However, this analysis method does not take into account the transaction-based communication methods of on-chip components, or the overhead present in converting data for transmission through a network. The irregular topology methodology determines link bandwidth based on link width and frequency, as described above; such an approach ignores the delay present in the switch due to arbitration and forwarding, and is valid only in situations with no contention. Neither method accounts for the fact that during a transaction, as much as half of the information sent is not actual data but rather additional information (address and side-band information [13]), which means that some of the available bandwidth will be dedicated to information other than the core data in any transmission. Both methods can attempt to minimize hop delay, but neither method sets hard constraints on the actual time delay of a transaction. Finally, neither method explicitly considers the effect of network interfaces, either on power, area or delay during topology generation.

3.2.2 Tile-Oriented Design Methods

Hu and Marculescu present a method of mapping applications to regular network topologies based on their power consumption in [31]. The proposed method targets tile-based systems, where each tile consists of a processing element and a router used for communication. The authors make use of an energy-per-bit power model to determine the energy costs of the various communication requirements, and propose a mapping algorithm to the underlying topology which minimizes the over-all power in the system, while meeting the communication requirements of the on-chip cores. The bandwidth constraint is enforced by ensuring that the volume of communications mapped to a given link does not exceed the link's maximum capability. The potential solution space of core to tile mappings is represented as a tree structure, and a branch and bound algorithm is used to explore the solution space intelligently.

The authors further expand their energy-aware mapping process in [32] by incorporating scheduling into the mapping process, both of the tasks of a given application as well as the communication of the application. Once again, a tile-based platform is targeted, consisting of a processing element and a network router to manage communication. In addition, the authors assume that support exists for the execution of multiple tasks per processing element, and that communications can be initiated at specific times; these assumptions are based on the existence of an operating system at each processing element. The proposed scheduling method assigns tasks and communication slots to specific processing elements based on the variance of a tasks execution time and power consumption for various elements. Once all tasks are assigned, a repair process attempts to move tasks in the system so that the number of missed deadlines is minimized.

Ogras and Marculescu present a method for generating hybrid network topologies, neither totally irregular nor totally regular, in [33]. The method adjusts 2D mesh topologies by

introducing long-range links into the system, in order to generate "small world" effects in the system (a reduction in overall latency especially amongst far-off components). The algorithm implementation takes as input a description of the application in the form of communicating cores mapped to a mesh topology. Based on the frequency of communication between components, switches in the system are selected as recipients of an additional long-range link. The aim of the algorithm is to maximize the critical network load, defined as the load at which the network enters a congested state. To determine this value, the algorithm uses an analytical approximation method based on the communication frequency of cores and the latency of contention-free packets. As well, because the addition of links changes the topology of the system, the method also generates new routes for various packets so that the system remains deadlock-free. The new route is based on x-y mesh routing and makes use of the available long-range links when they do not generate cyclical dependencies in the network.

Finally, Hu et al. propose a technique for sizing local buffers in on-chip routers with the aim of improving performance under a given area budget [34]. The proposed approach is, once again, aimed at tile-based system, assuming that all nodes consist of a processing element and a dedicated router; as well, the work assumes that some local storage exists in each processing node, in the form of a local memory, which is used to store packets before they enter the network. A packed-switched system is assumed, rather than wormhole-switched; this allows each packet to be treated as an atomic, independent entity during analysis. The method works by starting with each channel having buffering space for one packet, then proceeds to identify congested channels (which become potential bottle-necks) and increases the available buffer space on these channels, until all available buffering space has been used. To identify which buffers become bottle-necks, the entire system is modeled as a queue-based model. Queue Theory is then used to determine those input buffers which become full, based on the injection rate of packets specified by the application. The proposed method is found

to yield the best results when non-uniform access patterns are present in an application.

All three methods examined above target very specific tile-based architectures, similar to those found in high-performance parallel computing. All three works seem to assume a distributed memory architecture, although only the last work examined explicitly states this to be the case. These assumptions limit the use of the proposed methods for on-chip applications, since such architectures are rarely implemented due to cost and performance constraints. In addition, none of the three works take into consideration the heterogenous nature of on-chip components, except in the form of varying packet injection rates and target destinations. This is particularly problematic in the case of [32], where the authors assume all nodes in the system contain some form of operating system. Such an assumption is acceptable for general processors or DSPs, but not for dedicated hardware units, DMA units, I/O controllers or memories. The different behaviors of possible embedded components is not considered, and a message-passing application model seems to be assumed. Similar to other analyzed works, the methods do not take into consideration the specific behavior on on-chip systems, in particular the use of transaction-based communications. Finally, the power models presented in [31, 32] are based on a power-per-bit analysis method. However, the authors do not explicitly take into consideration the data overhead due to transactions and the network itself.

3.2.3 Irregular Topologies Based on Optimization Methods

Sirnivasan et al. propose to solve the problem of NoC generation by treating it as a pure optimization problem, and propose a design methodology based on Integer Linear Programming [35]. The authors use a Communication Trace Graph to characterize their application prior to the generation of a network on chip [36]; this data structure is similar to the core graph mentioned in [27]. The proposed method has the primary aim of minimizing power

consumption, area and hop count between on-chip cores. These three parameters are incorporated into an objective function to be minimized. As with any optimization problem, certain constraints are placed on the solution space, as dictated by the problem at hand. In this case, the constraints consist of the bandwidth capabilities of input and output ports, the total area of a proposed solution and the aspect ratio of a given solution. The NoC generation engine incorporates power and area models of switches and links, and makes use of a floor-planner to generate early area and power consumption information.

The proposed method works by arranging cores in a layout in such a way as to minimized area, and places potential network switches on the boundaries of various components; each component is then uniquely mapped to a specific router. From this first step, redundant routers are eliminated. The remaining routers are connected amongst each other and to IP cores such that the objective function is minimized and the constraints are met. Routes are generated during the connection process, and the bandwidth constraints are verified by ensuring that the sum of required bandwidths present on at a given port is less than or equal to its bandwidth capability (a similar process was used in [27, 30]). Deadlock is avoided by the introduction of Virtual Channels at a later stage in the design process. By using the floor-planner, the proposed method can determine area estimates, as well as the length of various links, and therefore, the power consumed in said links.

The authors also present an updated topology generation method which uses approximation methods to generate topologies [37]. As with the original method, the process consists of two steps: a mapping stage and a connection stage. During the mapping stage routers are placed at the four corners of a given core, and the core is mapped to one of the routers on it's boundary. The aim of this stage is to minimize power consumption in routers, and the connection between two routers is abstracted as a point-to-point link. The authors treat the problem as a min-cut max-flow problem, and solve it by sub-division (the x-coordinate location of routers is found first, then the y-location). The connection stage, during which

the topology is generated, is solved using Integer Linear Programming methods with integer relaxation, which allows the authors to generate results which use, at most, twice as many routers as the optimal solution. The authors show that the proposed method generates close-to-optimal solutions in far less time than an exhaustive method [35].

Finally, Srinivasan and Chatha propose a method for generating topologies for guaranteed throughput NoC Architectures referred to as SAGA [38]. Unlike their previous work, this method makes use of a genetic algorithm to generate topologies, with each solution being represented by a three-level hierarchy. The highest level in the hierarchy stores the number of routers in a solution, the second level stores potential mapping of router ports to cores, and the third level stores various communication traces for the cores in the system. In addition, since the proposed method targets guaranteed throughput networks (such as those seen in [20, 25]), a schedule of packets at each port in the system is generated; the Earliest Deadline First method is used to create these schedules. Each communication requirement between components is characterized by the number of flits sent, the period requirement in clock cycles and the deadline in clock cycles. The NoC topology generator attempts to meet these requirements through scheduling and topology selection.

Similar to [27, 30], the work presented here targets area and power consumption. However, once again, the throughput requirement is reduced to a bandwidth inequality which ignores the overhead present in transaction-based communication and packetization prior to transmission over the network, as well as the time overhead of multi-hop communication. As well, the role of network interfaces is ignored (in terms of delay, area or power consumption); the process of mapping cores to specific switches in [37] implies the assumption that switches have special, dedicated ports for connections to cores, but this is never explicitly stated. In addition, the authors make certain assumptions regarding regular topology networks, such as the fact that mesh topologies must conform to grid placement during layout, which are not necessarily true; a mesh topology is primarily characterized by its connectivity, and an

example of a mesh floor-plan that does not follow grid alignment is shown in [19]. Finally, in the case of SAGA [38], communications between components are characterized by the number of flits, their period and their deadline, in a manner similar to the description of tasks requirements in a data flow graph. However, this approach does not address the fact that the flits are associated with transactions; the deadline and period of transactions is not considered in the presented work. As well, the authors do not discuss the effect that scheduling has on the over-all transaction throughput of the connected cores.

3.2.4 Additional Research Projects

ASNoC

Xu et al. propose a design method for application-specific NoCs based on hierarchical networks of communicating components [39]. As opposed to design flows presented thus far, this system takes as input a behavioral specification of the target application (presented in some high-level language such as SystemC or C), and an architectural description of the computation nodes available in the system. The design flow then distributes the application behavior onto the available computational units, and generates a distributed memory model, which will then be converted to a distributed shared memory model. The design process then goes through five stages. In the first, communication traces of the application are obtained, either through simulation or from statistical models. Secondly, the NoC topology is generated for the given communication pattern. Third, area estimates are obtained based on library models of the NoC components (switches, network interfaces and links). In the fourth step, a network simulator (OPNET) is used to simulate the proposed system and obtain performance results of the given application (both network performance and application performance). In the fifth and final step, power estimates are obtained based on a library of power models of the components and the activity measurements obtained from the

simulator.

The topology generation process (stage two) takes as input the communication information obtained from the first stage, in the form of a communication graph; the vertices represent computation nodes, while the weighted edges represent communications of a given volume between nodes (similar to a Core Graph [27]). The topology is generated using a recursive process, which subdivides the graph into smaller graphs using k-way partitioning; each sub-graph has an associated cost, based on the volume of local communication (where both source and sink are in the same sub-graph) as well as inter-network communication (the source and sink are in separate sub-graphs); the two types of communications are weighted by their cost in cycles. The set of sub-graphs with the smallest associated cost is selected as the final topology. At the same time, the individual memories used during the communication trace extraction are merged into a distributed shared memory.

Unlike previous works, the process proposed here does not use the bandwidth inequality method discussed above in the design process. A topology is judged based on relative improvement compared to others, but no hard requirement is enforced during the generation process. Having said that, the proposed method does attempt to minimize over-all latency in the system by grouping components with large communication in the same sub-network. In addition, the proposed design method models a given application in terms of computation units with distributed memories; even after the memory space is merged, the architecture is still distributed. As such, the design flow does not take into consideration low-level interactions, such as a processor updating its cache from a memory unit somewhere in the system. Finally, a macro-network simulator is used to estimate the performance of a given solution; the simulator is augmented to permit it to more accurately handle on-chip communication aspects. However, no mention is made of wether the simulator models communication as being transaction-based (a low-level view of the communication process, but appropriate for on-chip systems), or message based (a higher-level view of communication

often encountered in parallel computing systems, which are generally based on distributed memory architectures).

Minimum-Contention NoCs

Ho and Pinkston propose a design method similar to that presented by Xu et al., in that a given application is first analyzed to determine its access patterns, and a network is then generated around these patterns [40]. The design flow assumes that all resources in the system are processors communicating with each other, and that each processor is executing one process only. Each processor has its own network interface, but multiple such interfaces can connect to each switch. The authors present models for time and path conflicts of messages present in the system. Time conflicts occur when messages content for the same resource at the same point in time, while path conflicts occur when a message may use the same resource as other messages. By ensuring that any potential conflict is either temporal or path-based due to routing, but not both, contention-free communication can be achieved. By keeping the number of messages that fall in both sets to a minimum, minimum contention in the system is achieved, with the absolute minimum being no contention.

The proposed design method begins with the analysis of a given application to determine the sets of contention messages and the contention periods of such messages. The system then proceeds to generate the network topology which includes the connection of processors to switches and the number of links between various switches. The topology generation process is iterative in nature, and begins by connecting all components to a central switch (a cross-bar). The switch is then partitioned sequentially until design constraints (such as maximum switch size) are met. At each stage in the process, the minimum number of links between switches is found using a graph-coloring process combined with simulated annealing for the placement of processors in the system. At all times, the aim of the system is to arrange matters so that no contention occurs between messages. The proposed method

was compared to a fully-connected cross-bar, as well as mesh and torus topologies, using a collection of scientific applications, and was found to perform better than almost all, thanks to the elimination of most contention in the system.

The primary shortcoming of the proposed method is that it is applicable only to a small set of systems consisting only of processing elements (such as chip multi-processors). As well, message passing is assumed to be the fundamental method of communication, even when shared memories are used; this eliminates some of the complexity present when transaction-based communication takes place, and ignores the additional over-head of such methods. Finally, the method is tested using scientific applications, commonly encountered in the field of parallel computing; however, such applications generally do not have hard requirements placed on their completion time (and therefore on the performance of the network). In contrast, many embedded on-chip applications, in particular in the field of multimedia and networking, have strict throughput requirements of one sort or another (such as the number of frames decoded and displayed per second, or the number of packets processed per second).

Binary Tree Hybrid NoC Designs

Jeang et al. present both a potential NoC platform as well as a design approach for said platform in [41]. The proposed NoC architecture is a hybrid design, incorporating both local interconnects in the form of buses, cross-bars or multi-layer interconnects, as well as NoC switches for communication amongst local networks. A binary tree is used as the NoC topology, with each internal node being a switch and each leaf node being a local network. The NoC implements a wormhole-switched architecture, using asynchronous messaging between routers to transmit information. The switches themselves use a priority-based arbitration method in the case of contention over an output channel.

The design method for the proposed interconnect starts by grouping cores together into local networks; cores are grouped into such a network if their communications can be ar-

ranged to be temporally disjoint. Once this initial grouping is performed, a graph can be constructed where each node represents a bus system and each edge represents a communication requirement. The edge weights represent the communication ratio (similar to volume requirements in a core graph) between two local networks, and are specified by the user. Based on this graph structure, switches are generated and connected appropriately by grouping the local networks with the largest communication ratios together.

The proposed method is novel in its attempts to leverage communication locality, but does not enforce any hard requirements on achievable throughput or latency; as such, it may not be well suited for applications with hard performance requirements. In addition, while the authors present detailed information on the implementation of the switch architecture, they do not describe the interfacing mechanism that converts bus data (or cross-bar) to the format required by the network. More importantly, they do not describe the transition process between the synchronous domain of the local network and the asynchronous domain of the switches.

Other Approaches

Ascia et al. present a mapping procedure based on genetic algorithms, which can be used to map the cores of an application onto a mesh architecture [42]. The proposed method distinguishes itself from others in that the obtained results contain all mapping solutions present in the pareto set [42] of a given application's solution space. In this way, the process attempts to present the solutions that best balance various design goals (in the current case power consumption and latency). The authors use a simulation engine to determine the performance and power metrics of various mapping solutions. The genetic operations of cross-over and mutation are altered for the given process, so that both operations attempt to reduce latency and alleviate congestion in the system, while maintaining the random aspect of these operations. The proposed methodology is compared to methods based on those

proposed in [28, 31], but adjusted to find pareto-optimal solutions.

Zhou et al. also propose a procedure for mapping application cores onto mesh topologies using a genetic algorithm [43]. However, unlike the previous method, in this case only average latency is considered as a fitness function in the system. The authors use queue theory to analytically compute the average delay of a given solution, based on the packet injection rate at various nodes. The solutions themselves are represented as strings, and employ a relational form, determining the relationship of cores in the topology.

Papadopoulos et al. present a method for the generation of optimum NoC systems which operates on both the application being deployed and the NoC structure itself [44]. The method relies on the optimization of dynamic data types in the application, in an attempt to tailor parameters such as memory accesses in the system. Once the optimization process completes a set of pareto-optimal points are used to simulate the application in various NoC implementations. The authors make use of an updated form of the NOSTRUM NoC simulator [45], and consider various parameters such as topology, routing method and packet length. The application is distributed in the form of threads onto the NoC topology, where each node in the topology is a processing element with an associated switch.

As with previous methods examined here, however, the work proposed does not take into consideration the specific communication method used by on-chip systems. In addition, the method does not explicitly set any hard constraint on the performance aspects of a generated mapping; the generated solutions attempt to balance performance with power consumption, but in so doing may not meet the communication requirements of the targeted application.

3.3 Conclusion

The past chapter has introduced the NoC concept and some examples of its possible implementation. These examples include regular and irregular topologies, best effort and guaran-

teed bandwidth methods, and various forms of routing procedures. The chapter also includes a review of automated design methods proposed for the field of NoC design. These methods generally target topology generation or mapping of an application to a regular topology. Many incorporate detailed models of power consumption and area, in an attempt to optimize these parameters. However, an underlying feature of most design methods having been proposed is the use of high-level communication models, as described in Section 2.3.2. Such models are adequate in the field of parallel computing and wide-area-networks, but are not always appropriate in on-chip situations.

Chapter 4

Transaction-Oriented NoC Design

This chapter presents the actual design and analysis method being proposed, and its implementation. The chapter is divided into four sections, dealing with the actual objective of the method being proposed, the over-all structure of the implemented program, the topology generation process and the topology analysis process. The reason for so many sections is that the system incorporates the topology analysis method into the topology generation method, depending on the topology generation algorithm being used. This permits iterative improvement of topologies based on predicted performance, but also makes the program structure more complex.

4.1 Method Objectives

The main aim of the design and analysis methods being proposed is to generate application-specific network topologies for NoC interconnects, and to determine at what operating frequency the interconnect must run to actually meet the communication requirements of the application. Network analysis is included because no interconnect design can be complete without it. A shared bus could be used in many cases to meet communication require-

ments if it could operate at very high speeds compared to the cores it connects. Because of this, the method described here attempts to generate a topology which will minimize the recommended operating frequency of the interconnect. This generally leads to a reduction in the power consumed by the circuit, as switching power is proportional to the switching frequency. By attempting to reduce the recommended frequency, the method will also help conform to physical limitations present in current on-chip manufacturing technologies.

In the current case, the term *application-specific* refers to the generated topology, rather than other parameters that can be altered for specific applications, such as the link width of the network, flow control or buffering methods. For a more detailed list of such parameters one can consult a number of other works [4, 5]. The proposed algorithm generates two types of irregular topologies, both of which have the aim of minimizing latency between components and utilizing a minimum number of network resources. The remainder of this sections covers two important preliminary aspects which must be addressed before describing the actual design and analysis method. Section 4.1.1 describes the supported network model; this is the underlying structure that the topology generator assumes of the network. Therefore, the proposed method should only be used for these types of networks. Secondly, Section 4.1.2 formally defines the input and output data structures of the method being implemented.

4.1.1 Supported Network Type

Given that many NoC systems have been proposed [4], it is important to establish what NoC type the generator targets; the behavioral characteristics and permissible structures of the topology will be dictated by the NoC type. The topology generator being proposed targets best effort, packet switched NoC systems, using wormhole switching [4, 5]. Static, look-up table based routing is assumed, with route tables embedded in the header flits of each packet (source routing) [19, 20]. No virtual channel support is assumed, and deadlock-free

operation is ensured through the topology and route selection. These features are selected as they reduce the complexity of the NoC component parts, as well as the area requirement of components; in particular, wormhole switching and source-based routing ensure that the switches in the system do not require large buffers or look-up tables. Selecting designs that minimize complexity and area favors on-chip implementations [4, 19, 20].

On-chip components access the network using Network Interfaces (NI), which perform transaction conversion into a format supported by the network (packetization). Separate NI types exists for master and slave interfaces, and it is assumed that a component with both master and slave capabilities will make use of two network interfaces. This makes the network interfaces less complex, and can increase performance. Point-to-point connections can be established between the NIs and switches, as well as two NIs of opposite type (master and slave). This allows simple point-to-point links to be created even if the core interface does not natively support this feature. A transaction is converted into a packet composed of a fixed number of flits; currently, in the case of burst transfers, it is assumed that each burst beat is converted to a separate packet.

The switches have the sole role of transferring data through the network. Each switch is composed of a fixed number of ports, where each port consists of an input port and corresponding output port. The point-to-point links between NoC components (switch-to-switch as well as switch-to-NI) are composed of two unidirectional channels, which together make up a link. Stop-Go flow control [5] is assumed on the links, as it is the simplest form of flow control. Switches are assumed to be capable of performing arbitration and forwarding data independently for every output port, meaning that each output port has its own dedicated arbitration and forwarding unit. Round-robin arbitration is currently supported by the system performance analyzer. The arbitration method impacts the perceived performance of each core in the system. It is assumed that the implemented network will have one global clock, although the cores connected to it need not share the same clock [14]. This specifica-

tion is important, as the analysis method will assume that only one clock is used throughput all network devices, allowing for their synchronization.

4.1.2 Method Input and Output

Application-specific NoCs must be tailored to the characteristics of a particular application, meaning that the application must be characterized in some way for the topology generator. At the same time, not all aspects of an application are as important from the communication infrastructure point of view; only the communication characteristics of the application matter. There are multiple ways of describing such communication characteristics, starting with high level descriptions of a given application (such as a task graph). However, the generated NoC must meet the communication characteristics of the actual components in the system, as opposed to behavioral requirements of the algorithm. For this reason, the Core Graph was introduced and used in NoC topology generation [27, 28]. The core graph is a representation of all communicating components present in a given application, including processors, dedicated hardware, memories, I/O controllers and others. The core graph is formally defined as a graph structure C where:

- Each vertex $v_i \in V$ represents an on-chip component interface (either master or slave). Components with both master and slave interfaces are represented using two vertices.

- Each edge $e_{i,j} \in E$ represents a communication requirement between interfaces i and j (one of the interfaces is a master, while the other is a slave). The weight of each edge, $w(e_{i,j})$ represents the volume of information transferred between i and j, in bits per millisecond. As well, each edge is characterized by the information payload per transaction, information flow (read or write) and burst support.

The core graph represents each core by its communication interface. If a core can both initiate and receive information, then it is assumed that it will have separate master and

slave interfaces. This process is common in most embedded systems, as most standard on-chip interface protocols distinguish between master and slave interfaces [7, 13, 14]. The core graph C of an application is provided to the topology generation and analysis program in the form of an input file, and is then used by the program to generate a topology.

A second input file is used to specify the additional parameters of the network structure being developed. Section 4.1.1 described the basic network parameters that are assumed by the method; however, there are additional parameters that can be specified for such networks:

- Number of flits per transaction.
- Arbitration delay of a switch.
- Forwarding delay of a switch.
- Packetization and de-packetization delay of a NI.
- Maximum possible number of ports per switch.

These are implementation based parameters, and can vary from network to network, or even from implementation to implementation of the same network [29].

Given a description of an application and specific details about the target network, the desired output consists of a topology description and a specific clock frequency of operation that allows the network to meet the application requirements. The network topology is represented (both inside the program and as output) using a graph structure referred to as a Topology Graph T, and formally defined as:

- Each vertex $r_k \in R$ represents a resource item; the vertices are sub-divided into two categories as follows:

- The set of network interfaces $i_k \in I$ represent the network interfaces present in the network. There is a one-to-one mapping between every interface in the core graph and the set I.
- The set of switches $s_k \in S$ used to connect the various network interfaces.

- Each edge $l_{k1,k2} \in L$ represents a pair of unidirectional channels which form bidirectional links between the network resources.

The topology graph distinguishes between switches and network interfaces. This allows a topology to be represented more accurately, both in terms of its structure as well as its performance. In dedicated, low-latency topologies the packetization and de-packetization processes of network interfaces can account for a considerable fraction of the over-all delay, and must therefore be taken into consideration.

Associated with every topology is a route record, which describes how data will move from one component to another through the network. The record consists of a series of entries for every master interface in the system, describing routes for all possible transaction targets of the interface in question. Each entry consists of an enumeration of vertices from the topology graph, starting with the switch directly connected to the current NI (corresponding to master interface in the system) and ending with the NI of the destination core (or interface, if the destination core has both master and slave interfaces).

4.2 General Program Structure

The proposed NoC design method creates a custom topology for a given application based on performance parameters for that application. The analysis method being proposed is incorporated into the topology generation process to provide feedback. This makes the program structure more complex than a simple two-stage implementation where the first

stage generates the topology and the second analyzes it. In addition, two topology generation algorithms are proposed. Both algorithms are heuristic in nature, and either one, in isolation, may not always be effective for all encountered applications. For this reason, both methods are available, and the user can select which type of topology to generate at the start of the program. The differences in generated topologies, and the criteria for algorithm selection will be discussed in more detail in Section 4.3.3.

Figure 4.1 shows a high-level flow diagram of the design and analysis process, which starts with the user selecting the type of topology to be generated. The two topology types will be described in further detail later. Once the topology type is selected the program proceeds to its main processing stage, where the topologies are designed and analyzed. The design process is based around two component steps: generation of a topology graph and analysis of the graph. Depending on the topology type, this process may be repeated iteratively, meaning that the topology graph can be generated based on an earlier version of that graph. The same analysis method is used for both topology types, which is why it is discussed separately. In fact, the analysis engine can be used in isolation to simply analyze any proposed system, by supplying it with a core graph, associated topology graph and route table. Once a topology is generated and analyzed, the resulting topology graph, operating frequency and route tables are output to the user.

Figure 4.2 shows the structural implementation of the proposed method. The main component of the program is the generator object, which incorporates core and topology graphs which it operates on. The program also incorporates a set of requirement data structures, which are constructed based on the core graph and used during topology generation. The analysis engine is incorporated into the generator object as part of its method set, and operates on the topology graph and requirement data structures. As long as topology and route information exists, along with a set of requirement structures (derived from the core graph), the analysis engine can bypass the topology generation algorithms and be used by itself.

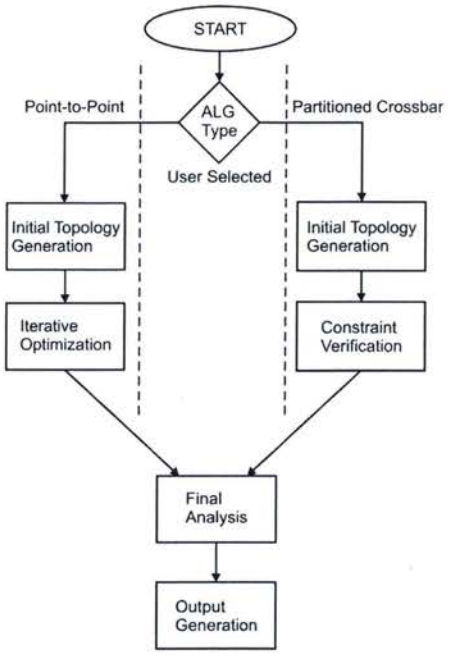

Figure 4.1: Main Program Flow

This allows the engine to be used either as a system analyzer or as a topology generator and analyzer.

4.3 Topology Generation

The topology generation algorithms are designed to meet two goals: to create a physical connection for all required communication paths, and to reduce the required frequency of operation of the interconnect. The first requirement is simple to meet, while the second requires some analysis, based on the information presented in Section 2.3.2. Equation 2.2 specifies a latency requirement for all communications in a system, and Equation 2.1 specifies its component parts. If a communication path has a high required throughput, it will have a low associated latency requirement. The slave processing speed cannot be changed at this point, so the only remaining improvement area is the transport latency of the interconnect

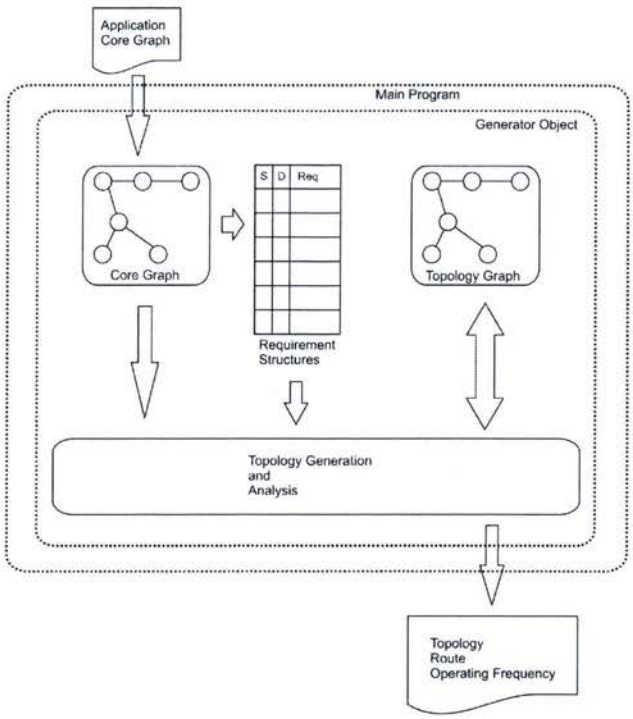

Figure 4.2: Main Program Structure

itself. The passage of a transaction packet through the interconnect has an associated latency in cycles, based on the implementation of the network and its topology. Topology plays a part because each hop in the network will add additional latency to a packet. Based on the architectural design of the network components, each transaction will have an associated clock cycle latency, meaning that the forwarding of information in the network will take a set number of clock cycles to complete. Given a latency in clock cycles, the associated time unit latency is computed as follows:

$$Lat_{time} = Lat_{cyc} \cdot \frac{1}{F_{op}} \tag{4.1}$$

Above, F_{op} represents the interconnect operating frequency, Lat_{cyc} represents the clock cycle latency of a transaction, and Lat_{time} represents the time latency of a transaction, in seconds.

Given that Lat_{time} has a fixed maximum above which communication requirements are no longer met, the aim is to minimize this value. This can be done either by increasing F_{op}, which cannot be done indefinitely (because of physical limitations such as delay and noise, as well as added power consumption), or by reducing Lat_{cyc}.

The ideal situation would be to have minimum hops between all components; this is the fully-connected crossbar. Reducing the number of hops is important because in an on-chip network each hop has an associated processing delay, even in wormhole systems that operate in a pipelined fashion. In such cases we are assured that all communication paths have minimum latency, by introducing only 2 hops between source and destination. However, such solutions can be come prohibitive in terms of area [4, 5]. A second approach is, then, to identify the communicating components with the largest required throughput and the smallest associated latency requirement, and attempt to minimize the Lat_{cyc} they encounter. This allows the operating frequency of the interconnect to be reduced despite the fact that some communication paths experience high latency, because the paths in question do not have large throughput requirements and can operate with such latencies.

The two algorithms presented below attempt to do this in two different ways, by generating and operating on topology graphs. Both algorithms start by creating an initial topology which is then adjusted to meet various requirements. However, as shall be seen, the initial topology and adjustment methods differ for the two algorithms. Two algorithms are proposed to account for variations in the application core graph. While it has been found that in most instances the two generated topologies offer similar performance, there exist cases where one approach is superior to the other.

Both algorithms start by simply providing physical connections for the required communication paths, and then adjust the generated topology to improve selected connections or meet specified requirements. Since both algorithms start with minimal topologies that only provide a communication path they reduce the required number of network compo-

nents being used. While in a parallel computing environment such an approach would not be beneficial, due to the communication characteristics of the components and the nature of the applications, it is convenient for on-chip systems. Because of how communication occurs in transaction-based on-chip systems (see Chapter 2.3.2), aggregated bandwidth is not necessarily useful, as transactions are generally centralized to the slave interfaces in the system. In such cases, having multiple paths to a destination will not really improve matters, as transactions will still interfere with each other at the destination network interface (more on this in Chapter 6).

4.3.1 Algorithm 1: Point-to-Point Oriented Topologies

The main steps of the point-to-point oriented algorithm are shown in Figure 4.3, and consist of an initial topology generation phase followed by iterative steps which aim to improve the system topology. At each iteration, the system is analyzed, and the required operating frequency is derived; thus, the analysis process is invoked multiple times for this algorithm. The algorithm is referred to as *Point-to-Point Oriented* because the initial topology resembles a point-to-point communication infrastructure, with network components being used to implement the underlying signaling and multiplexing of paths.

The initial topology is generated by closely following the structure of the core graph. A three step process is followed, as described below:

1. For every master or slave interface in the core graph, insert one network interface in the topology graph.

2. Add one switch for every vertex v_j in the core graph with more than one incident edge, and connect the switch to the network interface i_j corresponding to v_j.

3. Finally, connect the network interfaces amongst themselves or to various switches so that the required communication paths are created.

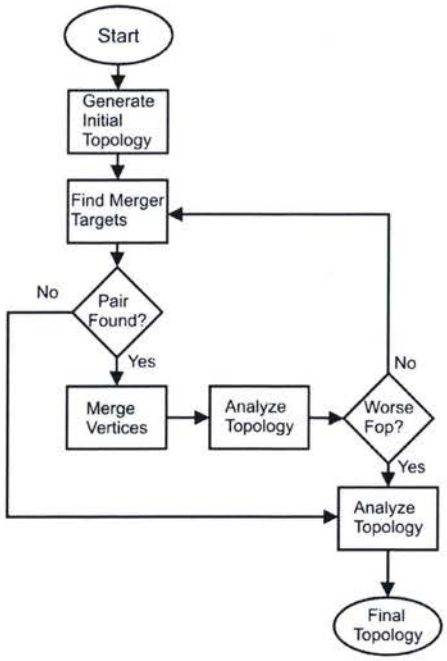

Figure 4.3: Point-to-Point Oriented Algorithm

The above process implements point-to-point links amongst communicating interfaces using NoC components. Switches are used to multiplex amongst multiple incoming links (hence step two). The above process results in a topology consisting of network interfaces, with switches present only where required for multiplexing; sources and destinations are separated at most by three hops in the network. Each switch added in the initial steps has an associated *correspondence* with a master or slave core in the core graph, which is stored in the topology graph vertex object.

Once the initial topology is created, the iterative process attempts to reduce the cycle latency of high throughput paths. This is done by merging switches in the system, so that the hop count can be reduced. The switch pair to be merged is selected using the following algorithm:

1. For every switch vertex in the topology graph compute the total aggregate traffic passing through it, in transactions per millisecond. Omit from this computation vertices

added during a partitioning process.

2. Select the switch with the highest aggregated traffic that corresponds to a master vertex in the core graph. This will be the first merger target.

3. Find the switch that has the largest transaction traffic to the first merger pair. This will be the second merger target.

4. Merge the two target switches.

The above process is shown in Figure 4.4 below, where two switches are merged to reduce delay in the shown paths.

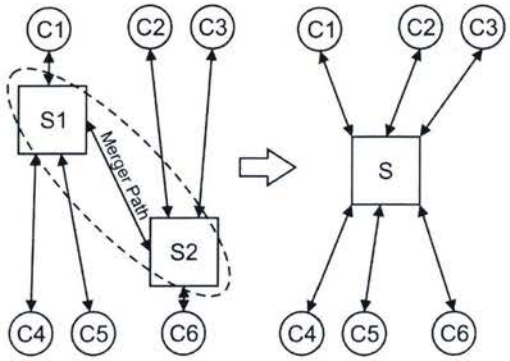

Figure 4.4: Example of a Switch Merger

Once the two switches are merged, a second structural change must be made: if the resulting switch exceeds the port limit, it will have to be partitioned into two smaller switches. A possible way of avoiding this problem would be to merge only switches that are small. However, by merging two switches and then partitioning the resulting switch again, it is possible to generate an improved topology, from a latency point of view, as will be described. During the partitioning process, a new switch vertex is added to the topology graph, connected to the switch that must be partitioned. Then, edges are transferred from the original switch to the new switch until the port count requirement is satisfied. If the original switch

was extremely large, it will have to be partitioned into more than two switches. In this case, more vertices are added to the topology graph, and edges are moved until the port requirement is met. The selection of edges to move is important; moving the wrong edge will invalidate the merger process. In addition, it isn't enough to select the edges with the smallest traffic flowing through them, since this traffic, though small, may originate from a core with large traffic requirements on other links. If a core has a high throughput requirement, it must experience small delays for *all* its transfers. Large delays to one transaction destination, even if such transactions are rare, can lead to that master device not meeting its throughput requirement. To avoid this situation, edges to move during partitioning are selected as follows:

1. For every link i connected to the switch to be partitioned, build a list Lni_i of all network interfaces directly or indirectly connected to it.

2. Find the biggest transaction requirement for every network interface in Lni_i. Let the largest of these values be $Tworst_i$, associated with port i.

3. Select the port with the smallest value $Tworst_i$ and move this link to the new switch.

When the merger and partitioning processes are performed together, the result is to reduce latency of select paths (those with high throughput requirements) at the expense of other paths.

After the merger and partitioning steps are complete, the topology is analyzed, and the new frequency of operation is determined. If the recommended operating frequency has improved (i.e. is lower than previously) or remains the same after the above operations, the changes are made permanent, and the new topology once again undergoes the process. By allowing mergers that neither reduce nor increase the frequency of operation, the algorithm can reduce the number of resources used by a topology. The cycle stops once a topology

change results in an increase in the frequency of operation, or when no more merger targets can be found.

4.3.2 Algorithm 2: Partitioned Crossbar Topologies

The second algorithm is simpler in nature and operation than the first, and is based on the observation that the fully-connected crossbar is an ideal interconnect from the point of view of connection latency and connectivity, on-chip. However, as was stated previously, systems with multiple connected elements can require very large such topologies. The solution adopted by this algorithm is to subdivide such crossbars, thus maintaining the beneficial aspects for some of the connected cores. Given that the internal structure of NoC switches often consists of a crossbar interconnect in addition to buffering and arbitration [4, 5], a type of cross-bar can be implemented using a network switch.

Unlike the previous algorithm, this one is not iterative. It consists of three stages, the generation of an initial topology, the adjustment of said topology and its analysis. The initial topology is generated by creating a network interface for every vertex in the core graph and then connecting all interfaces to a central switch, thus achieving a fully connected cross-bar. To meet the port number requirements, the partitioning procedure described for the *point-to-point algorithm* is used to move certain connections to secondary switches, hence the name *partitioned crossbar*. An example of the partitioning process is shown in Figure 4.5 below. Finally, when the port number requirement is met, the resulting topology is analyzed, and the results are presented to the user.

4.3.3 Topology Comparison

The partitioned crossbar algorithm may be considered less effective than the point-to-point oriented algorithm, given that it does not rely on feed-back from the analysis engine to

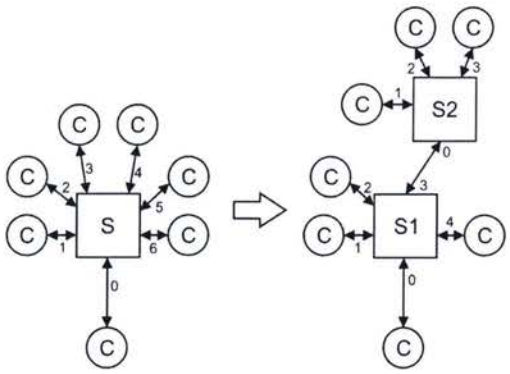

Figure 4.5: Example of the Switch Partitioning Process

iteratively improve the topology. However, since the algorithm starts from the ideal case and works its way backwards it can mitigate this problem, provided that the partitioning algorithm works well. It will be shown that in most instances the topology generated this way yields similar performance when compared to the *point-to-point* algorithm. One of the main differences, apart from general structure, between the two algorithms is that the second will generally use less switches than the first, but each switch will be larger. If more than one switch is in use, all but one of the switches will have the maximum allowable number of ports, since the algorithm only moves links that are in excess of the maximum permitted port count.

The *partitioned-crossbar* algorithm is more suited to applications that exhibit highly centralized communication patterns (a small number of components are transaction destinations). In such situations, additional components are un-necessary, and the topologies generated by this algorithm are ideal. As well, this algorithm is guaranteed to generate two-dimensional topologies (in that links never overlap). In contrast, *point-to-point* topologies are more suited to applications that contain large numbers of components, and distributed communication patterns. applications that contain components with single connections (for example a pipeline with components connected sequentially) will also benefit from *point-to-*

point topologies, because in these cases NIs can be connected directly without the use of switches.

4.3.4 Complexity of the Algorithms

Having described the actual steps of the two topology generation algorithms, it is possible to determine their complexity in relation to the system input. This section analyzes only the steps described above, however; the complexity associated with the topology analysis procedure is difficult to accurately derive, as it is based on a number of separate parameters such as the core graph, topology graph and network requirements. The complexity analysis presented here is, itself, based on multiple parameters. The two primary input parameters are the number of vertices in the core graph, n_v and the number of edges in the core graph, n_e. The third input parameter that must be considered is the maximum permissible port count, n_{port}.

Since the partitioned crossbar algorithm is the simpler of the two, it will be the first to be analyzed. The algorithm is divided into two parts, the initial topology generation and the adjustment process. During the topology generation process, one NI is added and one link is made for every vertex in the core graph, which yields a complexity of $O(n_v)$. The partitioning stage consists of sequentially moving links from the central switch to added switches. To select which link to remove, the partitioning algorithm described above must be used. In the case of a central switch, this means parsing all NIs in the topology graph, and yields an $O(n)$ complexity (this situation is a special case where all NIs are directly connected to the central switch). This partitioning process will be executed $n_v - N_{port}$ times, to account for the amount of links connected to the central switch in excess of the minimum port requirement. Therefore, the final complexity is $O((n_v - n_{port}) \cdot n_v)$.

The Point-to-Point algorithm is more complex in its structure. In addition, it relies on the

analysis engine for feedback. In the following analysis the analysis complexity is neglected, as it would make any complexity estimation difficult, given that the complexity of the analysis method is based both on the core graph as well as the topology graph and the communication requirements of the original application (the actual weights of all edges in C). Taking the same approach as before, the algorithm is divided into an initial topology generation phase and an iterative improvement phase. The complexity of generating the initial topology is $O(2n_v+n_e)$, based on the fact that a set of NIs and switches is created for every core (the worst-case scenario), and a number of links equal to n_e is generated. In the worst case situation, the iterative merging and partitioning process occurs $n_v - 1$ times, as all switches undergo the merger procedure. The merger process itself has $O(3n_v)$ complexity as all switches in the topology are parsed at most three times during the process. Finally, the partitioning process occurs, but this time the worst-case complexity of the process is $O((n_v - n_{port}) \cdot n^2)$, as each connection to the current switch has to be expanded fully to determine the interface with the largest communication requirement. The complete complexity of the iterative process is, then, $O(3n_v \cdot (n_v - 1) \cdot (n_v^2 \cdot (n_v - n_{port})))$.

Table 4.1 below summarizes the complexities of the two algorithms. Because of the heuristic nature of the proposed method, both algorithms should have polynomial time complexity. However, the analysis method changes this to some extent. The partitioned crossbar method does not directly rely on the analysis method and can be said to have polynomial complexity in all situations. The point-to-point algorithm trades this characteristic for improved topology generation.

Table 4.1: Algorithm Complexity

Algorithm	Initial Topology	Iterative Process
Point-to-Point	$O(2n_v + n_e)$	$O(3n_v \cdot (n_v - 1) \cdot (n_v^2 \cdot (n_v - n_{port})))$
Partitioned Crossbar	$O(n_v)$	$O((n_v - n_{port}) \cdot n_v)$

4.3.5 Route Generation

Because of the relative simplicity of the topologies generated by the two algorithms, route generation is currently handled using a shortest path search algorithm through the topology tree. The routes are built based on the links found in the core graph, which specifies what communication paths exist in the system. The search process begins and ends with the source and destination network interfaces of the two cores involved. The search expands out from the source network interface, with all immediate neighbors added to a search list. Each time a vertex in the topology graph is added to the search list, that vertex is flagged, so that it cannot be added multiple times. The process continues to add vertices to the search list until the destination vertex is found. Then a back-trace process is used to build the route itself.

The primary limitation of the above method is that, if the topologies become complex, the search process may not always find the minimum path. In addition, the route generator does not explicitly verify that deadlock is avoided in the system. The only feature which it incorporates to help against this situation is the fact that it does not allow vertices to be added to the search table more than once. Rather, it relies on the underlying structure of the generated networks to avoid deadlock situations (more on this in the next section). For this reason, this form of route generation is tightly coupled to the topology generation algorithm, and would not be appropriate in other situations. This is why, for regular topologies that incorporate the potential for deadlock, more complex generation methods are used [27].

4.3.6 Deadlock-Free Characteristic

As with any network implementation, NoC systems can suffer from deadlock, livelock and starvation. The proposed topology generation method is built to avoid such situations; livelock is avoided by using only minimum-length paths, and starvation is eliminated through

the use of round-robin arbitration on switch outputs [18]. This leaves deadlock as the only risk that must be handled. The topology generation algorithms incorporate deadlock avoidance in the structure of the generated topologies. The remainder of this section discusses this feature.

The method used to analyze deadlock situations is through the use of the channel dependency graph [18]. A sufficient condition for deadlock avoidance is to eliminate any cycles in a network's channel dependency graph. Because it is simpler in nature, the partitioned crossbar topology will be analyzed first, followed by the point-to-point topology. Since the partitioned crossbar starts by connecting all component NIs to a central switch, this situation should be analyzed first. All connections in a network are composed of two uni-directional channels. It is assumed that a channel is composed of the output port driving it as well as the actual connection wires. Figure 4.6 A shows a network consisting of 4 components connected through a central switch. As can be seen, 8 channels exist in the network, because of the existence of two uni-directional channels in each link. Figure 4.6 B shows the corresponding channel dependency graph of the system. As the figure shows, no cycles exist in this graph, because half the channels are sources and the other half are sinks, and sink channels cannot directly access any other channels (they can only insert data to the input port they are connected to). Because send and receive channels are separated thus, the only way to generate cycles in the channel dependency graph is to create cycles in the topology graph.

The second stage of the algorithm partitions the central switch, if necessary, by moving links from the central switch to a secondary switch connected to the central switch (as described previously). This process is shown in Figure 4.7 A, where a 4-port switch is partitioned. Figure 4.7 B shows the resulting change in the dependency graph; the move simply introduces an additional, intermediate channel between the source and destination channel of any links that were moved to the adjacent switch. If the partitioning process is

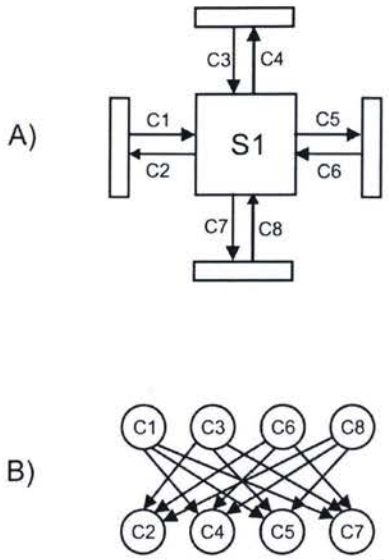

Figure 4.6: Channel Dependence Graph Example

repeated further, then additional channels will be introduced, but the dependency relation will not change. Another way of looking at this situation is to consider if the partitioning process can introduce cycles in the topology graph; currently, it cannot.

Point-to-point topologies are more complex in structure. The initial topology is generated by following the structure of the core graph. If no cycles are present in the core graph, by observation of the algorithm described in Section 4.3.1 it can be seen that the resulting network topology will have no cycles. The iterative operations performed by the algorithm are mergers and partitions. The above section shows why partitions cannot introduce cycles in the channel dependency graph. The merger process is essentially the revers of the partitioning method, and does not lead to the creation of any cycles in the topology graph either, which means it does not create cycles in the channel dependency graph.

There are situations, however, when core graphs can contain cycles. This happens when a number of master components communicate with the same subset of slave components. In such cases, the initial topology generated by the point-to-point algorithm can contain cycles, which will not be eliminated by the merger or partitioning procedures. In this case, deadlock

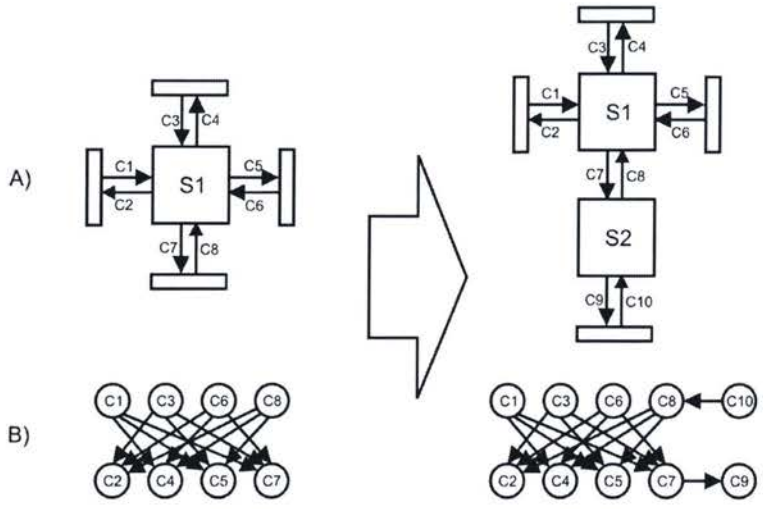

Figure 4.7: Partitioning Effect on Channel Dependence

avoidance is provided by the route generation process, which ensures that, during a search, each vertex in the topology graph can be added to the search list only once. This feature eliminates the possibility of cycles emerging.

4.4 Topology Analysis

Depending on the algorithm used, the network must be analyzed a number of times to determine its behavior. Because of this, the analysis method must strike a balance between how accurate the used network models are and the time required to analyze a given system, so that the overall process can complete in a reasonable time frame. In addition, because multiple components can actively communicate in the network, the analysis method must somehow incorporate this behavior. To address this problems, this section is divided into two parts, one covering the theoretical aspects of analyzing and predicting the behavior of a system, and the other describing the actual implementation of the proposed algorithm.

4.4.1 Theory

The topology generator creates a topology for a given application, which is itself described by a core graph. The application can be thought of as a collection of components communicating with each other. The communication patterns of the components are known at the outset, and will not change during the lifetime of the application, a situation referred to as *well-behaved communication.* The aim of the analysis method is to recommend an operating frequency that will allow the network to meet communication requirements of the application. In situations where only one master component generates transactions, such an analysis is easy to perform, based on equation 4.1. It consists simply of finding the communication path with the longest cycle latency, and ensuring that the associated time latency is less than the maximum time latency, as specified by equation 2.2. The operating frequency can then be computed based on equation 4.1 to yield a time latency small enough to meet the throughput requirement, as specified by equation 2.2 (generally the actual time latency should be less than the maximum permissible latency, to allow for processing delay in the slave).

The situation changes when more than one master interface exists in the system. Now more than one device is injecting transactions into the network, which can interfere with each other, through contention. The effect of contention will be to add latency to transactions, as they will have to share resources. Because purely digital networks are being considered, modulation of two information streams on the same link is not possible in such cases. Two types of contention areas exist in the over-all application: network contention and exit-point contention. Network contention occurs inside network switches, as multiple transactions attempt to use the same output port; hence the need for arbitration in the output ports. End-point contention occurs when multiple transactions are aimed to the same target. At this point, transactions will have to wait for one another to complete. Both this situations add cycle delay to each transaction, from the point of view of a master component. To determine

- n represents the total number of entries in the current path.

The above cycle delay corresponds to the delay experienced for a one-way traversal of a path. Equation 4.2 is based on the pipe-lined way in which flits traverse the network. Essentially, the header flit will experience a set delay, based on the number of stages (switches) in the path, and the arbitration delay at each stage. The remaining flits are forwarded automatically, and will arrive at the destination once per cycle after the first header. A single write transaction would experience a delay of $L1_{cyc} + D_p$, where D_p represents the slave processing delay. In the case of a burst, each burst beat would experience the same delay and an n-beat burst transfer would accomplish n payload transfers, so the per-transaction latency remains the same. This assumption holds if burst beats are encoded into single packets.

In the case of read transfers and read bursts, the latency formula changes. During read transactions, a request packet is first sent by the master device, and some time later a stream of response packets is sent by the slave device back to the master, who is waiting for the response [46]. For this instance, a per-transfer cycle latency ($LR1_{cyc}$) is computed using the following equation:

$$LR1_{cyc} = \frac{(L1_{cyc} + (D_p + L2_{cyc}) + (n-1) \cdot D_{int})}{n} \tag{4.3}$$

Where

- $L2_{cyc}$ represents the one-way cycle latency from the slave back to the master device.
- D_{int} represents the inter-transmission delay for burst response packets.
- n represents the number of burst beats.

In the above equation, the latencies of the request packet and all the response packets are summed and the obtained lump latency is divided by the number of burst beats, to obtain the per transfer latency. As the equation shows, bursts will yield smaller per-transfer

latencies. The above formula is composed of two components: the forward traversal time of the request flit, and the time taken for all responses to arrive. Since response packets travel in a pipe-lined fashion, the first response packet will experience a return delay similar to Tl_{cyc}, (Tl_{cyc2} in the formula), and the remaining responses will arrive at set intervals after the first, based on the network characteristics.

As mentioned previously, contention can degrade performance in two places in the network: switch output ports and slave network interfaces. Specifically, contention in the system will increase the values of Ds, the arbitration delay of a header flit, and D_p, the processing delay of a target slave. To determine by how much, on average, these values will increase for a given traffic pattern, one must determine the average number of transactions that can be encountered at these two points in the system. Once that information is available, increased latencies can be computed based on the behaviors of the network components.

To determine the extent that transactions would overlap, and therefore interfere with one another during regular operation, the topology graph is converted into a collection of simplified models that are then using for simulation. Each model consists of a list of token generators and a token consumer. The token consumer corresponds to a switch output port, while the token generators correspond to the input ports that access the output port, as shown in Figure 4.8. The tokens represent transaction packets passing through the system. Each token generator has an associated number of tokens that it must produce per unit time (1 ms), which is based on the aggregated traffic volume that arrives at the input and targets the current output port. Because of this, the same input port can appear in more than one simplified model, as it may forward information to more than one output port. The number of transactions per unit time, and therefore the number of tokens being generated, is computed by dividing a required volume of information in a path per unit time by the transaction data width. Because multiple transaction streams with different transaction data-widths may pass through the same input and target the same output, the streams are

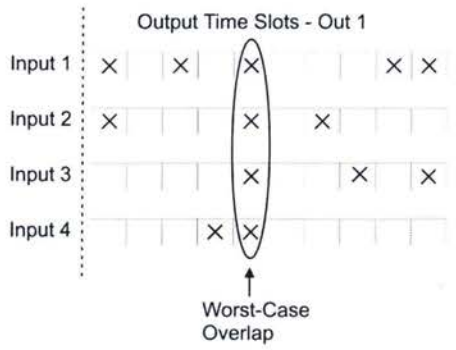

Figure 4.9: Output Contention Example

- n represents the total number of input ports targeting the current output port, and therefore the maximum number of potential overlapping transactions.
- Nt_k represents the total number of time slots where k transactions overlapped.
- $Slot_t$ represents the total number of slots.
- Lat_{nom} represents the nominal arbitration latency in cycles.
- Lat_t represents the traversal latency of a packet passing through a switch (arbitration and forwarding delay).

The above equation specifies that the delay experienced by each transaction increases proportionally with the number of transactions overlapping, and is the same for all transactions. This is true in situations where round robin arbitration is used, since all transactions will experience the same amount of delay in overlap situations. If some form of priority-based arbitration were used, a set of equations would be used, one for each priority level. Finally, in first-come-first-serve situations, the individual delay seen by each port would depend heavily on its relative transaction throughput compared with the other contenting ports.

End-point contention at the network interfaces of slaves can be computed by observing the number of overlapping transactions on the output ports immediately up-stream of the

network interface input. This equates to finding the contention level on switch output ports connected directly to network interfaces. Given that a number x of transactions will overlap at the last switch output before the interface, it is reasonable to assume that these transactions will also interfere at the network interface, since they are all being forwarded there. The processing delay D_p experienced by any one transaction will be augmented by a value D_{cont}, computed as follows:

$$D_{cont} = (C_i - 1) \times (L_{burst} \cdot D_{ppk}) \tag{4.6}$$

Above, C_i represents the contention level at output i which is directly connected to the current network interface, L_{burst} represents the worst case burst length of transactions arriving at this destination and D_{ppk} represents the processing and packing delay of a slave. This value is used because in worst-case situations a transaction will have to wait for a burst read transaction that has just started. The value of C_i is reduced by one to account for the fact that the above analysis is conducted from the point of view of one of the overlapping transactions, and the added delay should, therefore, be caused by the remaining transactions.

In the simplified models specified above, the number of distinct time-slots available at the output port is based on the frequency of operation of the network. Subsequently, contention in the system is based, partially, on the frequency of operation. However, contention analysis is being used to determine a valid minimum frequency of operation for the proposed system. This cyclical relationship between the frequency of operation and the contention in the system means that it is not possible to start from one parameter and derive the other, especially if the minimum frequency is desired.

Because of this, the analysis method follows the following approach: it selects a frequency of operation which it fixes, and then determines the contention in the system. This process, however, is repeated for a large number of frequency points, until the minimum valid fre-

quency is found. A valid frequency is one where none of the latency requirements in any of the paths are violated. The algorithm starts from a fixed starting point and first finds a valid frequency of operation. It then finds an invalid frequency of operation. Then, on the interval defined by these two values, it performs an interval halving process, until the minimum valid frequency is found. At each frequency point being considered, the contention analysis is performed and the path latencies are verified for all connection paths in the system.

4.5 Method Limitations

Having described the characteristics of the proposed topology generation and analysis method, it is important to also establish the limitations of said method. Some of the above paragraphs have hinted at certain limitations or areas of improvement, but this section aims to formally list them. The first limitation, obvious from the title of the paper, is the fact that the proposed method targets systems using transaction-based component interfaces. Both the topology generation and analysis method are built with this underlying assumption in mind, and would not be appropriate for stream-based architectures, for example. The decision to target transaction-based components and protocols was made because a large number of on-chip systems take this approach to their design. The second obvious limitation comes from the supported network parameters. The current system supports best-effort wormhole switched networks, which are being pursued for on-chip applications. However, circuit-switched or time multiplexed networks also exist. The analysis method would have to be altered before it could support such networks. As well, the method is not equipped to handle adaptive systems, in particular adaptive routing procedures.

The third set of limitations are related specifically to the proposed topology generation and analysis procedures. Currently, both topology generation algorithms support only single links between network components, be they switches or NIs. Multiple links between

switches are not currently supported, although they could be useful in minimizing contention, as shown in [40]. As well, the route generation method currently used, based on the shortest path algorithm, would have to be updated to incorporate cycle prevention procedures. Finally, only one of the proposed topology generation methods is guaranteed to be two-dimensional in nature, that being the Partitioned Crossbar method. The Point-to-Point algorithm follows the application core graph and can, in certain situations, have overlapping links. This would make actual layout of such systems more complex, depending on the application. As well, component performance is affected by the final layout of the proposed topology, on-chip; the proposed method does not address limitations that can emerge due to physical considerations. For example, the need for intermediate buffering on long links to allow for high-speed network clocks, and the effect this can have on performance is not currently addressed.

4.6 Conclusion

The preceding chapter presents the actual structure of the topology generation algorithms, as well as an analysis of their complexity and limitations. The algorithms are latency oriented, and incorporate partitioning and merging procedures designed to benefit components with high transaction throughput requirements. The chapter then presents a method of analyzing a generated topology that can incorporate the contention expected in a real system. A specialized form of simulation is used, based on Petri Net representation of the switches in the network. Individual models are analyzed in isolation and the computed contention effects are, then, computed for complete paths in the system. This approach helps reduce the analysis computation time.

Chapter 5

NoC Simulation Environment

The topology generation and analysis method being proposed in this document concerns itself primarily with the attainable performance of a proposed network. Because of this, some method of performance evaluation is needed. This chapter introduces a simulation environment based on the SystemC transaction-level modelling language. The simulation models are built to support a certain type of network and to allow parametrization of transaction generation and encoding. The chapter's three main sections will describe in detail the supported network model, the parametrization options available in all models, and the implementation of the models themselves, from a behavioral point of view.

5.1 Supported NoC System

As with large area networks, NoCs can have a wide range of characteristics. Before describing how the simulation models were implemented, the targeted NoC model must be defined. This section describes the supported features of the simulation models, including the packet format, switching methods employed and routing procedure used. These models were built for simulation purposes, which means that some of their characteristics can be parameterized

from one simulation run to the next. The simulation environment supports a packet-based best effort system using source routing and wormhole switching. The following subsections will elaborate these aspects further.

The general structure of the supported NoCs has been presented in past works [19, 20] and consists of on-chip components communicating with each other using the NoC infrastructure. These components access the NoC system using Network Interfaces, which are dedicated logic circuits meant to convert information from the format used by the on-chip components to a format used by the network as shown in Figure 5.1. This process will be discussed in more detail in the section discussing packet format. The presented models support the AMBA AXI protocol [14] for on-chip component communication. This protocol was selected as it is the latest iteration of the popular AMBA line of protocols, and unlike previous iterations it is a dedicated point-to-point protocol, which hides infrastructure details from communicating components. This features allows on-chip components to be simpler, and usable in different designs as no pre-built support for a particular communication system (shared bus, cross-bar) is required.

Packet Format

Amba AXI information is transferred in parallel, using information words of 8-1024 bytes. Multiple such transfers can be grouped together in a burst transfer for more efficient communication, but the atomic transfer size remains the same. One of the features of Networks-on-Chip is the use of relatively narrow buses between individual components. This makes actual routing of a chip layout easier and permits the components themselves to operate at higher frequencies in some instances. Because of this, the primary aim of a network interface is to serialize incoming information for transmission through the network. In addition, the network interface must take the support information associated with a transfer (address and

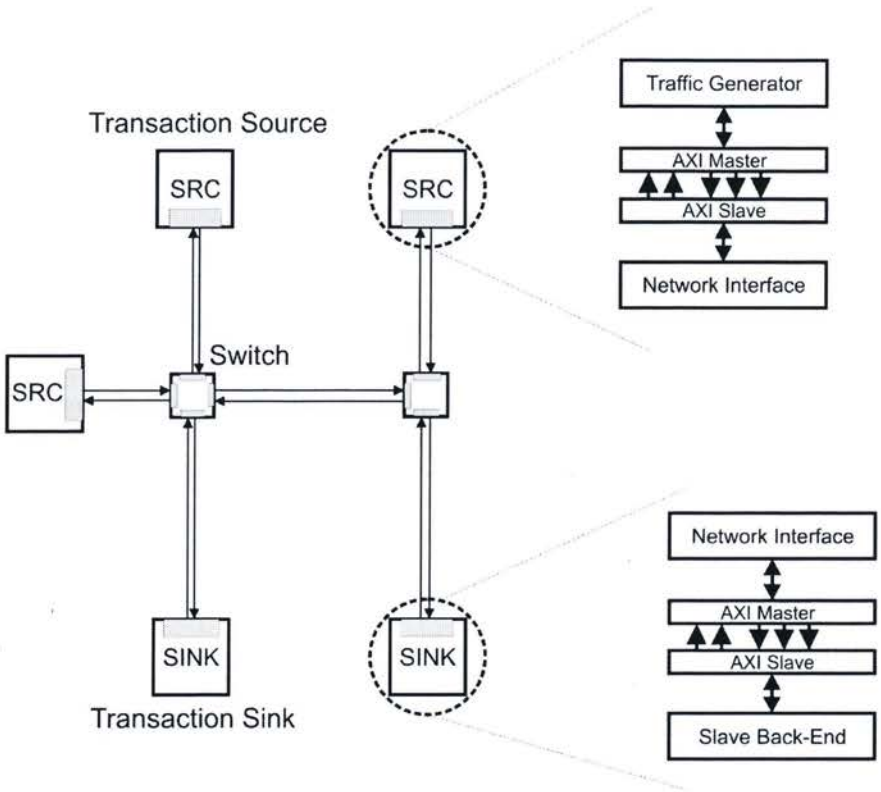

Figure 5.1: Structure of Intended Systems

side-band information) and transmit it along with the data itself.

Each packet is built around an information transfer, meaning that a single transactions will result in a packet. Burst transfers of multiple beats will generate a packet for every transfer generated. Write transactions are converted to information packets which are then sent from the master interface to the slave interface through the network. Read transfers consist of a request packet sent by the master interface, and a number of response packets sent by the slave interface. The number of data packets and response packets for read and write transfers corresponds to the number of burst beats for the transaction. Each packet sent across the network contains the following information regarding the transaction being performed:

- Transaction address.

- Transaction type.
- Additional sideband information.

Write packets and read response packets will also contain actual data.

Because wormhole routing is used, each packet is broken into smaller flow-control units (flits) which are processed by the network. Flits are either of header type or body type. Header flits encode the routing information needed to transfer information through the network, while body flits contain the actual transaction information (address, data and sideband information). Each flit has an associated ID number encoded into the flit's most significant bits. Apart from this value, all other data encoding is variable, and can be specified for a given system, along with the number of header and body flits.

Routing Method

The model implements static routing based on look-up tables, and assumes source routing [5]. This means that route information is encoded into packets and sent through the network. This eliminates the need for storing route tables at each switch and reduces switch area. However, this does increase the size of packets in the system, as well as the data overhead per transaction. The route tables are stored within the network interfaces, as dedicated read-only memories. Routes are selected based on the base address of the targeted devices.

The routing method used is referred to as "Street-Sign Routing" [19], as it essentially consists of a series of directions. A route consists of a series of bit fields, each of which identifies the output port in a switch to which the packet should be forwarded. The route for each packet is encoded into the packet's header flit(s), along with a counter variable which gets incremented at each switch. Figure 5.2 shows a 2-hop route, with the packet being forwarded to port 2 at the first switch, and port 0 at the second switch. In each switch, the output port to which the packet must be forwarded is decoded based on the

value of the counter field and the route table. As it passes through the switch, the counter field is incremented, so that it points to the next bit field.

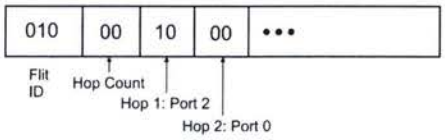

Figure 5.2: Example of a 2-Hop Route

Switching Method

Wormhole switching was first proposed in the field of parallel computing, as it could reduce the buffering requirement at individual switches, and speed up throughput through the system, by transferring flits in a pipelined fashion [18]. These characteristics make wormhole switching attractive for on-chip systems where area is at a premium [19, 20]. As was mentioned above, packets are divided into flits, with some flits (the header) containing the route information. This means that a switch can start arbitrating and forwarding part of a packet as soon as the header flits have been received and the output port was decoded.

In the proposed simulation models, switches begin arbitration as soon as all header flits have been received. Once arbitration completes, flits are forwarded to the output port and out of the switch. At the same time, the path between the input port and output port is locked until all flits of a packet have been forwarded. When this happens, the path is opened and arbitration can begin again. To ensure that starvation does not occur in the system, the switch models implement round-robin arbitration amongst the petitioning input ports.

5.2 Parametrization Options

To permit the implementation of a wide range of systems, the developed simulation models were designed so that most aspects of packet design could be specified at run-time. The behavior of the transaction generators (master interfaces) can, also, be controlled to create various types of traffic. The following sections will give a detailed description of the parametrization infrastructure present in the system models.

Access Patterns

The component interface models are the traffic generators for the proposed system. They generate transactions that are then transported to various destination interfaces. To allow for the simulation of various systems, the traffic generators must be programmable so that they generate transactions of various types (length, flow-type) to specific interfaces (identified by their base address). This pairing of transaction type and transaction destination will be referred to as a spatial access pattern form now on. In addition, the distribution of transactions in time must also be controllable. The distribution of transactions in time will be referred to as a temporal access pattern.

Both types of access patterns are specified to the master interface in the form of configuration files. Spatial access patterns are presented as series of entries, with each entry containing the following information:

- Transaction destination, in the form of a base address and NoC identifier.
- Transaction burst length.
- Transaction type (read, write or both).
- The distribution of read and write access, in the form of a percentage value.

- The over-all access rate to the current destination.

The over-all access rate refers to the fraction of all accesses generated by the interface that should go to the current destination. The total access rates for all spatial access patterns must add up to unity; if an interface generates transactions for only one destination, then that destination will have an over-all access rate of 1.0.

Temporal access patterns are built on top of spatial access patterns. The complete list of temporal access patterns specifies a pool of available transaction destinations that can then be arranged in time. Two types of temporal arrangements are available: random Poisson process transaction generation [47] and fully specified temporal patterns. Poisson temporal patterns take the form of Poisson events, meaning that the inter-event time is a random variable with a decaying exponential probability density function of the form:

$$p_T(T) = \alpha e^{-\alpha T} \tag{5.1}$$

The inverse of the α parameter specifies the generation rate, in transactions per cycle, and a transaction destination is selected based on the over-all access rate defined above. This process will be described in more detail in later sections.

Fully specified temporal patterns eliminate the random aspect of transaction generation. Instead, the temporal arrangement of transactions is specified by the user. The patterns are specified as a sequence of entries, as with spatial patterns. Each entry contains the following information:

- Target spatial destination.

- Number of transactions to the current destination.

- Number of wait cycles before the next temporal pattern.

If a transaction generator is configured to generate temporal patterns, it will continuously loop through all the entries in its list of temporal patterns. Each entry specifies the destination and a number of transactions that are to go to that destination, followed by a wait period, in cycles, before the next temporal pattern is invoked. Figure 5.3 shows a possible arrangement of temporal patterns.

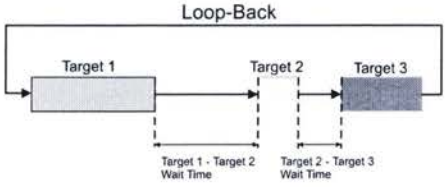

Figure 5.3: Example Temporal Pattern

Transaction Encoding and Route Tables

The second parametrization aspect of the models is the encoding of the transaction information into packets. For any system, the number of header and body flits per packet can be specified. Once this is done, information encoding structures specify to the system how the route tables and transaction information are encoded into the flits.

Encoding specifications are provided to the system using a set of data structures. Each such structure specifies a flit location, a shift amount, and a bit-mask, and there is one such structure for every piece of data to be encoded (such as address, flit ID, route information, etc.). The flit location specifies which flit the data will be encoded into. The shift amount specifies the actual bit location of the start of the data, and the bit-mask can be used to isolate the current data. The following transaction information is encoded into every packet sent through the network:

- Transaction source.

- Transaction destination.
- Transaction data.
- Transaction address.
- Transaction side-band information.

The route tables for each packet are encoded into the header flits of the packet. As described earlier, the routes consist of a count variable and the actual port indicators for each hop in a route. The count variable, as well as each port indicator entry have dedicated encoding structures that specify where each piece of data should go.

5.3 Simulator Models

Having described the supported network model, as well as the parametrization options that are available within the environment, this section describes the actual implementation of the traffic generators and sinks, network interfaces and switches. The main focus of the section is in describing the operational behavior of these components, and how the parametrization options described above are incorporated.

5.3.1 Traffic Generators and Sinks

The traffic generators (master components) periodically generate a transaction which is passed to the network interface and transported to a slave component (traffic sink). The slave components will either consume the incoming transactions (in the case of write transfers) or generate a series of response packets (during read transactions). The traffic generator behavior will be described first, followed by the sink behavior, which is less complex. Both

master and slave components incorporate AXI interfaces. Master components incorporate AXI Master Interfaces, and slave components incorporate AXI Slave Interfaces.

All simulation models can be thought of as finite-state machines, each of which performs some actions in each state. The state diagram of the traffic generators is shown in Figure 5.4. After the de-assertion of the reset signal, the master component goes into its main generation state, where a read or write transaction is generated. The selection of the transaction parameters will be discussed in more detail later. The transaction is sent to the AXI interface, and the master component goes into a wait state, until the AXI interface finishes the transaction in question. At this point, depending on the type of temporal access patterns that were selected, as well as the length of the last transaction, the master component will either proceed to the generation state, or it will go into a second, self imposed wait state. The wait period for the second wait state is specified by the α parameter discussed above.

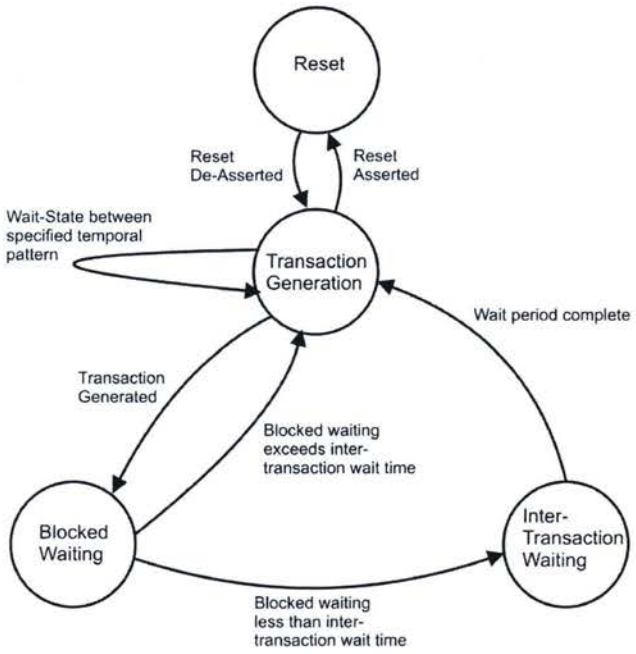

Figure 5.4: Transaction Generator States

During the transaction generation state, the master component must specify a transaction

target, a transaction type (read or write), the burst length of the transaction and transaction data in the case of write transfers. Because the simulation models are concerned with the flow of transactions through the network rather than the actual content of the information, random data is currently used as payload. The transaction target is selected based on the list of spatial access patterns provided to the generator. The parameters of the transaction are selected in the following order:

i. Transaction destination.

ii. Transaction type.

iii. Burst length.

In the case of Poisson process transactions, a uniform random number generator is used to create a random pointer value in the range of 0.0-1.0. This pointer value determines which target is selected by observing what range it falls in. The ranges are constructed using the over-all access-rate to each destination. In a situation where two possible targets exist and are equally likely, the range $(1.0, 0.5]$ will correspond to the first target, and the range $(0.5, 0.0]$ will correspond to the second target. Once the destination is selected, the other parameters of the transaction (burst length, and type) are selected in a similar fashion, from the parameters specified for the current access target. In the case of fully specified temporal patterns, the destination selection process is eliminated by the specified pattern of accesses. An index is used to sequentially target each destination specified by the temporal patterns. In-between accesses to a particular target, the master component waits in the generation state, but does not generate transactions until the wait time between each target is complete.

Contention in the network can lead to variations in the time a transaction takes to complete. This, coupled with the fact that each generator blocks until a transaction completes

before continuing its operation can alter the characteristics of Poisson process temporal patterns. The over-all generation rate is based on the inter-transaction wait time as discussed earlier. However, the wait time now fluctuates, due both to the random nature of the process, and to the contention in the system. This means that the over-all generation rate of master interface fluctuates. If the original rate is extremely high, an under-run situation will occur, where less transactions are generated, since the inter-transaction time is increased. The degree of the under-run effect is dictated by the amount of traffic in the system, and the resulting transaction delays. Conversely, master components with small over-all generation rates can generate more transactions than originally specified (because of the variation in the random inter-transaction wait time).

The traffic generator models attempt to minimize this variation in the effective over-all generation rate, by continually keeping track of the time spent in the blocked mode (state 2 in the diagrams). In cases where the wait time is less than the average inter-transaction time for a given over-all rate, the master component will generate an adjusted wait time, based on the transaction rate and the time already spent blocked. In situations where the blocking time exceeded the specified average wait time, the master component by-passes the second wait state shown in Figure 5.4 and proceeds directly to the generation state, in an attempt to "catch up" with the required generation rate. At the same time, each traffic generator keeps a record of the number of transactions completed over a period of time, and can compute the average generation rate over a time segment. This information is used by the master components to ensure that the average generation rate is not exceeded; if the measured generation rate exceeds the specified rate, transaction generation stops for the remainder of the current time period. Figure 5.5 shows an example of how cores deal with blocking delay. Core 1 has exceeded its inter-transaction time simply waiting for the previous transaction to complete, so it immediately generates a new transaction. Core 2, on the other hand, generates a wait time based on its over-all rate and the time spent waiting

in the blocked mode.

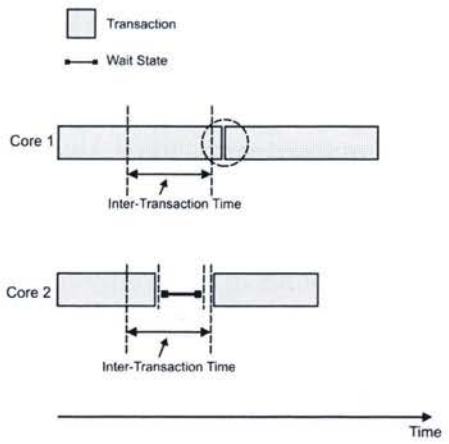

Figure 5.5: Adjusted Transaction Generation

Traffic sinks (slave devices) have a simpler role to play, as they must deal with already existing transactions. When a transaction is received by a slave, it enters a busy state for a period of cycles, which represents the processing delay of the slave; the delay parameter itself can be specified for every slave before the beginning of simulation. When the processing delay is finished, the slave has two options. In the case of write transactions, the slave returns to the ready state, and waits for further transfers. During read transfers, the slave enters a response stage, and generates response packets for the transaction; once this process completes, the slave returns to the ready state.

5.3.2 Network Interfaces

Once transactions are generated by the master components, they must be converted to the appropriate format before they can be transmitted over the network. This task is fulfilled by the network interface modules in the system. Two types of interfaces exist: master and slave network interfaces. They differ in their over-all behavior, as well as in the components they incorporate. Master network interfaces incorporate an AXI slave interface, allowing

them to communicate with master components. The slave network interfaces incorporate AXI master interfaces, which allow them to interface with system slave components.

The structure of a Master Network Interface is shown in Figure 5.6. The AXI Slave interface is used to interface to the master component, the output block is used to construct and send packets, and the input block receives response packets. Finally, the control unit updates the state and control signals of all other components based on the current state of the interface. Master network interfaces, in their quiescent mode, wait for transactions from their AXI interfaces. When such a transaction is received, the network interface proceeds to construct a network packet for every beat in the transaction. Data is encoded into each flit using the method and data structures described in Section 5.2. The network interface stores route records for all potential transaction targets for the current component. When a transaction is received, the appropriate record is selected using the transaction address. As soon as a flit is constructed, it is sent over the network so long as there is enough buffer space available in the downstream input port. Once a packet has been completed and sent, the network interface either returns to its quiescent mode, or goes to a wait state, depending on the transaction type. In the case of write transactions, the master network interface will return to its quiescent mode and wait for the next beat in the transfer (if there is one). In the case of read transactions, the network interface enters a wait state, until such time as its input port signals it that a complete response packet has been received. This packet is then forwarded to the AXI slave interface and further to the master component.

The structure of the slave network interface is similar to that of the master network interface, but backwards, as it were. In its quiescent mode, the slave network interface waits for packets to arrive from the network. Once a packet is received, the network interface will decode it, and forward the transaction to its AXI master interface and further to the system slave. If the received packet was a write transfer, the interface will then return to its quiescent mode and wait for further packets from the network. In the case of a read transaction, the

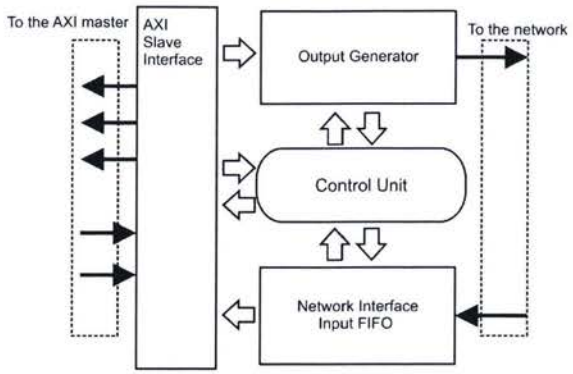

Figure 5.6: Master Network Interface

network interface will forward the transaction information to the slave component, and will then move to a transmission state. In this state, the interface can transmit the slave read response as response packets. Write packets are forwarded to the component as soon as they are received, even if they are part of a burst. This is done so that the network interface is not blocked more than necessary for any one transfer. As well, this allows the network to interleave multiple writes to the same slave component in time.

5.3.3 Interfacing Between the Core and NI: the AXI Protocol

The above subsections describe the traffic generator and sink models, as well as the network interfaces used to access the network. The current subsection examines how the generators and sinks (essentially the actual components that make up the SoC) communicate with the network interfaces. Any number of protocols could be used to interface on-chip components, starting with dedicated protocols, designed specifically for one set of components. However, such an approach reduces re-usability, which is why most computing systems use standardized interfaces between components. In the area of on-chip systems, a number of such standard protocols exist and were discussed in section 2. The latest implementation of the AMBA protocols target point-to-point interconnects similar to the OCP, in the form

of the AMBA AXI interface [14]. This protocol was selected as the interface mechanism between generators, sinks and network interfaces.

This section will not describe in-depth the operation of the AXI protocol; for such purposes, there is ample documentation available [14]. Rather, this section describes how the protocol was implemented for the current simulation environment. The AXI protocol specifies the communication method between two entities, one of which is considered the master or initiator, and the other the slave or responder. To transfer data, five channels are specified and used. Two address channels are used to transfer address information as well as transaction information between the master and slave. The two channels are the Read Address Channel and The Write Address Channel. Each one carries the address for read or write transactions, respectively, along with additional data such as the length of the burst, caching description of the data and others. This information is referred to as the side-band information. The AXI protocol specifies that the Read and Write Address channels can be merged in certain circumstances; however, the simulation environment uses separate address channels. The data on the address channels is written by the master interface, while the slave uses only one signal for hand-shaking purposes. The master also controls the Write Data channel, where data for a write transfer is written. The slave, on the other hand, writes data to the Read Data channel, and to the Write Response channel. The channels, and their drivers, are shown in Figure 5.7.

In the simulation environment, the Read and Write Data channels carry place-holder variables, since the operation of the components is not modelled beyond their role as generators or sinks. The address information is obtained from the configuration files specified to the traffic generators, and is used to specify which component is being targeted. However, no offset values are added, again because of the high-level of the component models. In the case of the side-band information, only the transaction type and burst length are encoded, while the other fields are left blank. Finally, the AXI protocol specifies that write transfers must

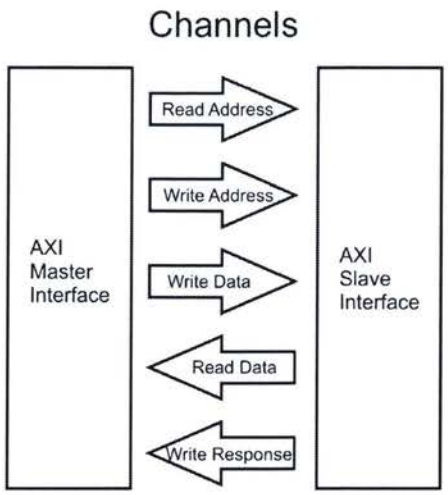

Figure 5.7: AXI Channel Arrangement

also contain a response stage, where the slave components acknowledges receiving the write information correctly. However, in the interest of performance, the simulation environment transfers responsibility for the response stage to the network interface rather than to the slave device being targeted. This means that as soon as all the write information has been received, the network interface drives the appropriate signals in the Write Response channel, completing the transfer, from the traffic generator's point of view. The network essentially assumes responsibility for the safe delivery of the write data to the destination. This allows the traffic generators to proceed to a new transaction if needed, rather than wait for the write response to travel back from the slave device across the network. Figure 5.8 shows which fields are actually used inside the simulation environment during an AXI transfer.

Because currently the simulation environment encodes burst beats as single packets, it is important to distinguish between the arrangement of a transfer at the source and at the destination. In the case of write transfers, a transaction that has 4 burst beats at the source will become a set of four 1-beat bursts at the destination, because the slave network interface generates a write transfer as soon as a write packet is received. This approach is used to allow multiple write transactions targeting the same device to be interleaved, even during

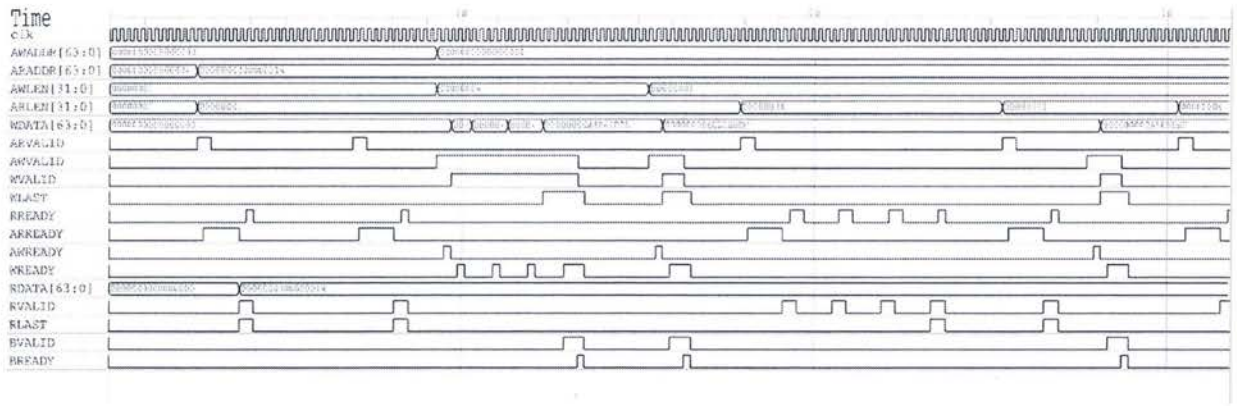

Figure 5.8: AXI Implementation Example

burst operation. Note that this approach would have to be further expanded in situations where shared memory is used for information passing, to ensure that race conditions or out of order information storage situations do not occur. A read transaction starts as a single packet sent by the traffic generator, and the response consists of a number of packets equal to the burst length of the transfer, generated by the slave network interface. The read response packets look similar to write packets, but travel from slave back towards the master device.

5.3.4 Switches

The network switches are perhaps the simplest components, behaviorally; a high-level diagram of a 4-port switch is shown in Figure 5.9. Every switch consists of essentially two component types, replicated and connected as many times as necessary to achieve the desired number of ports. The two components are an input First-In-First-Out unit, with some decoding capabilities, and an output forwarding unit, with integrated arbitration. A complete switch port is composed of an input FIFO and an associated output unit, as the figure shows. Each output unit can receive information from three input units, excluding the input unit associated with the current port. This means that a switch cannot, alone, return information to a sender.

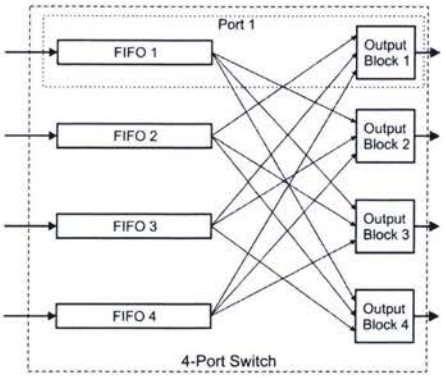

Figure 5.9: 4-Port Switch Diagram

The FIFO component of each port accepts incoming packet flits, and buffers them until such time as they are forwarded by one of the output units. Each FIFO performs a decoding operation on the flits that are at the front of the queue, to determine which output port the current packet has to be forwarded to. Before run-time, each FIFO is configured with details about the flit format. In particular, the number of header flits and the location of all routing entries is specified. In this way, the FIFO can start polling the required output port as soon as enough flits have been received to permit route decoding. A FIFO's depth can be specified individually for each switch in a system, to determine how performance varies with the buffering capacity of input ports. Each FIFO incorporates a counter which tracks the number of flits currently buffered. Once a FIFO's capacity is reached, the FIFO control unit will indicate to the up-stream output port that it is full, implementing a form of "stop-go" flow control in each inter-switch link.

Each output unit must forward flits from the input FIFOs of the switch to the input down-stream. The output block selects an input port from the pool of input ports actively petitioning it by using a round-robin arbitration scheme. This ensures against starvation of any one input, while at the same time keeping the block complexity down to a minimum. Once an input is selected, the output block will forward the flits coming from that input

for the duration of a packet. Once the final flit passes through the output block, the block returns to its quiescent state, waiting for and arbitrating amongst input petitions. The state diagram of the Output block is shown in Figure 5.10. To deal with the possibility of a downstream FIFO being full, each output unit must be able to buffer one flit until such time as the downstream FIFO circuit can accept it.

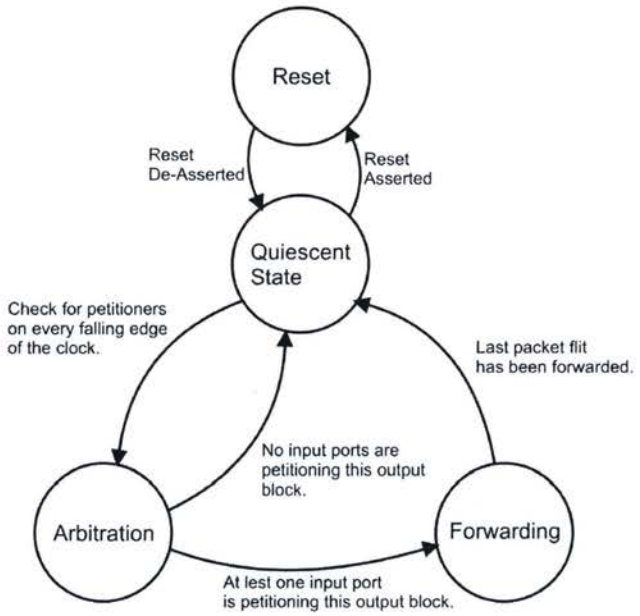

Figure 5.10: Output Block State Diagram

5.4 Conclusion

The preceding section has presented the simulation environment developed and used for the evaluation of the proposed design and analysis method. The simulation environment, implemented as a collection of SystemC models, allows the specification of any topology and a wide variety of encoding options for a given network structure. The traffic generators developed for the environment are based on on-chip component behavior, and incorporate actual signal-level implementations of the AMBA AXI protocol.

Chapter 6

Simulations and Results

In this section, a number of applications are used as tests for the proposed topology generator and analyzer, in conjunction with the simulation environment. The aim of this section is to examine how proposed topologies behave when compared with traditional regular topologies, as well as determining the accuracy of the prediction algorithm that is being proposed. Additional features will also be discussed where appropriate. The chapter begins with a description of the applications that will be used for comparison purposes. The following section presents simulation results aimed at comparing the generated topologies with regular topologies. The section will begin with a description of the tests performed, followed by the results obtained. A second results section analyzes the accuracy of the topology analysis engine and the recommended operating frequency. The third results section lists execution times for the topology generation and analysis procedure. A fourth section compares one of the proposed topologies with an irregular topology proposed in past literature. Finally, the last section examines the effect of using message-passing models with transaction-based components. In the following section, all results are listed in the form of graphs. The same information is available in tabular form in the Appendix.

6.1 Test Applications

This section presents the four applications used to test the proposed method. Each subsection presents the component structure and communication requirements of the application, in the form of its core graph C. In addition, the communication characteristics of each edge in the graph, such as the data-width and burst support are presented here, with reasons for the selected parameters, where necessary.

MPEG4 Decoder

The first application is an MPEG4 decoder, first proposed by van der Tol and Jaspers [48]. This multimedia application exhibits large on-chip communication requirements, and makes for an interesting test application for NoC systems. For this reason, it has been used in the past as a potential application in NoC design examples [27]. The application core graph is shown in Figure 6.1 [19]. The communication volumes shown are in MB/s, meaning that the application has very large communication requirements (the Up Sampler, Core 6, exceeds 1500 MB/s). Because of this, it is assumed that cores 4 and 6 (rast and up-sampler units) have data interfaces of 128 bits, and communicate using 16-beat bursts. All other master components are assumed to have 32-bit data interfaces and communicate in 4-beat bursts, except for the RISC processor, which uses 16-beat bursts. The memories are assumed to have 128-bit interfaces, to achieve greater data throughput.

Multi-Window Display Application

The multi-window display (MWD) application is based on a companion chip designed for high-performance television applications, and initially presented by Jaspers and de With [49]. As with the MPEG decoder, it has been used for NoC testing in the past [27]. The original chip was designed to accomplish various operations more efficiently by using dedicated pro-

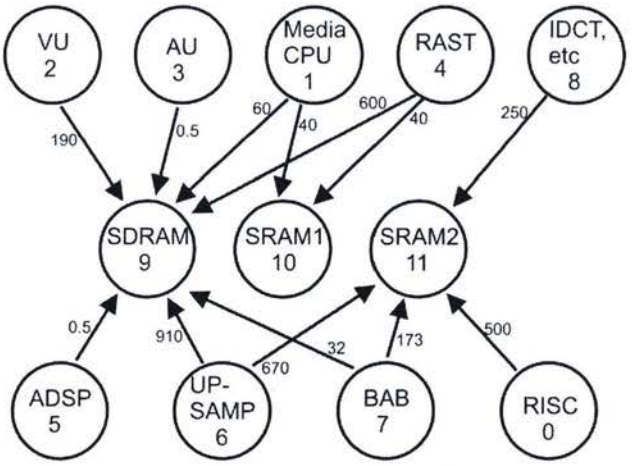

Figure 6.1: MEG4 Decoder Core Graph [19]

cessing elements; here the processing elements are connected using an NoC infrastructure. The core graph for the MWD application is shown in Figure 6.2; as before, the information volumes are in MB/s. Since all the application cores are dedicated to multimedia processing, it is assumed that all cores have large, 128-bit data interfaces, and all communicate using 4-beat burst transfers. Finally, most of the on-chip components have both master and slave interfaces to permit the implementation of information pipelines. For this reason, the core graph shows both master and slave interfaces for these cores, connected using a dotted line. The application core graph was presented in [27], and has been updated to incorporate master and slave interfaces.

Audio-Video Benchmark Application

The third test application is an audio-video benchmark presented by Hu et al. [34]. A core graph was constructed based on the task flow presented, and is shown in Figure 6.3. All information volumes are MB/s. Three additional memories were added to the application, to permit communication between the DSP and processor components (components CMEM1, CMEM2 and CMEM3). It is assumed that the ASIC components have dedicated master

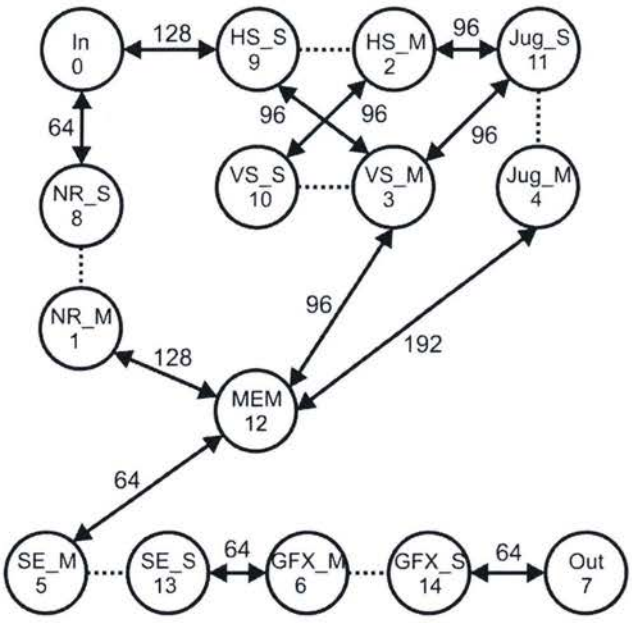

Figure 6.2: MWD Application Core Graph

and slave interfaces (with the exception of ASIC1, which needs no master interface). Given that the application is multimedia-oriented, large data-widths are assumed for more efficient communication. All master cores with the exception of the CPU and ASIC4 master interface have 128-bit data-widths, and communicate using either 4 or 8 beat bursts. Because the CPU and the ASIC4 master interface have the largest information volume requirements in the core graph, they use 256-bit data-widths, and transmit data using 16-beat bursts.

Layer-3 Switch

The final application is theoretical in nature, designed specifically for testing the current method. The application is a Layer-3 Switch, used specifically for backup operation of multiple servers. The switch has 6 ports, each of which operates at 100 Mbit speeds. The core graph of the application is shown in Figure 6.4, and consists of a general purpose processor used for routing, six ethernet controllers used to connect to the physical interfaces,

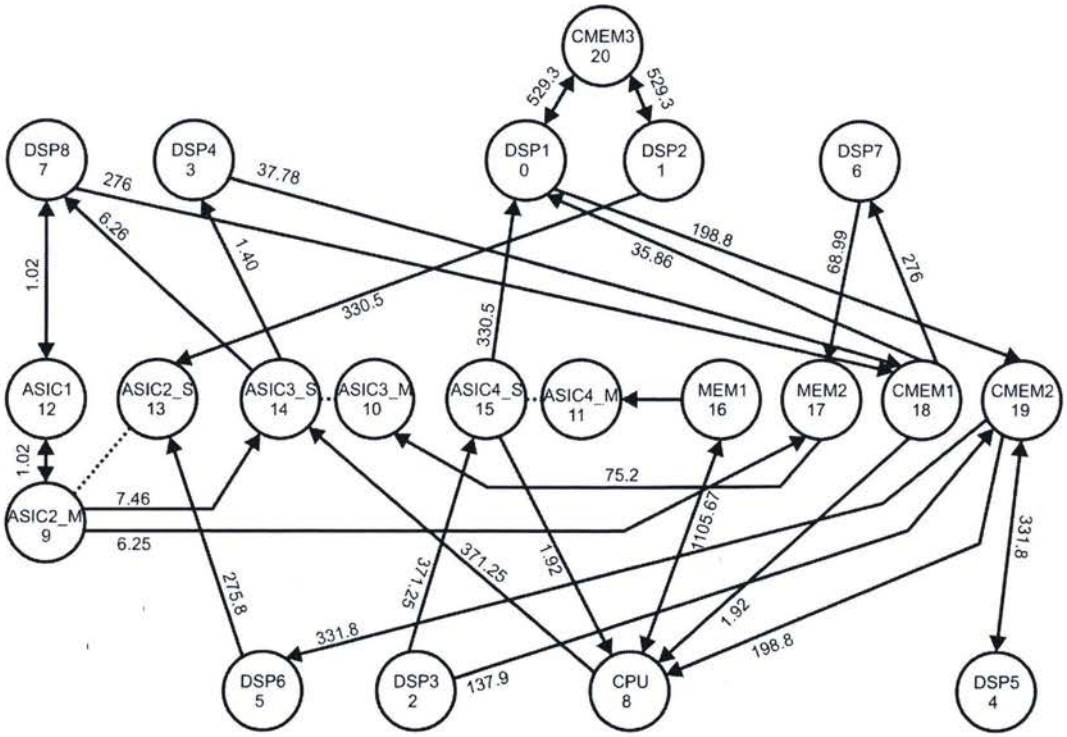

Figure 6.3: AV Benchmark Core Graph

two Direct Memory Addressing (DMA) units to transfer information between the various ports, and one shared memory being used for storage. The DMA units have both master and slave interfaces, allowing them both to be controlled by the processor and to independently transfer data. The processor, memory, DMA master interfaces and ethernet controllers all have 128 bit wide data ports, and support 16 beat bursts, because of the large data transfer requirements. The DMA slave interfaces are 32 bits wide, given that they only accept control data; for the same reason, the communication occurs in 1-beat bursts.

6.2 Topology Comparisons

The first set of tests was performed to compare the topologies generated by the proposed method with traditional topologies. The first three applications listed above were used for

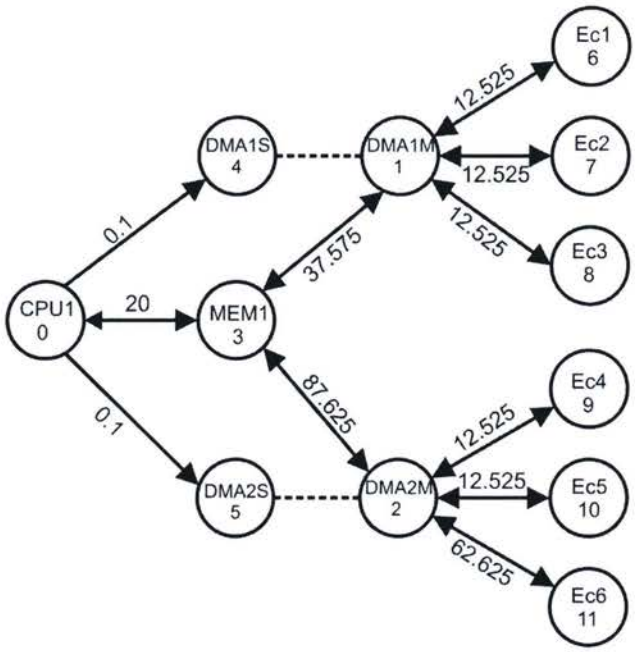

Figure 6.4: Layer-3 Switch Core Graph

these tests. The core graph of each application was used to generate two types of topologies, on based on each algorithm type. The following parameters were used for the supported NoC systems:

- Packets are 8 flits long, with 2 header flits.
- Arbitration delay of 8 cycles.
- Packing delay of 8 cycles.
- Un-packing delay of 15 cycles.
- Maximum port count limited to 10 ports.

The latencies are based on RTL models of switches and Network Interfaces. The flit length was selected to permit data widths of up to 256 bits to be encoded into one packet. Finally, the maximum port count limit was based on limiting switch complexity. Because the focus of

the research in this case is on interconnect performance, the latencies of the slave components were set to very low delays so that they do not affect the obtained results. It should be noted that a real application would not always have such reduced delays in all its slave components. Figures 6.5 and 6.6 show the obtained topologies for the MPEG4 Decoder, Figures 6.7 and 6.8 show the MWD irregular topologies, and finally Figures 6.9 and 6.10 show the Audio-Video Benchmark topologies. In the remainder of the results section Point-to-Point topologies will be referred to as Custom 1 topologies, while Partitioned Crossbar topologies will be referred to as Custom 2 topologies.

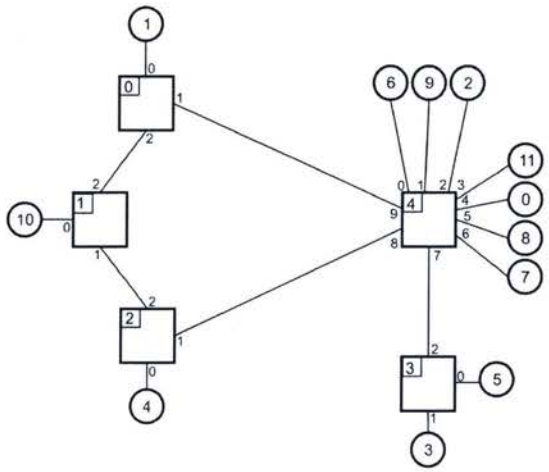

Figure 6.5: MPEG4 Decoder Custom 1 Topology

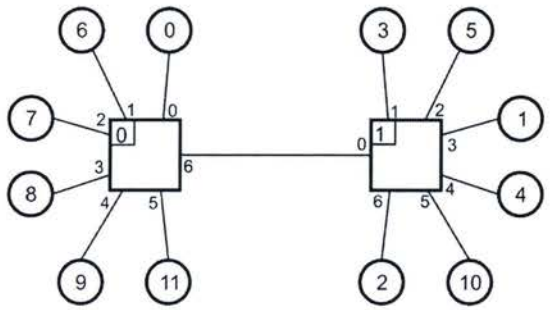

Figure 6.6: MPEG4 Decoder Custom 2 Topology

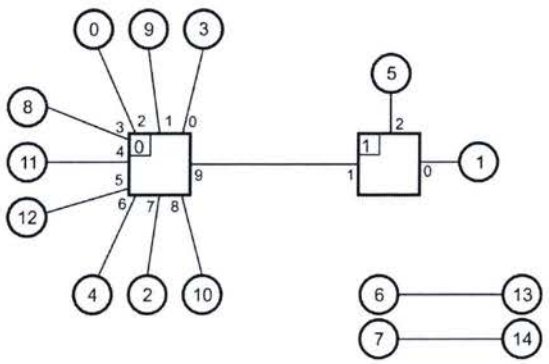

Figure 6.7: MWD Application Custom 1 Topology

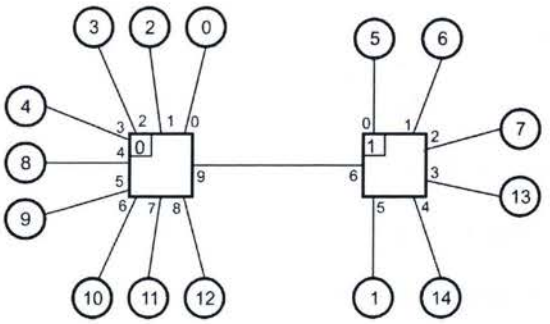

Figure 6.8: MWD Application Custom 2 Topology

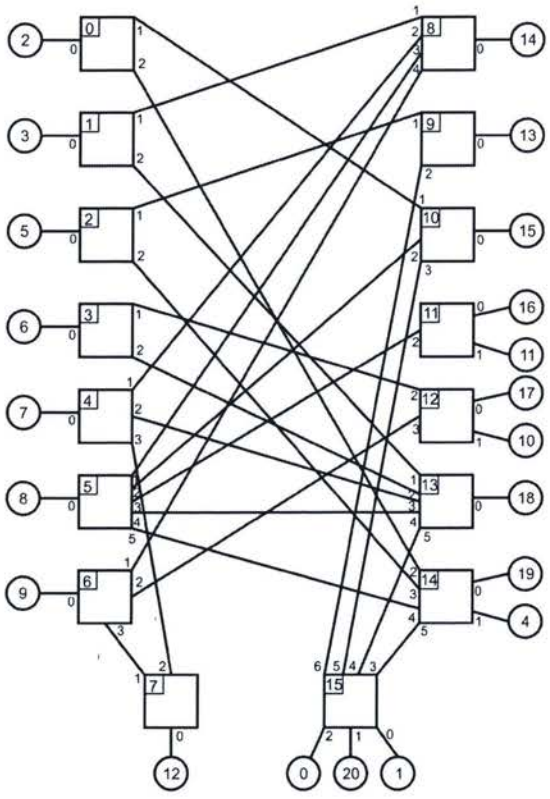

Figure 6.9: AV Benchmark Custom 1 Topology

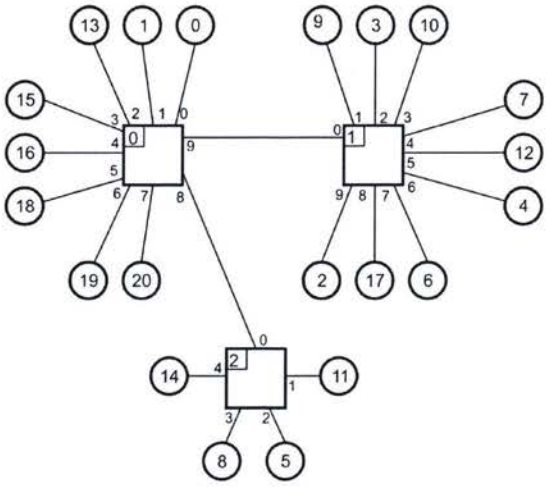

Figure 6.10: AV Benchmark Custom 2 Topology

The comparison was performed with two traditional topologies: the mesh and fat tree. The mesh was used for all three application, as it is has been proposed as a solution for on-chip systems [50]. The fat tree topology was used for the MPEG4 Decoder and the MWD Application, as an example of a performance-oriented regular topology; the topology provides increased performance due to it's redundant links [23]. The existence of multiple possible paths between sources and destination leads to reduced congestion in the system. Figures 6.11 and 6.12 show the regular topologies for the MPEG4 Decoder, Figures 6.13 and 6.14 show the regular topologies for the MWD Application and finally Figure 6.15 shows the AV Benchmark mesh topology. To map vertices in the core graph to locations in the regular topologies, a method similar to that presented in [27] was used.

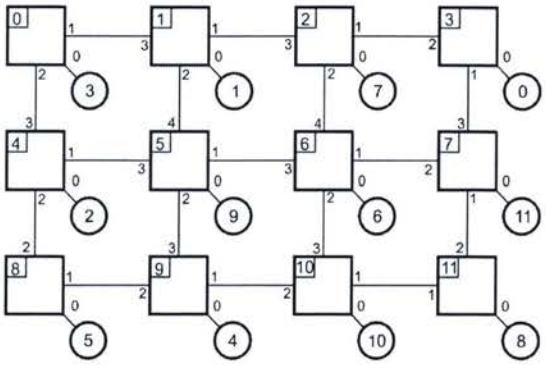

Figure 6.11: MPEG4 Decoder Mesh Topology

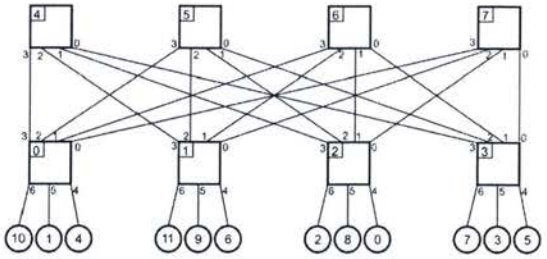

Figure 6.12: MPEG4 Decoder Fat Tree Topology

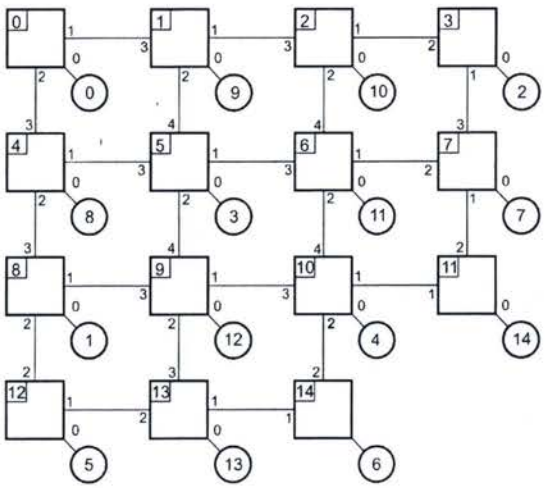

Figure 6.13: MWD Application Mesh Topology

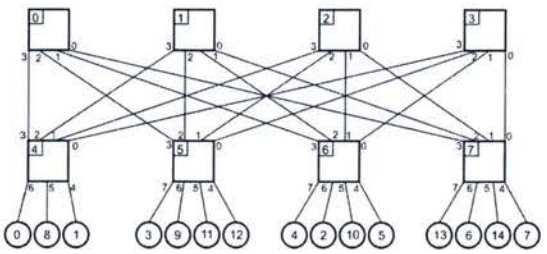

Figure 6.14: MWD Application Fat Tree Topology

The simulation runs were performed over periods of 40 000 cycles. Different theoretical frequencies of operation were assumed for the three applications used: the MPEG4 decoder and AV Benchmark used a frequency of 1GHz, while the MWD Application used a 500MHz frequency. These frequencies were selected based on the relative difference in communication volumes between the three applications. The parametrization options of the simulation models match the specifications listed above. In addition, the buffering capacity of each switch input port was set to 8 flits, and the buffering capacity of each NI input was set to 20 flits.

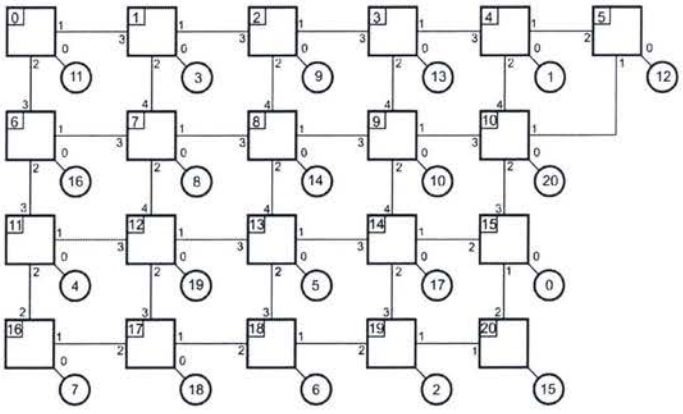

Figure 6.15: AV Benchmark Mesh Topology

The main figure of interest in the comparison is the achieved performance, which, given the characteristics of the communication protocols, is measured in total completed transactions over the duration of the simulation. Figures 6.16, 6.17 and 6.18 all show the obtained results for the sets of simulation runs. Each figure lists the initiating core on the x-axis, and the number of transactions achieved on the y-axis. The figures show interesting results, in that both the regular and irregular topologies demonstrate similar performance. There is some variation between completed transactions at different cores, but never more than a 20% difference between the best and the worst performing topology. On the face of it, it would seem there is no true difference between the two topology types.

To obtain a more complete picture, one has to look at the resource requirements of each topology. Two resources are primarily used in each network, interfaces and switches. Given that the number of cores in the system stays the same, the number of NIs in each topology will remain the same. The switch number and size, however, varies from one system to the next. Switches can be abstracted to two component types, the input FIFO and output block (this abstraction is of a high level nature, as the review chapter shows that a large number of support features can exist in the switch). Given that in the assumed network model a complete port consists of an input and output sub-port, each connected to a FIFO and

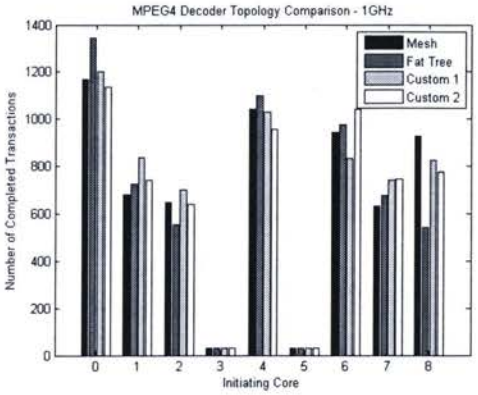

Figure 6.16: MPEG4 Decoder Transaction Results

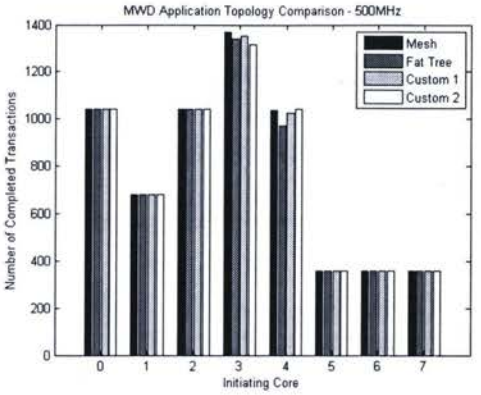

Figure 6.17: MWD Application Transaction Results

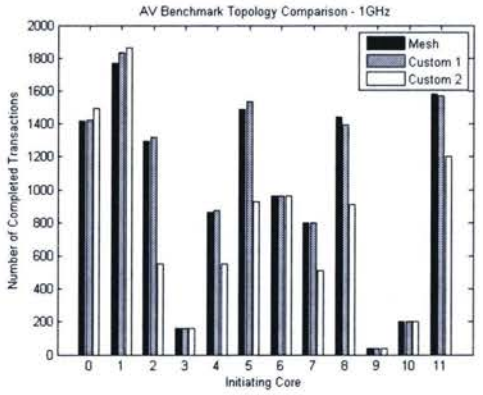

Figure 6.18: AV Benchmark Transaction Results

output block, respectively, a high level view of the resources of a topology can be obtained by simply counting the number of switch ports in the system. Such an approach to area estimation does not take into account the wiring complexity present in large switches, but it does provide an indication of the relative resource use. Table 6.1 shows the total number of switch ports for every topology and application.

In most cases, the irregular topologies require less than half the resources of the regular ones. Another way of looking at this situation is that the irregular topologies trade away aggregated bandwidth for increased area efficiency. However, the performance of all topologies

Table 6.1: Application Resource Use

Topology	MESH	FAT	CUSTOM1	CUSTOM2
MPEG4 Decoder	46	44	22	14
MWD Application	59	47	13	17
AV Benchmark	87	-	67	25

is similar, which suggests that the increased aggregated bandwidth in the regular topologies is not truly utilized. The one exception is the AV Benchmark, where the Partitioned Crossbar topology clearly performs less well than the Point-to-Point or Mesh topologies, both of which use more network resources. Here, the added bandwidth associated with more ports in the system is utilized, because the communication patterns are much less centralized than in the case of the MPEG4 Decoder, for example.

An interesting point worth examining in more detail was the fact that, from a performance point of view, the irregular topologies performed similarly to the regular ones, despite minimizing the latency on their communication paths. A possible reason for this behavior is the use of burst transactions which, as in traditional interconnects, attempt to distribute and hide the transaction latency amongst multiple transfers. To determine if this is the case, a special test was performed. The MPEG4 Decoder transaction patterns were altered so that only Core 6 (the Up-Sampler) generated any transactions; all other generation rates were set to zero. Three tests were performed, where the transaction burst length for Core 6 was set to 1, 4 and 16 beats. The tests were performed using the Partitioned Crossbar topology and the Mesh; the mesh has a minimum of three hops between any source and destination, while the custom topology only has two, in the case of Core 6. Finally, to accentuate the effect of additional hop delay, the arbitration latency was set to 20 cycles. Figure 6.19 shows the attained number of transaction for the three burst settings, and Figure 6.20 shows the worst-case latencies. The figure shows a clear performance increase for the custom topology using 1-beat burst. However, the figure also shows that using longer bursts is beneficial from

a performance point of view, as the number of obtained transactions almost doubles when going from 1 to 16 beats. Also, the performance difference drops off at higher burst settings, showing that, indeed, the use of bursts can hide the latency in a network.

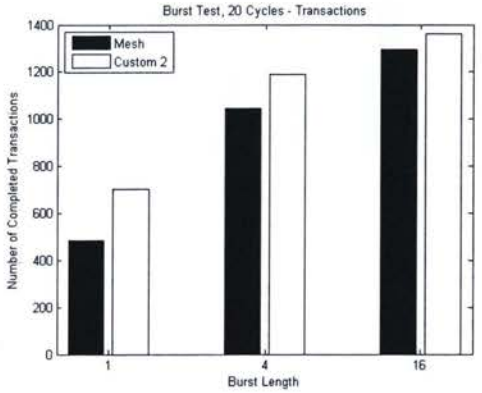

Figure 6.19: MPEG4 Decoder Burst Test 1 - Transactions

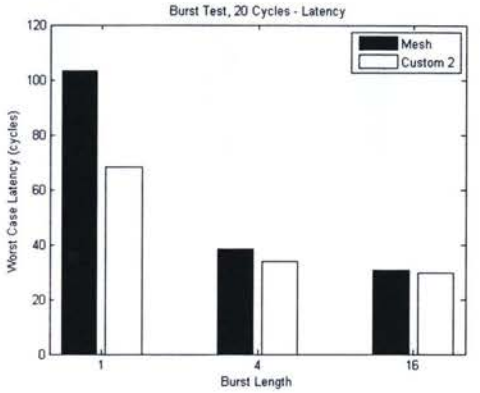

Figure 6.20: MPEG4 Decoder Burst Test 1 - Latency

Finally, the above test was repeated again, but this time the arbitration latency was set to 8 cycles. Figure 6.21 and 6.22 show the obtained transactions and latencies for this case. This time, the performance difference is less pronounced, even for 1-beat bursts. The reason for this is that, as the arbitration latency decreases, the packing and de-packing latency becomes more pronounced, and can dominate the overall latency in certain situations. In such cases, the performance increase obtained from using the application-specific topologies will be negligible and the main benefit of the custom topologies will be the decreased resource use.

6.3 Predictor Accuracy

The analysis method proposed in this document must also be tested, to determine its characteristics and limitations, especially given the fact that the prediction method used trades some accuracy for performance. This results section determines if the predicted operating

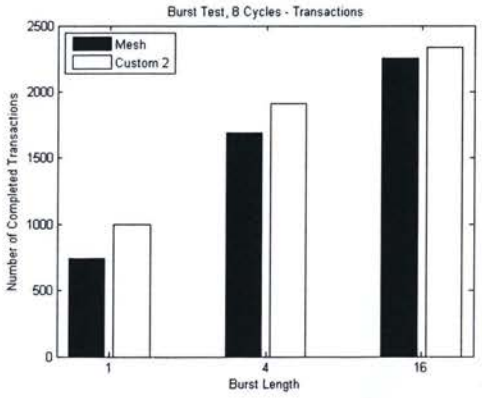

Figure 6.21: MPEG4 Decoder Burst Test 2 - Transactions

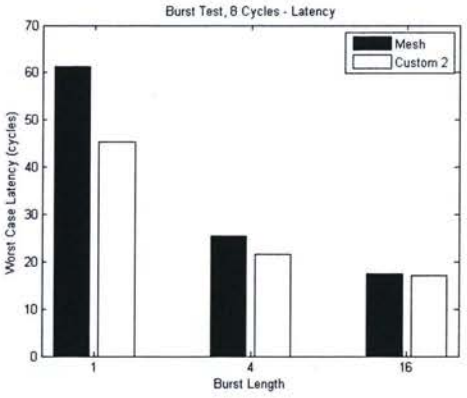

Figure 6.22: MPEG4 Decoder Burst Test 2 - Latency

frequencies for the four applications allow the generated topologies to meet the specified communication requirements. The first set of tests are performed using Poisson Transaction generation, to determine the predictor accuracy in such situations; for these tests, the MPEG4 Decoder, MWD Application and AV Benchmark are used. The second test is performed using the Layer-3 Switch, and fully specified temporal patterns are used. This second test is performed to determine how badly specific access patterns deviate from the uniform injection model assumed in the analysis engine.

6.3.1 Poisson Transaction Patterns

The MPEG4 Decoder, MWD Application and AV Benchmark were used to test the accuracy of the topology analyzer in the presence of Poisson traffic. For each application, the analysis engine computed a specific recommended frequency of operation, based on the information flow in the application and the specific network characteristics (such as arbitration, packing and de-packing delays). The average generation rate of each core, $1/\alpha$ was set so that each core tries to generate a number of transactions over the length of the simulation run. The total number of transactions is derived from the core graph, and the simulation time is

computed based on the number of simulation cycles and the specified frequency of operation.

Before actually examining the simulations a note must be made regarding the analysis engine. The engine computes a theoretical frequency of operation recommended for proper operation; this frequency may not be realistically feasible currently (due to technological considerations). In such instances, an alternative solution would have to be found, which would reduce the transaction latency further, or pursue much wider communication links, to allow more information to be transmitted at the same time. The primary aim of this section is to determine if the prediction method is accurate, rather than to make comments regarding implementation feasibility. The situation described here only occurs in cases where very large information requirements (in excess of 1GByte/s) occurs in the specified application core graph.

For the topologies presented in Section 6.2, the following NoC clock frequencies were computed: 3438 MHz for the MPEG4 Decoder, 573.4MHz for the MWD Application and 2310 MHz for the AV Benchmark. Figures 6.23, 6.24 and 6.25 show the obtained number of transactions over the duration of the simulation (40000 cycles, as before). In these figures, however, the first data set corresponds to the required number of transactions. A core meets its transaction requirement is it can complete this number of transactions.

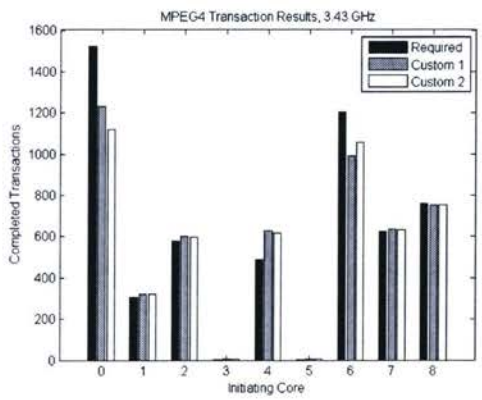

Figure 6.23: MPEG4 Decoder Analysis Accuracy Test

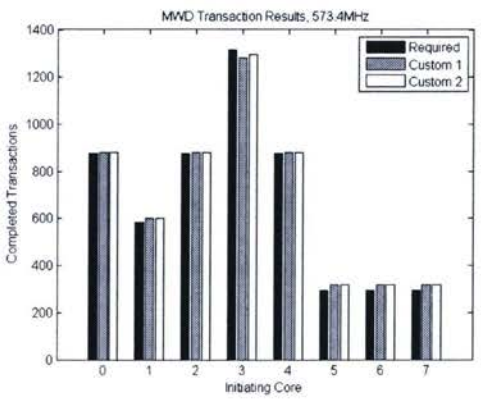

Figure 6.24: MWD Application Analysis Accuracy Test

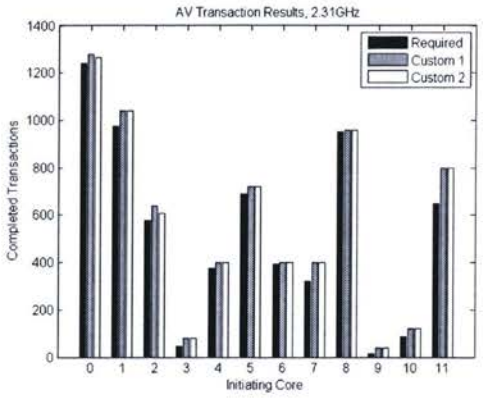

Figure 6.25: AV Benchmark Analysis Accuracy Test

In the above results, the worst case deviation is observed for Core 0 in the MPEG4 Decoder, where the achieved number of transactions is missed by 27%. In the remainder of results, the requirements are met fully, or deviate by less than 20%. Only Cores 0, 6 and 8 of the MPEG4 Decoder and Core 3 of the MWD Application actually miss their transaction requirements at all, out of a total of 29 cores. There a multiple possible reasons for the observed deviation. One reason is the fact that transaction packets are modelled as single units during analysis, while in reality each is a collection of flits. As well, the analysis method analyzes switches in isolation and then sums the effects across a path, which may neglect some of the more subtle effects present in the system, in particular where contention is concerned. Finally, for each switch output, it is assumed that transactions are equally likely to target any one of its time-slots, which may not be the case at all times (burst transfers group packets temporally and can cause increased contention at certain points). Nonetheless, these results show that the proposed analysis method can predict the behavior of an application with some accuracy, and without an over-large expenditure of resources.

A worst-case deviation of 27% might be considered un-acceptable, were it not for two facts. The fist is that this deviation is encountered in only one out of 29 cores. More importantly, however, is the fact that the proposed method is a high-level system synthesis

tool. As such, any generated systems will have to be further refined before and during actual implementation (layout). The frequency of operation of a component is affected by a number of issues in the implementation stage, including the complexity of the implemented logic circuitry (number of delay stages) as well as the length of wires in the system. Because of all this, the analysis method proposed here is meant primarily as a first approximation process which will then be further refined during physical implementation. Having said that, it is, of course, beneficial to be as accurate as possible even at this stage, which is why these tests were performed.

6.3.2 Specified Transaction Patterns

A second accuracy test was performed using the Layer-3 switch application. In this case, the transaction temporal patterns in the system were fully specified, to mimic the behavior that would occur in real systems. The same parameters were used when generating the Layer-3 Switch topology as were used for the other three applications. Figure 6.26 shows the generated topology. Only one topology is shown because, interestingly, both algorithms generated the same final network; such situations can occur, depending on the application core graph.

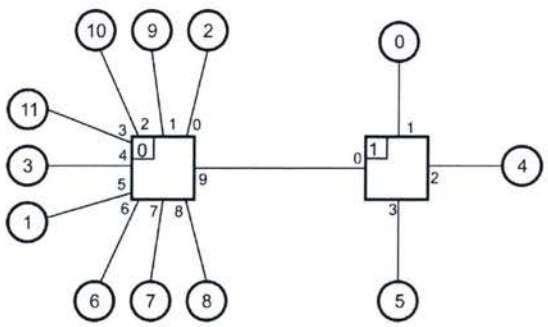

Figure 6.26: Layer-3 Switch Custom Topology

In this test, the patterns were specified completely. Figure 6.27 shows the patterns for

the three traffic generators, Cores 0, 1 and 2 (the CPU, DMA1 master interface and DMA2 Master interface, respectively). Note that the figure is not drawn to scale, but rather is meant to give a graphical representation of the patterns. For each transaction generator, the pattern shown is repeated continuously for the duration of the simulation run, with the specified wait interval between each activity phase. Each rectangular segment specifies an acces; the first number specifies the destination core, the second the burst length, and the letter specifies (r)ead or (w)rite transactions. The patterns shown are meant to represent a specific type of operation in the switch, a backup process over the network. Five of the six ports in the system are receiving 100 MBit/s streams of information headed for the sixth port. Periodically, the Processor sends control information to the DMA slave interfaces. As well, the processor performs monitoring operations, which generate traffic to the system memory.

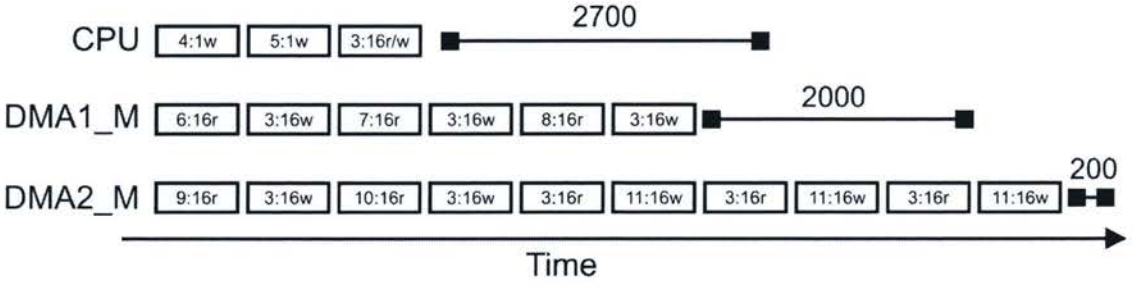

Figure 6.27: Layer-3 Switch Transaction Pattern

The simulation was, once again, performed over 40000 cycles. The predicted frequency of operation for this system was 229 MHz. Figure 6.28 shows the obtained results. As before, the first data set corresponds with the required number of operations, and the second corresponds to the actual obtained results. The figure shows that even when using specific injection patterns, rather than Poisson Event modelling, the predicted frequency of operation is still valid. The worst-case deviation observed is less than 5% in Core 3. Once again, the same reasons listed for the first set of accuracy tests can account for the discrepancy

between the required and actually completed numbers of transactions. These results show that specific patterns of traffic injection tend to be distorted by the network, to the point that they approach a more uniform temporal distribution. This effect can be attributed to the delay present in the various network components, as well as the interference from other transactions.

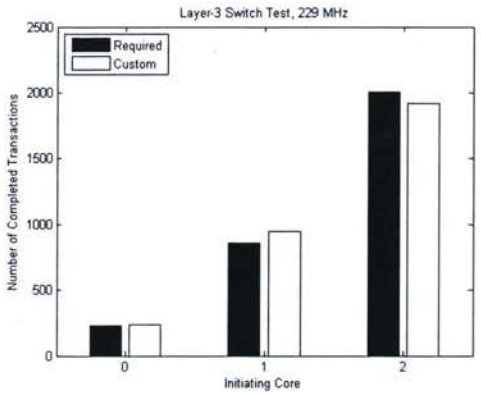

Figure 6.28: Layer-3 Switch Accuracy Test

6.4 Program Execution Time

Section 4.3.4 discusses the complexity of the two topology generation algorithms. However, the complexity of the topology analysis process was not analyzed, as it depends on a large number of parameters, both in the system input as well as derived data structures (most specifically the topology graph). This section attempts to mitigate this omission by presenting the execution time of the topology generation and analysis process. The program was run on a computer with a 2.4 GHz dual-core processor and 2GB or RAM, running the RedHat Fedora operating system. Table 6.2 below lists the execution times of generating and analyzing topologies for all applications described above.

The first thing worth noting is that none of the topologies took more than a minute

Table 6.2: Program Execution Time

Application	Custom 1	Custom 2
MPEG4 Decoder	22.817s	7.066s
MWD Application	12.089s	2.016s
AV Benchmark	35.606s	8.165s
Layer-3 Switch	4.484s	0.932s

to be generated and analyzed. In addition, and not unexpectedly, the partitioned crossbar topologies are generated quicker than the point-to-point topologies. The values are consistent, in that the applications with small communication requirements take less time to be analyzed than those with high requirements; this result is due in large part to the token-based simulation method employed in the analysis method. The point-to-point topologies further support the supposition that the complexity of the analysis method is based in part on the communication requirement of an application. However, these execution times also incorporate the iterative nature of the point-to-point algorithm. Nonetheless, the worst-case analysis and generation time was less than 40s, meaning that for a given application both topology types can be obtained in a very short time.

6.5 Topology Comparison

This section very briefly compares a topology generated by the proposed method with a topology generated in a method similar to that of [35]. The topology in question is listed in [36] and is for the MPEG4 Decoder already described above. Figure 6.29 lists the two topologies: a) represents the topology proposed here, based on the partitioned-crossbar method, and b) represents the topology presented in [36].

Before any analysis, it should be noted that the above diagram simply lists the connections among devices. It does not take into consideration their physical layout of the final

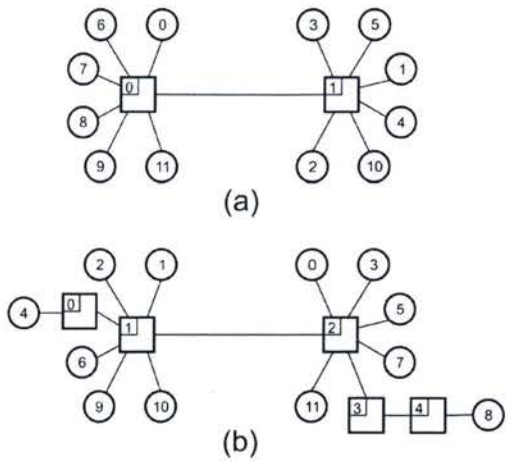

Figure 6.29: Topology Comparison: (a) Partitioned Crossbar Topology; (b) Topology Generated by Srinivasan et al. Method [36]

design. The first thing to note is the fact that the topologies are quite similar in structure, but that topology of Fig. 6.29(b) has three additional, 2-port switches. This is due to the fact that the method proposed by Srinivasan et al. begins with an approximation of the physical layout of the components, and adds switches to all corners of the component [36]. A subset of these switches is then used to connect the components, according to the application core graph. While these switches add latency to those communication paths, they also allow for higher operating clock frequencies in the NoC by eliminating long lines.

The second thing worth noting is the arrangement of the various components around the two large switches. For our partitioned crossbar topology of Fig. 6.29(a), cores 6, 9 and 11 are grouped around the same switch, since core 6 has the highest communication requirement in the system, with cores 9 and 11. Similarly, core 7 communicate only with cores 9 and 11 and is therefore also connected to the same switch. Finally, cores 0 and 8 have a large communication requirement to core 11, which is why it is connected to the same switch. It should be noted that none of the traffic originating at any of these master cores (0, 6, 7 and 8) has to travel more than two hops. The remaining cores are connected to the second

network switch, as they all have reduced communication needs compared to the above cores. In contrast, in Fig. 6.29(b) the cores are grouped based on layout first, meaning that their relative size and arrangement will dictate the topology. In this case, transactions from core 6 will have to undergo increased clock latencies when addressed to core 11, which can lead to decreased performance. In addition, the topology of Fig. 6.29(a) incorporates considerations of the data width and burst behavior of the cores, which is why core 0 is connected to switch 0, rather than core 4. Core 4 has larger requirements to core 9, but also uses larger data transfers (data-width), meaning that fewer transactions are issued over-all. In contrast, the method presented in [36] can lead to better performance at the fabrication phase, by allowing higher network clock frequencies to be achieved. Nonetheless, the transaction-based nature of the components in question, as well as the data flow characteristics will still have to be addressed. This discussion demonstrates the difficulty in making such design decisions. Both topology generation methods have strengths and weaknesses, in that the topologies proposed in this paper do not consider the final layout of the system, while the method proposed in [36] does not take into full consideration the communication characteristics of the cores and concentrates primarily on power and area issues. A fully integrated top-down design approach would first address performance requirements at a higher design level (component level) and then follow-up with further refinements at the final, physical level.

6.6 Message Passing Communication Model in Transaction Based Environments

This final section examines some of the problems that can occur when communication abstractions are used inappropriately. Specifically, this final test looks at the effects of using message passing communication models in situations where transaction-based protocols are

used (many embedded systems fall in this category). The MPEG4 Decoder was used as the basis for this test, mapped to the partitioned crossbar topology. The experiment consist of using the exact same simulation setup as that seen in Section 6.2, but using an operating frequency computed based on traditional methods used in parallel computing and large area networks. Specifically, the network must operate at such a speed that each component is capable of transmitting the worst case information volume in the application. In the case of the MPEG4 Decoder, using the partitioned crossbar topology, the worst case situation occurs at the output connected to the SDRAM unit (Core 9). This output must be able to accommodate an information volume of 1800 MB/s. The majority of this volume (1500 MB/s) is transmitted using 128-bit transfers, so this flow is converted to a number of 128-bit transfers, where each transfer is assumed to take 20 cycles to process; the simulation models used actually takes only 16 cycles per transaction, but the network will be over-designed in this case.

Figure 6.30 shows the obtained simulation results at the required operating frequency, 2360 MHz. This Frequency was obtained by multiplying the number of transfers that each output port must be able to forward by the latency of each such transfer. The results show that this frequency of operation is inadequate for the largest transaction generators, because the employed model is appropriate for write operations only. The added latency of read operations is neglected when computing a frequency of operation this way. In total, 5 out of the 9 traffic generators miss their communication requirement. Interestingly, the opposite may also happen, where this method of analysis will yield an operating frequency higher than what is actually required. In the case of the MWD Application, the described analysis method would yield an operating frequency of 630 MHz. However, Section 6.3 shows that, using 573 MHz, the obtained performance is within 3% of the requirement. In such situations, the fact that transactions do not necessarily overlap in time means that the frequency of operation can be reduced and the system requirements will still be met.

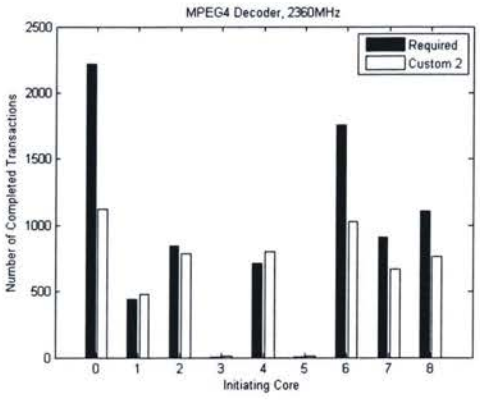

Figure 6.30: MPEG4 Decoder Traditional Design Results

6.7 Conclusion

The simulation and results chapter of a thesis is perhaps the most important part of any such document as it validates the methods being proposed therein. The results presented here address both the comparative performance of the proposed method in relation to traditional regular topologies, as well as the accuracy of the analysis method being used to determine the maximum operating frequency. The presented results show that the proposed method can provide equal or better performance when compared with regular topologies, depending on the communication specifications, while at the same time utilizing less resources. As well, the analysis method is capable of predicting the required frequency of operation to within 27% (in the worst case) despite requiring little computation time.

Chapter 7

Thesis Conclusion

The preceding document has addressed the issue of Network-on-Chip topology generation and analysis in transaction-oriented environments. The main motivation and contribution of the document is the analysis of on-chip communication protocols, and the generation of viable NoC topologies for such systems. The thesis defines formal criteria for meeting throughput requirements in transaction-oriented systems, and presents two methods of topology generation that cater specifically to transaction-oriented systems. In addition, a predictive form of network analysis is presented, aimed at estimating the required clock frequency of a given network. Such an estimation is required, since the operating frequency of the interconnect is one of its main operational characteristics, and determines what performance the network can provide.

The experiments conducted to test the method show that the generated topologies can provide equal or better performance when compared with traditional regular topologies, while using, on average, half the network resources of their regular counterparts. Additionally, the predictive analysis method was able to predict the required frequency of operation with a fair degree of accuracy, with performance deviations in individual cores not exceeding 27% in the worst case, and staying below 5% in most cases. Used as a first approximation

method for initial frequency estimation, the obtained accuracy is considered adequate. This is particularly true when one considers that the topology generation and analysis results are obtained in less than one minute of processing time.

The proposed methods are primarily high-level synthesis solutions, meaning that before they can be used, more work needs to be done at the physical implementation level. This includes the generation of physical layouts (generally based on standard-cell synthesis), analysis of layout constraints and verification of physical parasitical effects. Tools already exist to address these problems, and, in the future, can be further integrated with high-level analysis and synthesis methods, like those proposed here. For this reason, the primary areas of expansion for the proposed method would be integration with logic design tool-chains to allow automated generation of chip layouts. In addition, the topology generation process could be further expanded with the incorporation of more elaborate heuristic methods, or the augmentation of the current design process with some random search procedures which may further improve the generated results.

Publications

The work presented here has been accepted for publication as a Regular Paper in the IEEE Transactions on VLSI Systems, 2008.

Chapter 8

Appendix

This section lists all the data presented in Chapter 6 in tabular format. All listed values represent Completed transactions except where stated otherwise.

Table 8.1: MPEG4 Decoder Transaction Results - Topology Comparison - 1GHz

Core	Mesh	Fat Tree	Custom 1	Custom 2
0	1168	1344	1200	1136
1	680	728	836	744
2	648	556	704	640
3	32	32	32	32
4	1040	1100	1028	956
5	32	32	32	32
6	944	976	832	1040
7	632	676	744	748
8	928	544	824	776

Table 8.2: MWD Application Transaction Results - Topology Comparison - 500MHz

Core	Mesh	Fat Tree	Custom 1	Custom 2
0	1040	1040	1040	1040
1	680	680	680	680
2	1040	1040	1040	1040
3	1368	1340	1352	1316
4	1036	972	1024	1040
5	360	360	360	360
6	360	360	360	360
7	360	360	360	360

Table 8.3: AV Benchmark Transaction Results - Topology Comparison - 1GHz

Core	Mesh	Custom 1	Custom 2
0	1416	1424	1496
1	1768	1832	1864
2	1296	1320	552
3	160	160	160
4	864	872	552
5	1488	1536	928
6	960	960	960
7	800	800	512
8	1440	1392	912
9	40	40	40
10	200	200	200
11	1584	1568	1200

Table 8.4: MPEG4 Decoder Transaction Results - Accuracy Test - 3.437GHz

Core	Required	Custom 1	Custom 2
0	1525	1233	1120
1	305	320	320
2	580	600	596
3	2	4	4
4	488	628	616
5	2	4	4
6	1205	992	1056
7	625	636	632
8	763	752	752

Table 8.5: MWD Application Transaction Results - Accuracy Test - 573MHz

Core	Required	Custom 1	Custom 2
0	878	880	880
1	585	600	600
2	878	880	880
3	1317	1284	1296
4	878	880	880
5	293	320	320
6	293	320	320
7	293	320	320

Table 8.6: AV Benchmark Transaction Results - Accuracy Test - 2.31GHz

Core	Required	Custom 1	Custom 2
0	1243	1280	1264
1	976	1040	1040
2	578	640	608
3	45	80	80
4	378	400	400
5	690	720	720
6	392	400	400
7	322	400	400
8	953	960	960
9	17	40	40
10	86	120	120
11	648	800	800

Table 8.7: Layer-3 Switch Transaction Results - Accuracy Test - 229MHz

Core	Required	Custom 1 & 2
0	232	234
1	860	944
2	2005	1920

Table 8.8: MPEG4 Decoder Burst Test - Transactions

Burst Length	Mesh - 8cyc	Custom 2 - 8cyc	Mesh - 20cyc	Custom 2 - 20cyc
1	745	998	485	703
4	1692	1912	1044	1188
16	2256	2336	1296	1360

Table 8.9: MPEG4 Decoder Burst Test - Latency (cycles)

Burst Length	Mesh - 8cyc	Custom 2 - 8cyc	Mesh - 20cyc	Custom 2 - 20cyc
1	61.38	45.43	103.3	68.16
4	25.57	21.58	38.58	33.74
16	17.47	17.06	30.67	29.58

Bibliography

[1] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *IEEE Computer*, vol. 35, pp. 70–78, Jan. 2002.

[2] A. Allan, D. Edenfeld, J. Joyner, W.H., A. Kahng, M. Rodgers, and Y. Zorian, "2001 technology roadmap for semiconductors," *IEEE Computer*, vol. 35, pp. 42–53, Jan. 2002.

[3] "http://www.samsung.com."

[4] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, pp. 71–121, March 2006.

[5] G. D. Micheli and L. Benini, *Networks on Chips: Technology and Tools*. Morgan Kaufmann Publishers, 2006.

[6] ARM-Limited, "Multi-layer ahb overview," May 2004.

[7] ALTERA-Corporation, "Avalon interface specification," April 2005.

[8] W. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. Jerraya, L. Gauthier, and M. Diaz-Nava, "Multiprocessor soc platforms: A component-based design approach," *IEEE Design & Test of Computers*, vol. 19, pp. 52–63, Nov.-Dec. 2002.

[9] ARM-Limited, "Arm1136jf-s and arm1136j-s - technical reference manual," July 2007.

[10] ALTERA-Corporation, "Nios ii processor reference handbook," October 2005.

[11] A. Jerraya and W. Wolf, "Hardware/software interface codesign for embedded systems," *IEEE Computer*, vol. 38, pp. 63–69, Feb. 2005.

[12] ARM-Limited, "Amba 3 ahb-lite protocol," June 2006.

[13] "http://www.ocpip.org."

[14] ARM-Limited, "Amba axi protocol," March 2004.

[15] S. Schliecker, M. Ivers, and R. Ernst, "Memory access patterns for the analysis of mpsocs," *IEEE North-East Workshop on Circuits and Systems, Gatineau, Qubec, Canada*, pp. 249 – 252, 2006.

[16] F. Angiolini, P. Meloni, S. Carta, L. Raffo, and L. Benini, "A layout-aware analysis of networks-on-chip and traditional interconnects for mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, pp. 421–434, March 2007.

[17] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas, "Overview of the blue gene/l system architecture," *IBM Journal of Research and Development*, vol. 49, pp. 195–212, March/May 2005.

[18] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks - An Engineering Approach.* Morgan Kaufmann Publishers, 2003.

[19] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits and Systems Magazine*, vol. 4, pp. 18–31, Second Quarter 2004.

[20] K. Goossens, J. Dielissen, and A. Radulescu, "Aethereal network on chip: Concepts, architectures, and implementations," *IEEE Design and Test of Computers*, vol. 22, pp. 414–421, Sept.-Oct. 2005.

[21] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," *Proceedings of the IEEE Computer Society Annual Symposium on VLSI, Pittsburg, PA, USA*, pp. 117–124, 2002.

[22] D. Wiklund and D. Liu, "Socbus: Switched network on chip for hard real time embedded systems," *Proceedings of the International Parallel and Distributed Processing Symposium, Nice, France*, p. 8, 2003.

[23] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Paris, France*, pp. 250–256, 2000.

[24] F. Karim, A. Nguyen, and S. Dey, "An interconnect architecture for networking systems on chips," *IEEE Micro*, vol. 22, pp. 36–45, Sept.-Oct. 2002.

[25] T. Bjerregaard and J. Sparso, "Implementation of guaranteed services in the mango clockless network-on-chip," *IEE Proceedings-Computers and Digital Techniques*, vol. 153, pp. 217–229, July 2006.

[26] "http://www.arteris.com/index.htm."

[27] S. Murali and G. De Micheli, "Sunmap: A tool for automatic topology selection and generation for nocs," *Proceedings of the Design Automation Conference, San Diego, CA, USA*, pp. 914–919, 2004.

[28] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Paris, France*, pp. 896–901, 2004.

[29] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "Xpipescompiler: A tool for instantiating application specific networks on chip," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Paris, France*, pp. 884–889, 2004.

[30] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," *Proceedings of the IEEE/ACM International Conference on Computer Aided Design, San Jose, CA, USA*, pp. 355–362, 2006.

[31] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based noc architectures under performance constraints," *Proceedings of the Asia and South Pacific Design Automation Conference, Kitakyushu, Japan*, pp. 233–239, 2003.

[32] J. Hu and R. Marculescu, "Communication and task scheduling of application-specific networks-on-chip," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, pp. 643–651, Sept. 2005.

[33] U. Ogras and R. Marculescu, "It's a small world after all: Noc performance optimization via long-range link insertion," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, pp. 693–706, July 2006.

[34] J. Hu, U. Ogras, and R. Marculescu, "System-level buffer allocation for application-specific networks-on-chip router design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 2919–2933, Dec. 2006.

[35] K. Srinivasan, K. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, pp. 407–420, April 2006.

[36] K. Srinivasan, K. S. Chatha, and G. Konjevod, "An automated technique for topology and route generation of application specific on-chip interconnection networks," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, USA*, pp. 231–237, 2005.

[37] K. Srinivasan, K. Chatha, and G. Konjevod, "Application specific network-on-chip design with guaranteed quality approximation algorithms," *Proceedings of the Asia and South Pacific Design Automation Conference, Yokohama, Japan*, pp. 184–190, 2007.

[38] K. Srinivasan and K. Chatha, "Saga: Synthesis technique for guaranteed throughput noc architectures," *Proceedings of the Asia and South Pacific Design Automation Conference, Shanghai, China*, vol. 1, pp. 489–494, 2005.

[39] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "A design methodology for application-specific networks-on-chip," *ACM Transactions on Embedded Computing Systems*, vol. 5, pp. 263–280, May 2006.

[40] W. N. Ho and T. M. Pinkston, "A design methodology for efficient application-specific on-chip interconnects," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, pp. 174–189, Feb. 2006.

[41] Y.-L. Jeang, J.-M. Jou, and W.-H. Huang, "A binary tree based methodology for designing an application specific network-on-chip (asnoc)," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E88-A, pp. 3531–3538, Dec. 2005.

[42] G. Ascia, V. Catania, and M. Palesi, "An evolutionary approach to network-on-chip mapping problem," *Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Edinbourgh, Scotland*, vol. 1, pp. 112–119, 2005.

[43] W. Zhou, Y. Zhang, and Z. Mao, "An aapplication specific noc mapping for optimized delay," *Proceedings of the 2006 International Conference on Design and Test of Integrated Systems in Nanoscale Technology, Tunis, Tunisia*, pp. 184–188, 2006.

[44] L. Papadopoulos, S. Mamagkakis, F. Catthoor, and D. Soudris, "Application - specific noc platform design based on system level optimization," *Proceedings of the IEEE Computer Society Annual Symposium on VLSI, Porto Alegre, Brazil*, pp. 311–316, 2007.

[45] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The nostrum backbone-a communication protocol stack for networks on chip," *Proceedings of the International Conference on VLSI Design, Mysore, India*, pp. 693–696, 2004.

[46] T. Ye, L. Benini, and G. De Micheli, "Packetized on-chip interconnect communication analysis for mpsoc," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Munich, Germany*, pp. 344–349, 2003.

[47] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Nice, France*, pp. 1096–1101, 2007.

[48] E. van der Tol and E. Jaspers, "Mapping of mpeg-4 decoding on a flexible architecture platform," *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 4674, pp. 1–13, 2002.

[49] E. Jaspers and P. de With, "Chip-set for video display of multimedia information," *IEEE Transactions on Consumer Electronics*, vol. 45, pp. 706–715, Aug. 1999.

[50] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE Transactions on Computers*, vol. 54, pp. 1025–1040, Aug. 2005.