SOC INTERCONNECT ARCHITECTURE DESIGN AND EVALUATION UNDER TIMING CONSTRAINTS

by

Cristian Grecu

B.A.Sc., Technical University of Iasi, Romania, 1996

M. Eng, Technical University of Iasi, Romania, 1997

A thesis submitted in partial fulfillment of the requirements for
the degree of
Master of Applied Science
in
The Faculty of Graduate Studies
Department of Electrical and Computer Engineering
We accept this thesis as conforming to the required standard:

The University of British Columbia

December 2003

© Cristian Grecu, 2003

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

CRISTIAN GRECU — 19/12/2003

Name of Author *(please print)* — Date (dd/mm/yyyy)

Title of Thesis: SoC INTERCONNECT ARCHITECTURE DESIGN AND EVALUATION UNDER TIMING CONSTRAINTS

Degree: M.A.Sc. Year: ~~2003~~ 2004

Department of ELECTRICAL AND COMPUTER ENGINEERING

The University of British Columbia
Vancouver, BC Canada

SOC INTERCONNECT ARCHITECTURE EVALUATION

UNDER TIMING CONSTRAINTS

ABSTRACT

System on chip design steadily evolves toward different non-overlapping abstraction levels. Very different competence and design tools will be needed at each level. One specific level of abstraction will deal with interconnect technologies, with a pronounced trend towards networks on chip.

It is projected that, within five years, the large majority of end-user SoC products will consist of heterogeneous embedded processors, built on multi-processor SoC platforms (MP-SoC). There is a tremendous amount of research required to characterize the various topologies and their effectiveness for different application domains.

A common issue with all network-on-chip topologies is communication latency. Due to the increase of global wire delay with technology scaling, pipelining is required to hide the latency associated with the exchange of data across the chip.

The building blocks of a network-on-chip are intelligent switches, which provide a data transport mechanism across the chip. Their design is critical due to different architectural and circuit level trade-offs.

This work is novel in that it addresses the issues of quantifying the delay of different pipeline stages in an on-chip topology, and evaluates the effectiveness of a given topology in forthcoming technology nodes.

TABLE OF CONTENTS

CHAPTER V

CHAPTER VI

LIST OF FIGURES

LIST OF TABLES

ACKNOLEDGEMENTS

First of all, I wish to thank my academic advisor, Dr. André Ivanov, for the guidance, technical advice and moral support that he provided throughout my Masters.

Also, I would like to thank Dr. Res Saleh for his help and critical feedback.

I greatly appreciate the financial support provided by the Micronet R & D and Gennum Corporation. Without their support, this work would not be possible.

I would also like to thank to all members of Dr. Ivanov's research group for their helpful insights and for creating a friendly research environment. Special thanks are due to Partha Pande, for his help throughout the project and the time spent in long and fruitful discussions.

Over these years, it was a great pleasure to work in the SoC Research Group at UBC, and I take the opportunity to thank all the people in the group for creating an excellent, motivating atmosphere.

Finally, I would especially like to thank Gabriela for support and encouragement throughout my academic years at UBC.

1 Chapter I

Introduction

Developments in semiconductor technology have led to a point where the integration of tens or hundreds of different IP (***Intellectual Property***) blocks on a single chip is possible. One of the main challenges in integrating such systems is to provide a reliable, high performance on-chip data transport mechanism. In general data exchange among these modules is performed through so called "global wires", whose main characteristic is that they do not scale with technology improvement. Currently, designers have a choice of using ad hoc (point to point) interconnects or structured interconnects in the form of buses when designing large systems. Both these types of interconnects consist of global wires and exhibit the disadvantage of non-scalability.

Buses and ad hoc interconnects have another major drawback, i.e., their length is not predictable at the early stages of the design flow. As a consequence, it is difficult to estimate whether a given design will meet the initial performance requirements. In fact, designers spend a large part of their time running and optimizing logic-synthesis and physical-implementation (place-and-route) tasks. *The name given to this process, timing closure, refers to the application of EDA (Electronic Design Automation) tools and design techniques to meet RTL (Register Transfer Level) chip-timing specifications.* Unfortunately, global timing closure can be achieved today only after numerous iterations. There is no definite procedure to achieve timing closure and performance requirements while using non-structured interconnects.

Recently, the use of highly structured interconnect topologies was proposed to cope with the above-mentioned issues. Most of these topologies come from the parallel processing world and there is an impressive amount of research going on in the attempt of mapping them on silicon. This thesis focuses on the characterization of such a topology with respect to timing and on placing it in the perspective of technology trends.

1.1 Research Goals

There are a few different networks on chip that have been proposed by different research groups [1][2]. The general claim is that by using a highly structured interconnect scheme, it is possible to avoid the problems related to global wires and achieve timing closure while reducing the design time and meeting performance requirements. Signal transmission between any of the system components is highly pipelined, so, intuitively, the speed of operation of such an interconnect will be fast, at the expense of higher latency.

A network on-chip (NOC) is a structured, pipelined interconnect template; data will travel between one IP block to another following paths consisting of wires and intelligent switches.

There are two elements involved in the operation of a NOC: active elements, i.e., switches, and passive elements, i.e., wires between switches.

The purpose of this thesis is to analyze quantitatively the delays of the active and passive elements and to provide an insight in regards to the achievable performance in terms of minimum clock cycle. Specifically, there are performance requirements that the high performance systems on chip have to meet [3].

This work develops a method for the analysis of NOCs with respect to delays involved in data transport mechanism. By detailed circuit level design and analysis, we are able to quantify the contribution of each element of the interconnect template to the overall timing characteristic.

The primary goal is to demonstrate what are the main components of the delay, the parameters affecting the timing of a network on chip and the trade-offs that a designer has to consider when working with these concepts.

1.2 Research Approach

A detailed analysis was performed of both passive (wires) and active (switches) devices of the NOC described in [4].

In [4] we proposed the use of the butterfly fat-tree as the overall template to interconnect functional IP blocks in large SOCs. Architectural trade-offs of this topology were discussed in [5]. Here, the performance of this topology is analyzed in terms of minimum achievable clock cycle.

First, we develop a deterministic wire length model of the butterfly fat-tree. By mapping the butterfly fat-tree graph onto a square 2-dimensional area corresponding to the physical silicon die, we are able to come up with an accurate expressions for the inter-switch wires. This model was then enhanced by considering distributed RC effects and buffer insertion requirements; a projection of the wire delay model is provided by analyzing the interconnect technology trends, i.e., mainly, copper metallization and low-k dielectrics.

Next, we developed a delay model for the switches in the butterfly fat-tree network. Starting from the logical operation of the switches and implementing a virtual channel

wormhole routing strategy, we characterized each process involved in the switch operation in terms of technology independent delay units.

Finally, having the two components of the delay, namely, the inter-switch and intra-switch delays, we analyzed the achievable clock cycle of our interconnect infrastructure based on the butterfly fat-tree topology. The analysis shows that the infrastructure complies with ITRS 2001 projections for high performance SoCs with respect to the achievable clock cycle.

1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 describes succinctly a few topologies proposed by different research groups as suitable for on-chip implementation and provides a short theoretical classification of interconnect networks. It also gives an overview of the butterfly fat-tree topology and presents a simple solution for floorplaning and routing for networks-on-chip using this architecture. Chapter 3 details the design of a switch for a network-on-chip, together with the factors that governs designer's decisions. Chapter 4 provides an in-depth analysis of the interconnect-related issues: wire length modeling, resistance and capacitance effects, and buffer insertion. The the intra-switch delay calculation is given in Chapter 5. Finally, Chapter 6 summarizes the conclusions drawn throughout this thesis and provides suggestions for future work.

2 Chapter II

Related Work

A few on-chip micro network proposals for SoC integration can be found in literature. Kumar [4] and Dally [1] have proposed mesh-based interconnect architectures. These architectures consist of an *m x n* mesh of switches interconnecting computational resources (IPs) placed along with the switches. Each switch is thereby connected to four neighboring switches and one IP block. In this case, the number of switches is equal to the number of IPs.

The physical placement of such a micronetwork is reported in Fig. 1, with white squares representing the functional IPs and black squares denoting the switches.

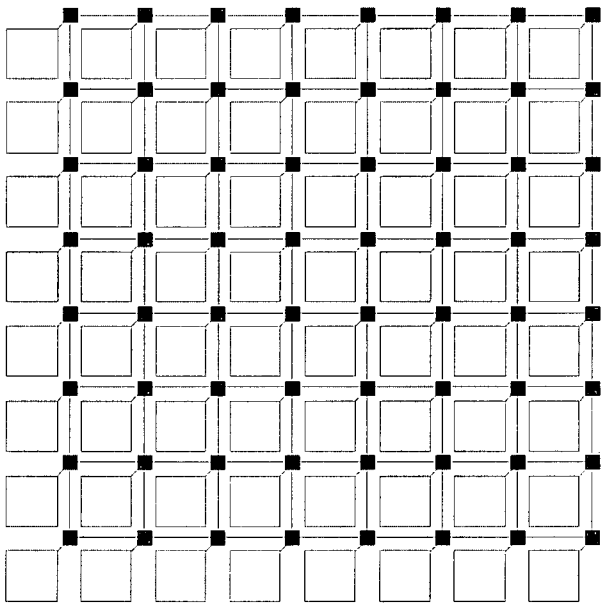

Fig. 1: Floorplan of a 64-IP-mesh network.

Guerrier and Greiner [7] have proposed a generic interconnect template for on-chip packet switched interconnections, where they have used fat-tree architecture to interconnect IP blocks. In generic fat tree architecture adding more links in parallel as switches become closer to the root switch increases transmission bandwidth between switches. As a result of this the architecture of the switches will also vary from level to level and they will not be reusable.

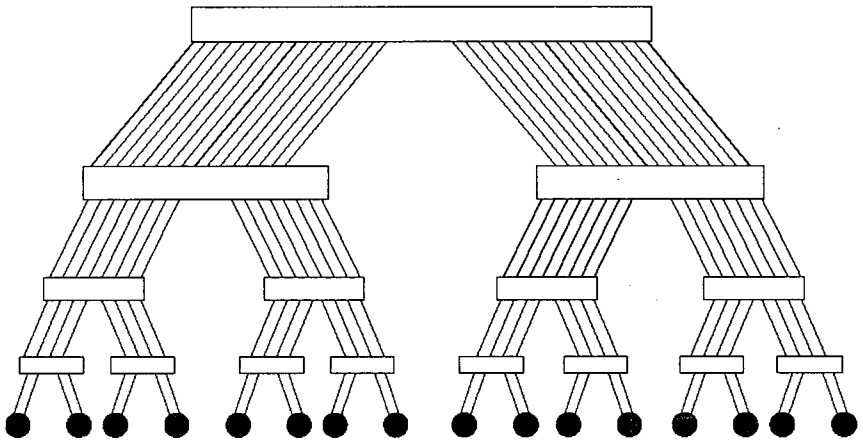

Fig. 2: Generic fat tree with 16 leaves.

The above works neither discuss the suitability of the proposed interconnect architectures in the SoC domain nor they show any comparison with other possible architectures. All of the above mentioned works proposed different types of interconnect architectures to solve the global wire delay problem; however none of them specifically deals with this.

In [4][8] we have described an interconnect architecture for a networked SoC, as well as the associated design of required switches and addressing mechanisms.

Addressing the wire delay problem and more generally analyzing the global timing closure in a communication-centric SoC is precisely the focus of this thesis.

2.1 Introduction to Interconnect Networks

Interconnection networks are currently used in many different applications, ranging from internal buses in VLSI circuits to wide area computer networks. Among others, these applications include backplane buses; telephone switches; internal networks for ATM and Internet switches; processor-memory interconnects for vector supercomputers; interconnection networks for multicomputers and distributed shared memory multiprocessors; clusters of workstations and personal computers; networks for industrial applications.

There are many factors that may affect the choice of an appropriate interconnection network. Some of the most important are the following:

1 *Performance requirements.* Processes executed by different processing elements exchange data and synchronize through the interconnection network. These operations are performed by message passing and/or by accessing shared resources. Message latency is the time elapsed between the time a message is generated at its source and the time the message is delivered at the destination. Latency negatively affects the *idle* times of the processing nodes and memory access times to remote memory locations. Also, the amount of information that a network can deliver is finite and limited. The maximum amount of information a network can physically deliver per unit time defines the *throughput* of that network.

2 *Scalability.* A scalable topology implies that as more processing elements are added, their memory bandwidth, I/O bandwidth and network throughput should increase proportionally. Otherwise, the components that do not scale may become

a bottleneck for the rest of the system, decreasing the overall performance accordingly.

3 *Simplicity.* Simple designs tend often to lead to higher clock frequencies and may achieve higher performance.

4 *Physical Constraints.* An interconnection network connects processing elements, memories and/or I/O devices. It is desirable for any network to accommodate a large number of components while maintaining low communication latency. As the number of components increases, the number of wires needed to interconnect them also increases. One major limitation in large networks is the arrangement of wires in a limited area, that is, the maximum possible wire density limits the complexity of a connection. Also the speed of such a system tends to be limited by the wire lengths. A significant amount of power is expected to be consumed to drive these wires.

5 *Cost Constraints.* It is obvious that the best possible interconnection network may be too expensive, in terms of design time and silicon area. There is always a trade-off between cost and performance.

2.2 Classification of Interconnection Networks

In order to choose an appropriate template for an on-chip interconnect network, it is useful to have a classification [9]. Known interconnection networks are categorized into four major classes based primarily on network topology:

- *shared medium networks,*

- *direct networks,*
- *indirect networks,*
- *hybrid networks.*

2.2.1 Shared-Medium Networks

The least complex interconnect structure is the one in which the transmission medium is shared by all communicating devices. Only one device is allowed to use the network at a time. Every device attached to the network has requester, driver, and receiver circuits to handle the passing of addresses and data. A unique characteristic of a shared medium is its ability to support *broadcast*, in which all devices on the medium can monitor network activities and receive the information transmitted on the shared medium. Due to limited network bandwidth, a single shared medium can only support a limited number of devices before the medium becomes a bottleneck [31]. They are known under the common name of *buses* and, in SoC environment, they are the first attempt to structure the data exchange medium. Some examples are AMBA [10], WISHBONE [11], CORECONNECT [12]. A conceptual example is given in Fig. 2.

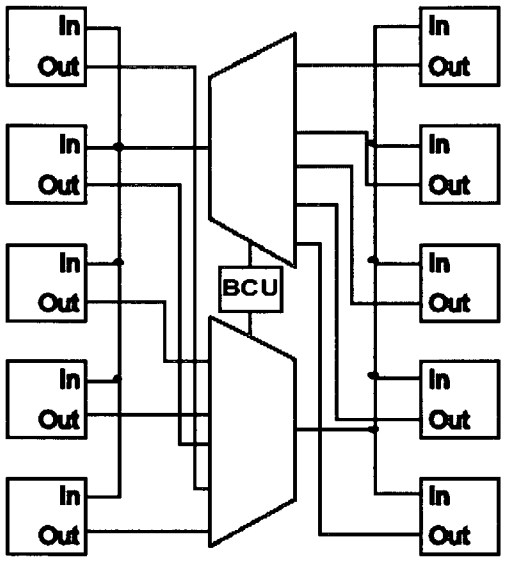

Fig. 3: Shared medium on chip interconnect.

Due to the very nature of the medium, several devices may attempt to use the bus simultaneously. To deal with this issue, a policy must be implemented to allocate the bus to the devices making such requests. Bus allocation is carried out by *arbiters*. In order to perform an access request, the initiator has to exclusively own the bus and become a *bus master*.

Most bus transactions involve request and response. After a request is issued (by the *master* device), it is desirable to have a fast response (from the *slave* device). Due to slow slaves, the bus bandwidth is wasted while waiting for a response. In order to minimize the waste of bus bandwidth, the *split transaction protocol* is being used in many bus networks. In this protocol, the bus mastership is released immediately after the request, and the slave device has to gain mastership before it can send the data. Split transaction protocol has a better bus utilization, but it requires much complicated control hardware. Buffering is needed in order to save messages before the slave device can get the bus mastership.

2.2.2 Direct Networks

Scalability is an important issue when designing SoCs with a large number of IP (Intellectual Property) blocks. Bus-based systems are not scalable because the bus becomes the bottleneck when more blocks are added. One way to address the scalability issue is to use a *direct network.* A direct network consists of a set of nodes, each one being connected to a (generally small) subset of other nodes in the network. Each node is an independent functional unit and it is connected locally to a *router*, which handles communication among nodes. For this reason, direct networks are also known as *router-based networks.* Each router has direct connections to the router of its neighbor. Usually, two neighboring nodes are connected by a pair of unidirectional channels in opposite directions. A bidirectional channel may also be used to connect two neighboring nodes. Each router supports some number of input and output channels. The channels connected to the local resource are called *internal channels*, and the channels connected to the other routers are called *external channels*. By connecting the input channels of one node to output channels of other nodes, the direct network is defined.

Direct network can be modeled by a graph *G(N,C)*, where the vertices of the graph, *N*, represent the set of processing nodes, and the edges of the graph, *C*, represent the set of communication channels. This simple model does not consider any hardware implementation issue, but it allows the study of network properties. From the graph representation, some basic network properties can be defined:

- *Node degree*: number of channels connecting a node to its neighbors;
- *Diameter*: the maximum distance between two nodes;

- *Regularity*: a network is regular when all the nodes have the same degree;
- *Symmetry*: a network is symmetric when it looks alike from every node.

A direct network is mainly characterized by three factors: topology, routing and switching. The topology defines how the nodes are interconnected by channels and can be modeled by a graph as indicated above. An ideal direct network would connect each node to all other nodes. No message would even have to pass through an intermediate node before reaching its destination. This fully connected topology requires a router with N links (including the internal link) at each node for a network with N nodes. Therefore, the cost is prohibitive for networks of moderate to large size. The engineering and scaling difficulties preclude the use of fully connected networks. As a consequence, many topologies have been proposed, trying to balance performance and cost parameters. In these topologies, messages may have to traverse some intermediate nodes before reaching the destination node.

For efficient use of network resources, a message may be divided into packets prior to transmission. A *packet* is the smallest unit of information that contains the destination address, carried in the packet *header*. For topologies in which packets may have to traverse some intermediate nodes, the *routing algorithm*, determines the path selected by a packet to reach its destination. At each intermediate node, the routing algorithm dictates the next channel to be used, which may be selected from a set of possible choices. If all the candidate channels are busy, the packet is blocked and cannot advance. Efficient routing is critical to the performance of interconnection networks. When a packet reaches an intermediate node, a *switching* mechanism determines how and when the router's input channel is connected to a certain output channel selected by the

routing algorithm. Some buffer space is required to store the packet until the next channel is reserved. If a packet is blocked, it requires some buffer space to be stored, until a free channel can be reserved.

Direct Network Topologies

Many network topologies have been proposed in terms of their graph-theoretical properties. The most known direct networks are the *n-dimensional mesh,* the *k-ary n-cube* or *torus,* and the *hypercube.*

Formally, an *n*-dimensional mesh has $k_0 \times k_1 \times \ldots k_{n-2} \times k_{n-1}$ nodes, k_i nodes along each dimension i, where $k_i \geq 2$ and $0 \leq i \leq n-1$. each node X is identified by n coordinates $(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$, where $0 \leq x_i \leq k_i - 1$ for $0 \leq i \leq n-1$. Two nodes X and Y are neighbors if and only if $y_i = x_i$ for all i, $0 \leq i \leq n-1$, except one, j, where $y_j = x_j \pm 1$. Nodes have from n to $2n$ neighbours, depending on their location; therefore the mesh is not regular.

In a *k*-ary *n*-cube, all k_i are equal to k and two nodes X and Y are neighbors if and only if $y_i = x_i$ for all $\leq i \leq n-1$, except one, j, where $y_j = (x_j \pm 1) \bmod k$. this change with respect to aforementioned mesh adds wraparound channels to the *k*-ary *n*-cube, giving it regularity and symmetry.

The hypercube is a special case of both *n*-dimensional mesh and *k*-ary *n*-cube. A hypercube is an *n*-dimensional structure in which $k_i = 2$ for $0 \leq i \leq n-1$, or a 2-ary *n*-cube, also referred to as a binary *n*-cube. For example, the network in Fig. 4(a) is a binary 4-cube.

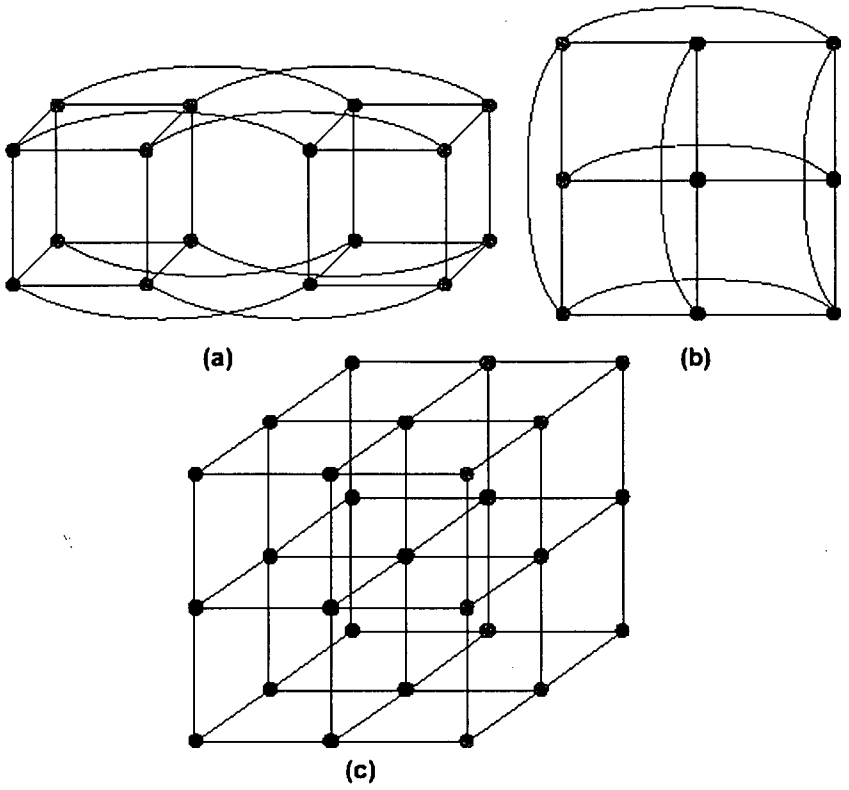

Fig. 4: (a) 2-ary 4-cube; (b) 3-ary 2-cube; (c) 3-D mesh.

Fig. 4 (a) depicts a binary 4-cube or 16-node hypercube. Fig. 4 (b) shows a 3-ary 2-cube or two-dimensional (2-D) torus. Fig. 4 (c) illustrates a 3-ary three-dimensional (3-D) mesh.

2.2.3 Indirect Networks

Indirect or *switch-based* networks are another major class of interconnection networks. Instead of providing a direct connection among some nodes, the communication between any two nodes has to be carried out through *switches*. Each switch can have a set of ports. Each port consists of one input link and one output link. The main difference between direct networks and indirect networks is that, in the case of

indirect networks, a subset of the switches is connected to one or more communicating nodes (IP block), while the rest of the switches are connected only to each other. In the case of direct networks, each router is connected to a local node *and* to other routers. In parallel processing, the terminology is *router-based networks* for direct networks, and *switch-based* networks for indirect networks. The interconnection of the switches defines various network topologies.

Indirect networks can also be modeled by a graph $G(N,C)$, where N is the set of switches and C is the set of links between the switches. Each switch in an indirect network may be connected to zero, one or more processing cores. Obviously, only switches connected to some processing core can be the source or destination of a message.

Similar to direct networks, the indirect networks are mainly characterized by three factors: topology, routing, and switching. In regular indirect network, the switches are usually identical and are organized as a set of stages. Each stage is only connected to the previous and next stage using regular connection patterns. Input/output stages are connected to the functional nodes as well as to another stage in the network. These networks are referred to as *multistage interconnection networks* (MIN) and have different properties depending on the number of stages and how those stages are arranged.

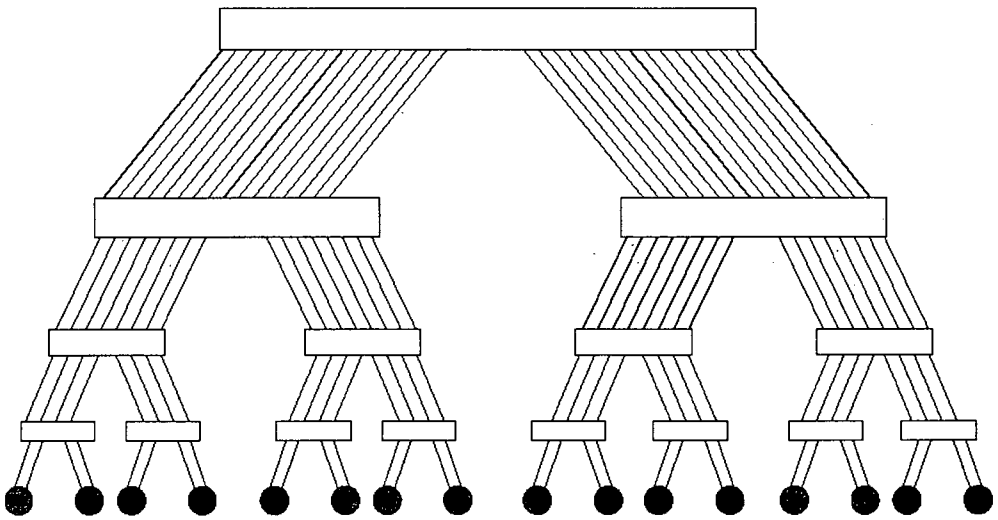

Fig. 5: MIN (fat tree) with 16 end nodes and four levels of switches.

MINs were initially proposed for telephone networks and later for array processors. MINs have been popular as alignment networks for storing and accessing arrays in parallel from memory banks. Depending on the interconnection scheme employed between two adjacent stages and the number of stages, various MINs have been proposed.

Fat-Trees

From the family of MINs, of particular interest are the *fat-trees*. Unlike traditional trees in computer science, fat-trees resemble real trees because they get thicker near the root, that is, the number of channels connecting two switches on adjacent levels grows from the leaves towards the root. Formally, fat-tress are defined as follows:

Definition 1: *A fat-tree is a collection of vertices connected by edges, constructed recursively as follows:*

- *A single vertex by itself is the root of the fat-tree.*

- *If v_1, v_2, ..., v_i are vertices ant T_1, T_2, ..., T_j are fat-trees, with r_1, r_2, ..., r_k as roots (j and k need not be equal), a new fat tree is built by connecting with edges, in any manner, the vertices v_1, v_2, ..., v_i to the roots with r_1, r_2, ..., r_k. The roots of the new fat tree are v_1, v_2, ..., v_i.*

The above definition is extremely general and can cover ordinary trees, fat-trees with variable-sized switches and multiple connections between vertices and irregular constructions. Some examples are shown in Fig. 6.

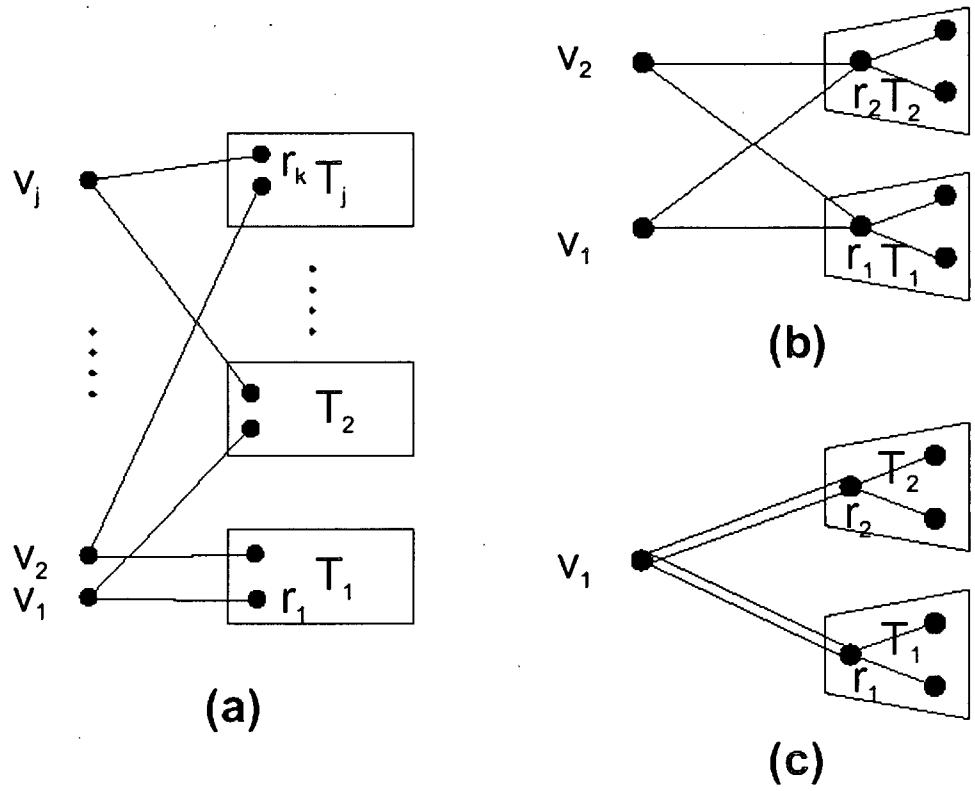

Fig. 6: (a) Fat tree recursively built by connecting the new roots v_1, v_2, ..., v_j to the roots r_1, r_2, ..., r_k of the subtrees; (b) Fat tree with two roots; (c) Fat tree with multiple edges between the root v_1 and the roots of the subtrees T_1 and T_2.

From the family of fat-trees branches the class of *k*-ary *n*-trees [13]. A formal definition of the *k*-ary *n*-trees is:

Definition 2: *A k-ary n-tree is composed of two types pf vertices: $N = k^n$ processing nodes and $nk^{n-1}k$ communication switches. Each node is an n-tuple $\{0, 1, ..., k-1\}^n$, while each switch is defined as an ordered pair $\{w,l\}$, where $w \in \{0, 1,..., n-1\}^{n-1}$ and $l \in \{0, 1,..., n-1\}$.*

- *Two switches $\{w_0, w_1, ..., w_{n-2}, l\}$ and $\{w'_0, w'_1, ..., w'_{n-2}, l'\}$*

  are connected by an edge iff $l' = l + 1$ and $w_i = w'_i$ for all $i \neq l$. This edge is labeled with w'_l on the level l vertex and with w_l on the level l' vertex.

- *There is an edge between the switch $\{w_0, w_1, ... , w_{n-2}, n-1\}$ and the processing node $p_0, p_1, ... , p_{n-1}$ iff*

  $$w_i = p_i \text{ for all } i \in \{0, 1, ..., n-2\}.$$

  This edge is labeled with p_{n-1} on the level n-1 switch.

From *Definition 2* it can be inferred that any path starting from a level 0 switch and leading to a given node $p_0, p_1, ..., p_{n-1}$ traverses the same sequence of edge labels $(p_0, p_1, ..., p_{n-1})$.

Minimally routing between a pair of nodes in a *k*-ary *n*-tree can be accomplished by sending the message to one of the nearest common ancestors of both source and destination and from there to the destination node. Thus, each message experiences two phases: an *ascending* phase to get to a nearest common ancestor, followed by a *descending* phase to get from that common ancestor to destination.

Fat-trees have many interesting properties. Leiserson [14] formally proved the so-called *universality theorem*, stating that a universal fat-tree of a given volume can simulate any other interconnection network of equal volume with only a polylogarithmic factor increase in the time required.

The number of ports of the internal switches of the fat-tree increases as we go closer to the root; this makes the physical implementation of these switches unfeasible. In SoC environment, a key requirement of these switches is that they must be reusable. If the number of channels differs from level to level, the corresponding switches need to be different, which poses difficulties in terms of logical design, placement, routing congestion, non-uniform power dissipation across the chip, etc. For this reason, some alternative constructions have been proposed that use building blocks with fixed number of ports. These solutions trade connectivity with simplicity: in a "complete" fat-tree an incoming message at a given switch may have more choices than in a corresponding network with fixed size (number of ports) switches.

2.2.4 Hybrid Networks

In general, hybrid networks combine mechanisms from shared-medium networks and direct or indirect networks. Therefore, they increase bandwidth with respect to shared medium networks and reduce the distance between nodes with respect to direct and indirect networks. However, for systems requiring very high performance, direct and indirect networks achieve better scalability than hybrid networks because point-to-point links are faster than shared-medium buses [9]. Most high-performance parallel computers use direct or indirect networks. In the case of the on-chip interconnects, hybrid networks are present in the form of hierarchical buses, a typical example being AMBA bus [10] shown in Fig. 7, where there can be, for example, a high speed bus (AHB – Advanced High-Performance Bus) hosting the CPU and DMA (Direct Memory Access) devices,

and a lower speed bus (APB – Advanced Peripheral Bus) hosting slower peripherals (UART, IOs); the two buses are linked together by a bridge.

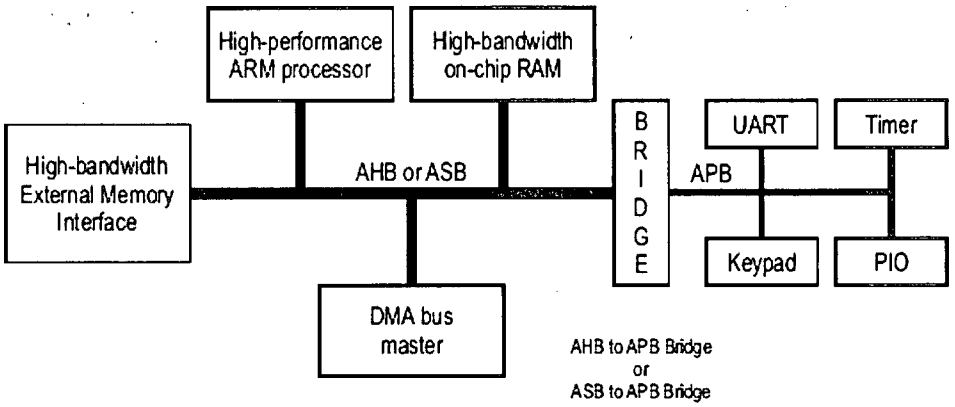

Fig. 7: Typical AMBA Architecture.

In general, hierarchical buses consists of multiple buses connected by bridges, with higher performance buses layered at the higher level of the hierarchy, as indicated in Fig. 8.

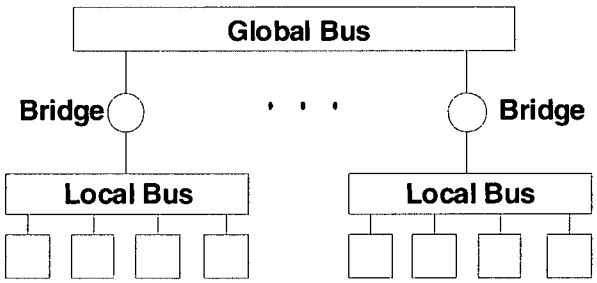

Fig. 8: A two-level hierarchical bus.

Another class of hybrid networks are the cluster-based networks. They combine the advantages of two or more kinds of networks at different hierarchical levels. For example, it is possible to combine the advantages of buses and point-to-point links by using buses at the lower levels in the hierarchy to form clusters and a direct network topology connecting clusters at the higher level. An example is DASH – Stanford

Directory Architecture for Shared Memory, whose basic architecture is shown in Fig. 9. At the lower level, each cluster consists of four processors connected by a bus. At the higher level, a 2-D mesh connects the clusters.

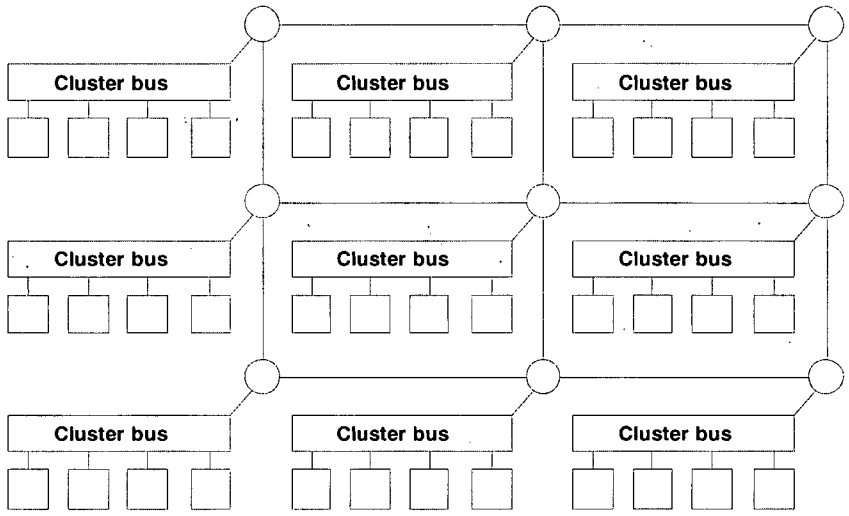

Fig. 9: Cluster-based 2-D mesh.

Other combinations of different structures are possible and have been studied in parallel processing: direct and indirect networks, shared medium and indirect networks, etc.

2.3 Architecture Overview

For the "Network on Chip" project at UBC we have chosen the butterfly fat-tree (BFT) as the interconnect architecture. The BFT offers a good trade-off between the properties of fat-trees and the reusability requirements of the SoC environment, in the sense that all the switches are identical (they have the same number of channels). The architectural evaluation and comparison of BFTs was done in [5]. In the following, a brief formalized description of the particular BFT used will be given. This will help us in developing a wire length model for inter-switch delay analysis in Chapter IV.

We use the BFT with N functional IP (FIP) blocks as shown in Fig. 10 [15][29] .

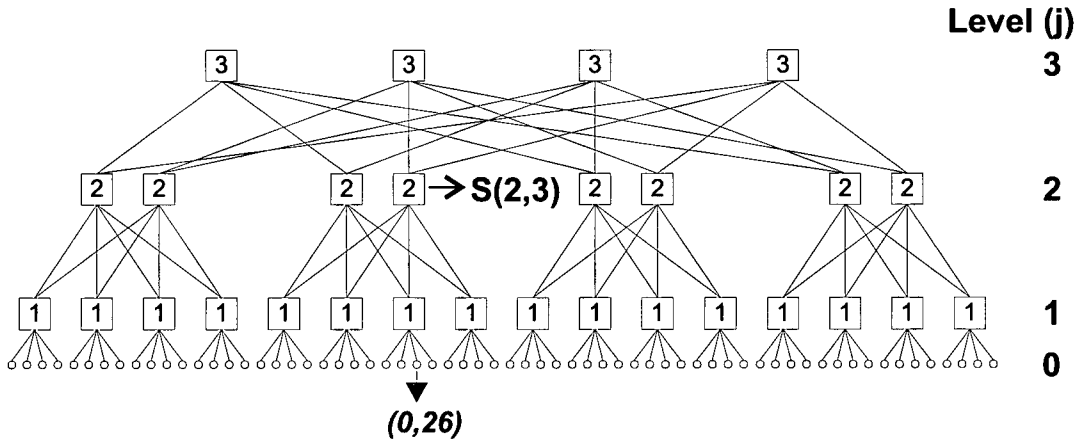

Fig. 10: BFT with 64 IPs.

Each node (leaf or vertex) is labeled by a pair of indices (j,a), where j represents the level of the node in the network and a represents the address of the node in that level (its index). The level of a node is defined as its distance from the leaves. At the lowest level ($j = 0$) are the N FIP blocks with addresses 0 to $N - 1$. Each switch $S(j, a)$ has six ports: $parent_0$, $parent_1$, $child_0$, $child_1$, $child_2$, $child_3$. The number of levels depends on the total

number of FIPs, i.e., for N IPs, the number of levels will be ***levels*** = $(\log_2 N)$ - ***3***. The FIPs are connected to $N/4$ switches at the first level such that the FIP $P(0, a)$ is connected to the $child_{a \bmod 4}$ of switch $S(1, \lfloor a/4 \rfloor)$. At the j-th level (for j = 1 to $(\log_2 N)$ - 3) there are $N/2^{j+1}$ switches. The connections of a switch are determined by the switch's address as follows: $parent_0$ of $S(j, a)$ is connected to $child_i$ of $S(j+1, \left\lfloor \frac{a}{2^{j+1}} \right\rfloor \cdot 2^j + a \bmod 2^j)$, and $parent_1$ of $S(j, a)$ is connected to $child_i$ of $S(j+1, \left\lfloor \frac{a}{2^{j+1}} \right\rfloor \cdot 2^j + \left(a + 2^{j-1}\right) \bmod 2^j)$, where $i = \left\lfloor \frac{a \bmod 2^{j+1}}{2^{j-1}} \right\rfloor$.

Each channel connecting two adjacent switches consist of two unidirectional links. The number of switches in the butterfly fat-tree architecture converges to a constant independent of the number of levels. If we consider a 4-ary tree as shown in Fig. 10, with four down links corresponding to child ports, and two up links corresponding to parent ports, then the total number of switches in level 1 is ***N/4***.

At each subsequently higher level of the tree the number of required switches reduces by a factor of 2. In this way the total number of switches, S, is calculated as:

$$S = \frac{N}{4} + \frac{1}{2}\frac{N}{4} + \frac{1}{4}\frac{N}{4} + \cdots \left(\frac{1}{2}\right)^{levels} \frac{N}{4} = \frac{N}{4}\left(\frac{1-\left(\frac{1}{2}\right)^{levels}}{1-\frac{1}{2}}\right) \xrightarrow[levels \to \infty]{} \frac{N}{2} \tag{2.1}$$

which illustrates that S tends to $N/2$ as N grows arbitrarily large. In the case of 64 IPs the number of switches is 28 as shown in Fig. 10.

2.3.1 Floorplanning and Routing for BFTs

Contemporary VLSI processes offer six or more layers of metallization for wiring. With the advent of Chemical Mechanical Planarization (CMP), it is feasible for process technology to continue stacking additional metal layers as long as the cost of the extra mask steps and processing are justified by area benefits. This produces an interesting effect on the traditional VLSI models: active devices are still largely limited to two-dimensional layout on the silicon substrate. However, wire layers can feasibly be stacked on top of each other creating a three-dimensional structure for interconnects. An efficient placement and routing of an interconnect topology such as the butterfly fat-tree requires a uniform distribution of resources (switches and metal tracks) across the chip [15]. A potential problem with the BFT structure is the wire congestion occurring towards the higher levels of the tree. This congestion can be avoided by intelligently placing the switches on the silicon substrate.

We start by showing that the chip area can be divided into smaller squares of equal size (subsequently called "tiles"), each of these squares containing the same number of functional nodes and switches. Then we account for the wiring per layer and vias required between layers. Each tile consists of a set of four IP blocks connected to the same level one switch, the corresponding level one switch, and, eventually, one more switch belonging to a level characterized by an index greater than one. Thus, the number of tiles is equal to the system size divided by the number of child ports of a switch (four).

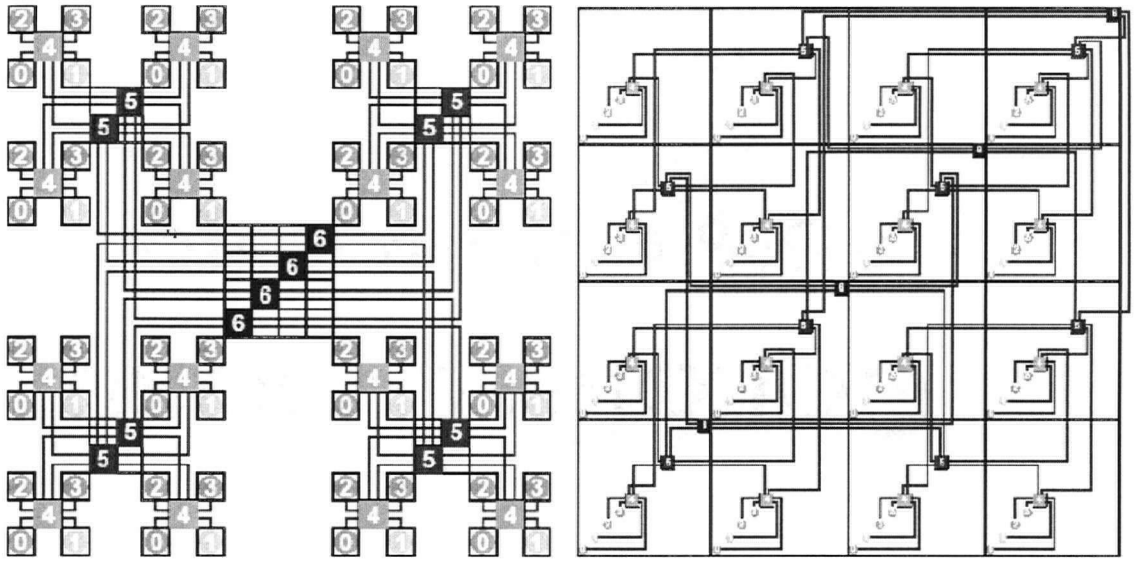

Fig. 11: BFT and physical layout.

Fig. 11 shows the original 2-D layout of the BFT (here with 64 leaf nodes). By rearranging the basic BFT as indicated in Fig. 12, at most two switches end up in each tile, along with four processing nodes.

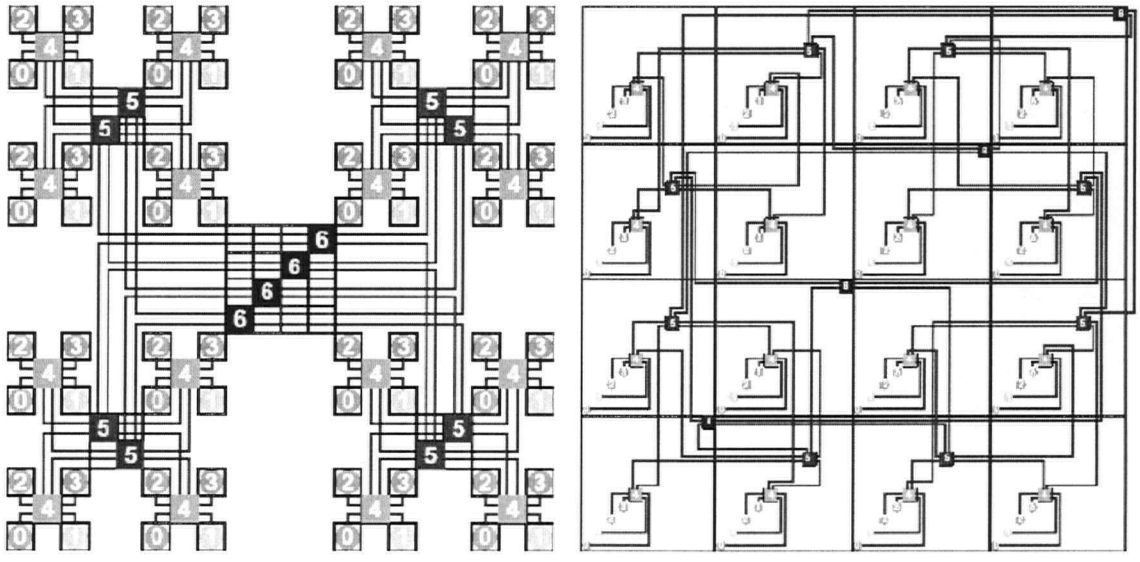

Fig. 12: Rearranged BFT and physical layout.

In the original BFT arrangement, all the switches lie along the same diagonal. In the modified layout, the diagonals are complementary such that, when folded together, the

next diagonal is always left open. Finally, each tile will contain four end nodes, the switch associated with those four end nodes, and, at most, one additional switch.

At each stage, after folding, the lower levels manage to leave both main diagonals free. One main diagonal is then consumed by the new switches added at the level onto which the lower levels are being folded. This, in turn, leaves one diagonal free in the folded box. Consequently, when this new level is now folded with its peers to create the next tree level, it will also create a structure with both main diagonals free so that the next level of switches can be added and the folding can continue further in this manner.

Wires

A simple strategy for wiring is to give each tree level, in a tile, its own pair of metal layers, one for horizontal wires and one for vertical wires. As in each tile there are at most two switches, four levels of metal will be sufficient for all levels, independently of the number of levels in the BFT. This is an upper limit, a lower limit being two metal layers – one for horizontal and one for vertical traces. Therefore, within each tile there will be 6 or 12 wiring channels, and the total number of wire traces will be given by the product ***2NW***, where N is the number of channels, and W is the data channel width. The first term indicates the fact that the channels consist of two unidirectional links in opposite directions. It is important to notice that by using this placement style, the wire congestion at the root of the BFT is completely avoided.

2.4 Summary

In this chapter, the butterfly fat-tree architecture is formally described, and VLSI implications of this interconnect are briefly outlined. We have also shown a strategy to physically place the active devices (switches and end nodes) on the silicon substrate. The chip area was symbolically divided into square tiles, each tile containing at most two switches, of which exactly one is a level one switch. Wire congestion toward the BFT root is thus avoided by arranging the active devices such that there are at most two switches per tile; the maximum number of channels is $24 \times W$ (six ports per switch, each channel consisting of a pair of unidirectional channels) . This simple model for the BFT interconnect will help the development of detailed models for inter- and intra-switch delays in Chapter IV and Chapter V, respectively.

3 Chapter III

Switch Design for Networks on Chip

3.1. Introduction

The building blocks of a network on chip infrastructure are the switches. Their function is to transport data from a source functional block to a destination functional block. They are responsible for the successful routing of messages through the network by implementing the specific flow control mechanism. When a message or packet header reaches an intermediate switch, a switching mechanism determines how and when the input channel is connected to the output channel selected by the routing algorithm. Flow control is tightly coupled with the switching technique for the synchronized transfer of information between switches and through switches in forwarding messages through the network. The flow control mechanism establishes a dialog between sender and receiver blocks, allowing and stopping the advance of information units. If a packet is blocked, it requires some buffer space to be stored. When there is no more available buffer space, the flow control mechanism stops information transmission. When the packet advances and buffer space becomes available, transmission is started again. A simplified block diagram of a switch that performs these basic tasks is given in Fig. 13:

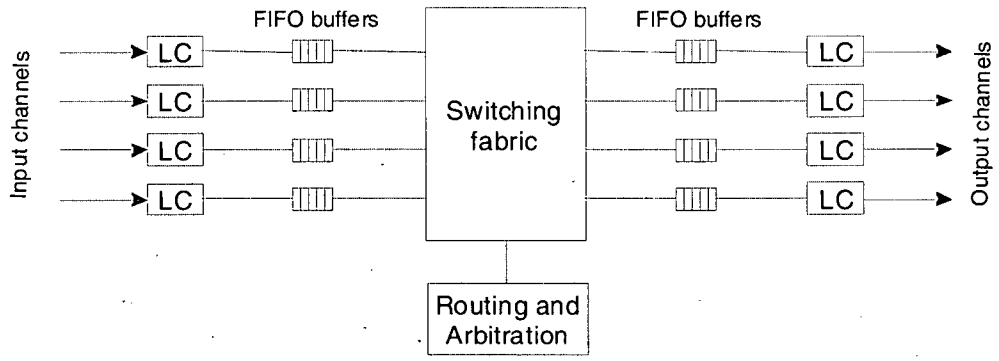

Fig. 13: Block diagram of a switch.

Thus, one can identify the factors that will govern the design of the switch: switching technique, routing algorithm, flow control. The effect of these factors on design decisions is analyzed in the following subsections. As a consequence, the building blocks of a switch, in the simplest case, are the following:

- Routing and Arbitration block: implements the routing algorithm and output buffer allocation;
- Link Controllers (LC): implements the flow control mechanism;
- Switching fabric: connects input channels to output channels according to the decision of the routing block;
- FIFO buffers: store the messages until a free channel is allocated by the Routing and Arbitration unit.

3.2. Switching Technique

The *switching techniques* determine when and how internal switches connect their inputs to outputs and the time at which message components may be transferred along these paths.

There are different types of switching techniques, namely: *Circuit Switching, Packet Switching,* and *Wormhole Switching* [9]. In *circuit switching*, a physical path from source to destination is reserved prior to the transmission of the data. This setting up of an end-to-end path causes unnecessary delay. In packet switching, data is divided into fixed-length blocks called packets, and instead of establishing a path before sending any data, whenever the source has a packet to be sent, it transmits the latter. *Packet switching* is advantageous when messages are short and frequent. Unlike circuit switching, where a segment of the reserved path may be idle for a significant period of time, in packet switching, a communication link is fully utilized when there are data to be transmitted. Packet switching is based on the assumption that a packet must be received in its entirety before any further routing decision can be made to forward the packet towards its destination. The need for storing entire packets in a switch in case of conventional packet switching makes the buffer requirement high in these cases.

In an SoC environment, the requirement is that switches should not consume a large fraction of silicon area compared to the IP blocks. In *wormhole switching,* the packets are divided into fixed length flow control units (*flits*), as indicated in Fig. 14, and the input and output buffers should be able to store only a few flits. As a result, the buffer space requirement in the switches can be small compared to that generally required for packet switching.

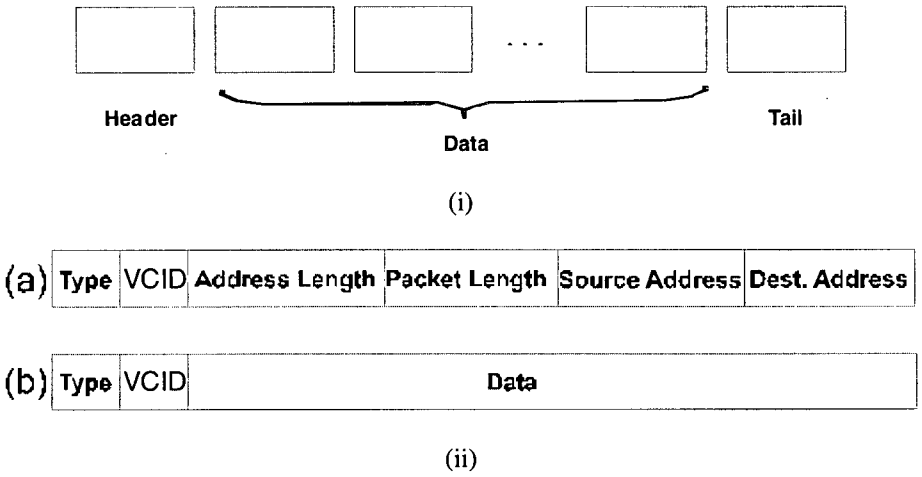

Fig. 14: (i) Message divided into header, data and tail flits; (ii)a: Header Flit, (ii)b: Data and tail Flits.

Thus, using a *wormhole switching* technique, the switches will be small and compact. The first flit, i.e., *header flit,* of a packet contains routing information. Header flit decoding enables the switches to establish the path and subsequent flits simply follow this path in a pipelined fashion. As a result, each incoming data flit of a message packet is simply forwarded along the same output channel, as the preceding data flit and no packet reordering is required at destinations. If a certain flit faces a busy channel, subsequent flits also have to wait at their current locations.

One drawback of this simple wormhole switching method is that the transmission of distinct messages cannot be interleaved or multiplexed over a physical channel. Messages must cross the channel in their entirety before the channel can be used by another message. This will decrease channel utilization if a flit from a given packet is blocked in a buffer. By introducing virtual channels [25] in the input and output ports, channel utility can be increased considerably. If a flit belonging to a particular packet is blocked in one of the virtual channels, then flits of alternate packets can use the other virtual channel buffers, and hence, ultimately, the physical channel. Thus, the

corresponding switch for the BFT architecture described in the previous chapter has six ports (two parent ports P0, P1, and four child ports C0, C1, C2, C3) each port consisting of two unidirectional links in opposite directions, each link being multiplexed over a few virtual channels (FIFO buffers), as in Fig. 15.

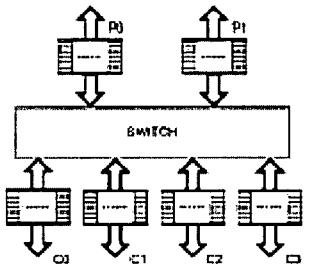

Fig. 15: Switch with 6 ports.

In order to implement virtual channels, multiple FIFO buffers have to be multiplexed over a single physical channel and, hence, an arbitration mechanism is required to implement virtual channel allocation policy. The simplified switch shown in Fig. 13 has to change to accommodate virtual channels, as indicated in Fig. 16.

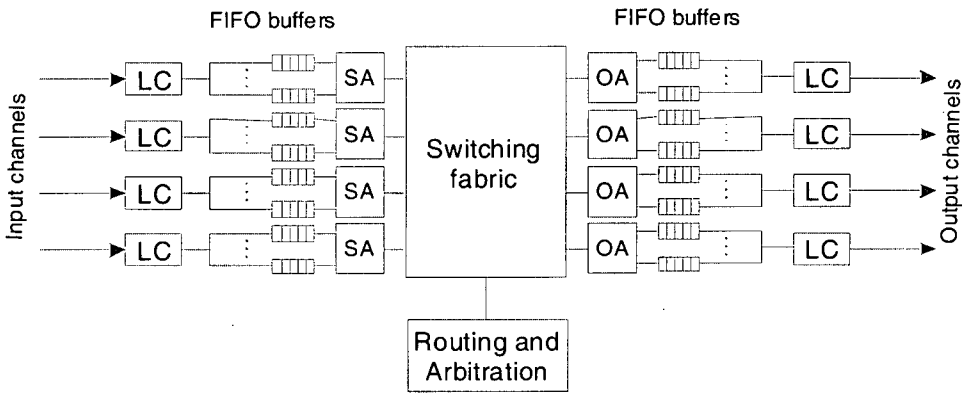

Fig. 16: Block diagram of a switch with virtual channels.

The arbitration is carried by SA (Switch Allocation) blocks at the input side, and by the OA (Output Allocation) blocks at the output side.

An important design parameter is the optimum number of virtual channels per physical channel that has to be implemented. The trade-off here is to maximize the throughput, while keeping the number of virtual channels low to minimize silicon area consumed by FIFO buffers [5]. In order to determine the optimum number of virtual channels, simulations were run using a flit-level wormhole routing simulator. In each simulation cycle, a pair source-destination is randomly selected from the leaf nodes, with equal probability. Messages of equal length (number of flits) are injected at the source nodes. Simulation is run for a period of 20,000 simulation cycles, and then the average throughput is calculated. Throughput is defined as:

$$Throughput = \frac{(Total\ messages\ completed) \times (Message\ length)}{(Number\ of\ IP\ blocks) \times (Total\ time)}$$

Thus, throughput is measured, as the fraction of maximum load the network is capable of physically handling. A throughput equal to 1 means all end nodes are receiving one flit every cycle. Realistically TP <1 since it is improbable that all possible destinations are active each cycle. Successive simulations are run keeping the same message length, but increasing the number of virtual channels per link (FIFO buffers) from one to eight. Fig. 17 shows the effect of sweeping the number of virtual channels on throughput.

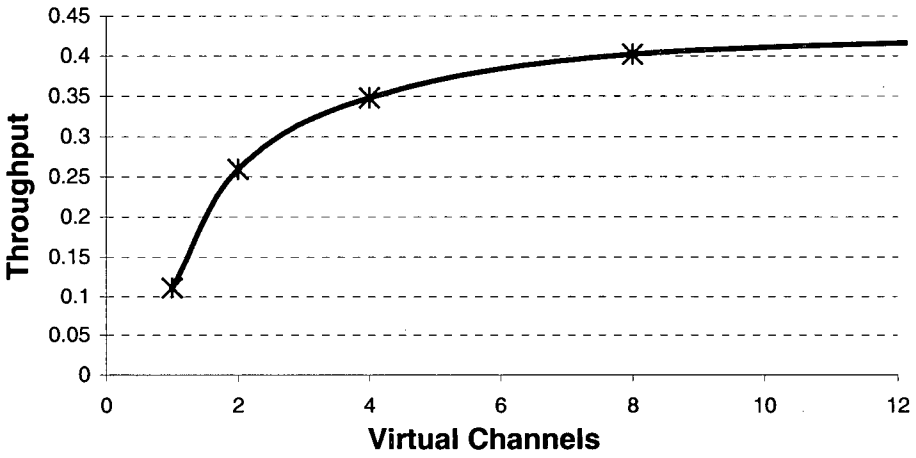

Fig. 17: Effect of multiple virtual channels on throughput.

From Fig. 17, if the number of virtual channels is increased beyond four then there is a trend towards saturation. Since additional buffers are required for each virtual channel, it is advantageous to reduce the number of virtual channels to lower the required silicon area. Thus, a switch with four virtual channels strikes an appropriate balance between high throughput and conservation of silicon area.

3.3. Building Blocks

The operation of the switch consists of one or more processes depending on the nature of the flit. In the case of a ***header*** flit, the sequence of the processes is: (1) ***Input Arbitration;*** (2) ***Routing;*** and (3) ***Output Arbitration***. In the case of ***data*** and ***tail*** flits, ***Switch Traversal*** replaces the routing process as the routing decision based on the header information is maintained for the subsequent body flits. These processes materialize as pipeline stages of the switch, and they alternate as indicated in Fig. 18.

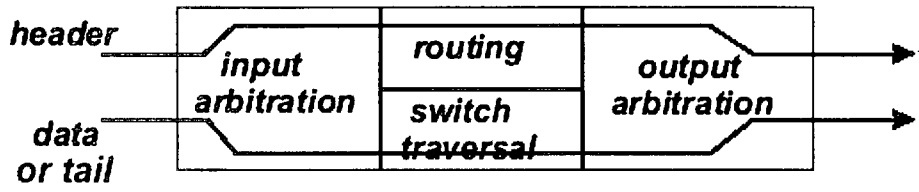

Fig. 18: Switch operation: processes.

When the first flit of a message, i.e. the header flit, enters the switch through a specific port, it is first object of input arbitration. If it wins the arbitration, the routing information contained in the header flit is extracted and fed to the routing block. The flit is directed to one of the other ports, according to the routing information and the specific routing algorithm implemented. At the output port, the flit is again subject to an arbitration stage and is assigned an output virtual channel depending on the availability of the output FIFO buffers. By the time the header flit has left the switch, the path for the rest of the packet is already created, in the sense that virtual channels are reserved and routing decision is made, such that the data and tail flits can follow the header flit in a pipelined fashion.

Each port of the switch consists of two links: an output link and an input link. The effect of the routing process is that an input link of a port is connected to the output link of another port, creating the physical path for message transmission. The block diagram of a pair of input-output links is represented in Fig.19. At the input side, there are four virtual channels multiplexed over a single physical channel through a multiplexer. A 4:1 arbiter circuit controls which of the virtual channels will enter the switch. If the incoming flit is a header flit, the winning channel is then subject to a routing phase and directed to one of the output ports by a demultiplexer. If the flit is not a header, then no routing is required and the flit follows the same path as the header.

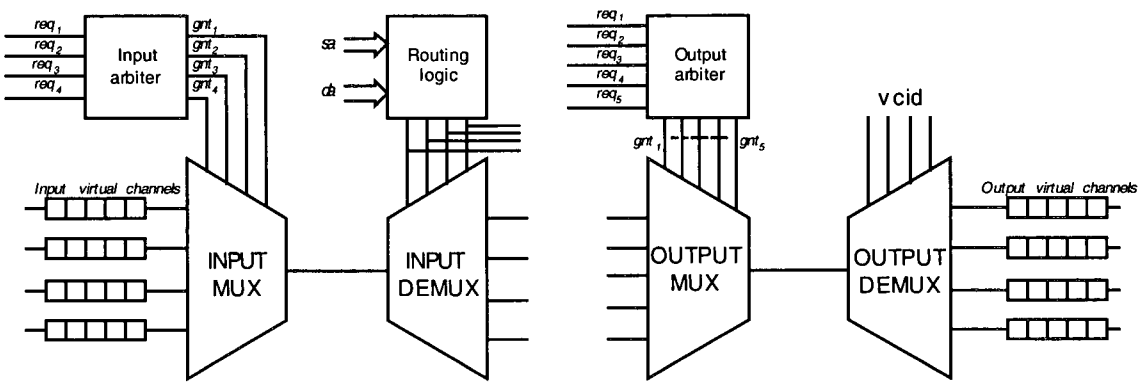

Fig. 19: A pair of input-output links of a switch.

When the flit reaches the output link, it requests access to free output virtual channel. This is again subject to an arbitration phase, as more input links can direct flits to the same output link. Because there are six ports in a switch, at the output there is a 5:1 arbiter required, as all other five input links can request access to a particular output link (a flit cannot exit the switch through the same port it entered the switch [4]).

Arbiter Circuit

The arbiter circuit mainly consists of a priority matrix, which stores the priorities [26] of the requesters and grant generation circuits used to grant resources to requesters. The matrix arbiter stores priorities between *n* requestors in a binary *n-by-n* matrix. Each matrix element *[i, j]* records the binary priority between each pair of inputs. For example, suppose requestor *i* has a higher priority than requestor *j*, then the matrix element *[i, j]* will be set to 1, while the corresponding matrix element *[j, i]* will be 0. A requestor will be granted the resource if no other higher priority requestor is bidding for the same resource. Once a requestor succeeds in being granted a resource, its priority is updated and set to be the lowest among all requestors.

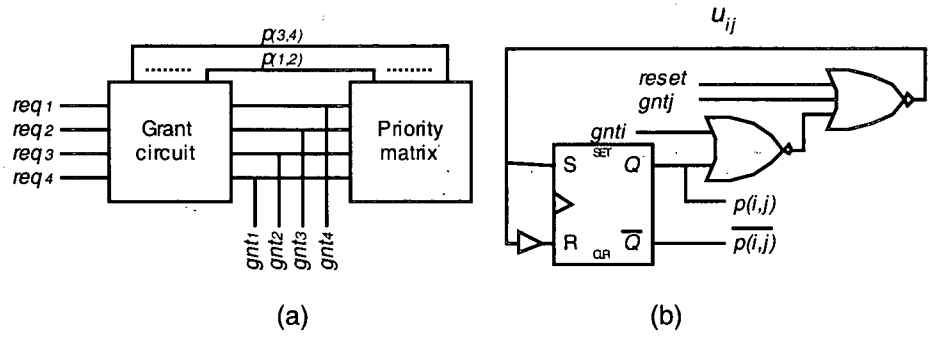

Fig. 20: (a) Block diagram of an arbiter; (b) one element of the priority matrix.

This arbitration policy is efficient because the flit with longest waiting time will have the highest priority, and the time a flit has to spend waiting for getting access to a resource is minimized in this way [26]. The priorities are stored in a matrix of flip-flops. Only the elements above the main diagonal are going to be physically implemented, due to the fact that P_{ij} and P_{ji} are complementary, i.e., if requestor *i* has higher priority than requestor *j*, then requestor *j* has lower priority than requestor *i* ($P_{ij} = \overline{P_{ji}}$).

$$\begin{bmatrix} X & P_{12} & P_{13} & P_{14} \\ P_{21} & X & P_{23} & P_{24} \\ P_{31} & P_{32} & X & P_{34} \\ P_{41} & P_{42} & p_{43} & X \end{bmatrix}$$

Fig. 21: Priority matrix for a 4:1 arbiter.

As an example, consider that the status of the priority matrix is as shown in left matrix in Fig. 22 and requestor *2* is granted access to the switch. Than after arbitration, column *2* is set to 1 and row *2* is set to 0, such that requestor *2* has the lowest priority with respect to all other requestors. This mechanism is implemented by using the grant signals to set/reset the flip-flops storing the elements of the priority matrix as shown in Fig. 20(b).

$$\begin{bmatrix} X & 0 & 0 & 1 \\ X & X & 1 & 0 \\ X & X & X & 0 \\ X & X & X & X \end{bmatrix} \rightarrow \begin{bmatrix} X & 1 & 0 & 1 \\ X & X & 0 & 0 \\ X & X & X & 0 \\ X & X & X & X \end{bmatrix}$$

Fig. 22: Priority matrix transition when requestor *2* is granted access.

The logic equations to express the value of grant signals are given as follows:

$$gnt_1 = req_1(\overline{req_2} + p_{12})(\overline{req_3} + p_{13})(\overline{req_4} + p_{14})$$
$$gnt_2 = req_2(\overline{req_1} + \overline{p_{12}})(\overline{req_3} + p_{23})(\overline{req_4} + p_{24})$$
$$gnt_3 = req_3(\overline{req_1} + \overline{p_{13}})(\overline{req_2} + \overline{p_{23}})(\overline{req_4} + p_{34})$$
$$gnt_4 = req_4(\overline{req_1} + \overline{p_{14}})(\overline{req_2} + \overline{p_{24}})(\overline{req_3} + \overline{p_{34}})$$

Applying De Morgan's law to the equations above, the gate level circuit for a grant signal is shown below:

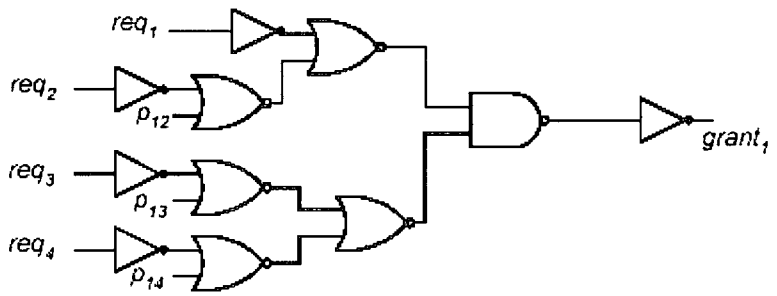

Fig. 23: Logic circuit to generate *grant*$_1$ signal.

The rest of the grant signals are generated similarly to Fig. 23. The critical path of the grant circuits consist of a sequence of one inverter, two successive 2-inpus NOR gates, a NAND gate and a final inverter. This critical path is continued with the priority matrix elements, as shown in Fig. 20(b). The detailed calculation of the delay of the critical path of the arbiter circuit is given in Chapter 5.

Routing Circuit

The routing circuit implements a simple LCA (Least Common Ancestor) determination algorithm. It compares a certain range of bits of the source and destination addresses [4] and, if the result string contains at least a '1' bit (LCA bit), it directs the message to one of the available parent ports. If the result string contains only '0' bits, it directs the message to one of the child ports. The address length is a function of the size of the system (number of IP components), and hence, the routing circuit depends on the size of the system. As such, the circuit to implement this simple algorithm for a 64 IP system (six bits address length), has the structure shown Fig. 24.

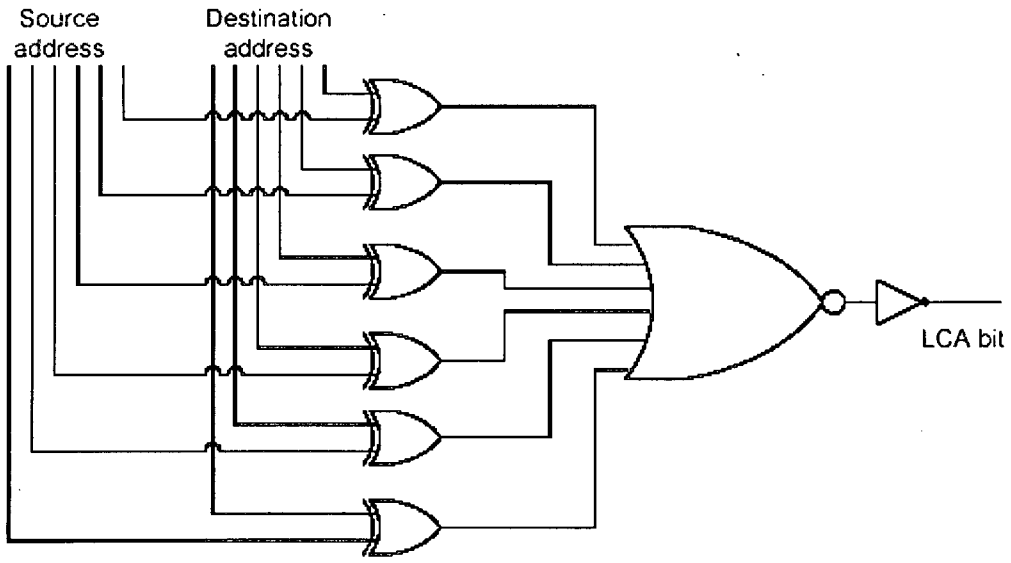

Fig. 24: Routing block.

The 6-inputs NOR gate is not feasible for CMOS implementation due to its large delay, and the solution is to replace it with a tree of NOR and inverter pairs [24]. The depth of the tree grows logarithmically with the number of inputs of NOR gate, which

helps in reducing the delay of the equivalent NOR gate. A 6-inputs NOR gate can be replaced by a two-levels tree of NOR/inverter pairs, as shown in Fig. 25.

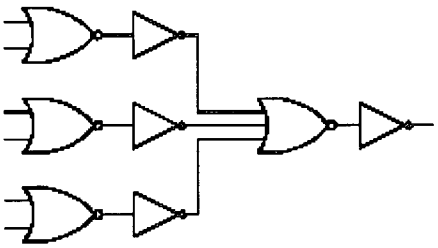

Fig. 25: Tree of NOR gates.

The size of the routing circuit depends on the number of inputs, i.e., on the number of bits in the address field, which has a logarithmic dependence on the system size. Accordingly, the size of the routing circuit is a logarithmic function of the system size.

3.4 Silicon Area Overhead

To evaluate the feasibility of the BFT interconnect scheme we need to study its silicon area overhead. As the switches are the integral active components of this infrastructure it is important to determine the amount of relative silicon area consumed by those.

After synthesis using Synopsys' Design Compiler and Virtual Silicon 0.18μm standard cell library, the total area of a switch with six ports and four virtual channels per port is reported as 35,500 equivalent 2-input NAND gates. From the total amount, less than 10% is used to implement arbitration, routing and traversal, while the rest is consumed by FIFO buffers implementing the virtual channels.

The number of IPs in a single SoC varies from one technology node to the other. Consequently, the number of bits required to address the IPs will also vary. This will be reflected in the length of the header flit as shown in Fig. 14(ii). Two bits are needed to specify each of *Flit Type* and *VCID*. Simulation results [5] show that throughput does not vary much with the packet length, as shown in Fig. 26. However packet length negatively affects the latency [5]. As a trade-off between throughput and latency the packets are assumed to consist of 16 flits and 4 bits will be sufficient to denote the packet length in each technology node.

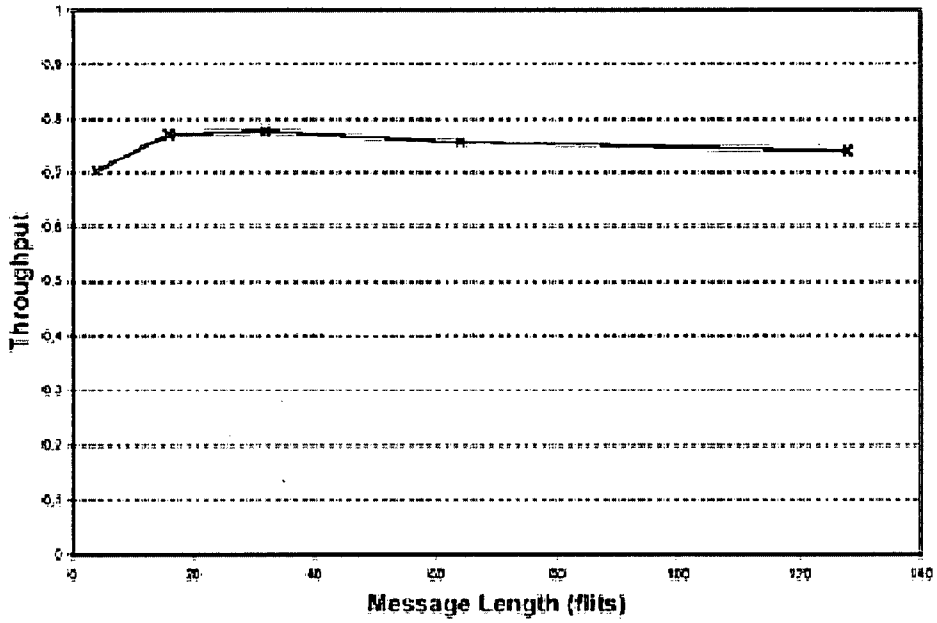

Fig. 26: Effect of message length on throughput.

An SoC consists of two types of IPs, the functional IPs integrated with the help of infrastructure IPs, i.e. the switches. The number of functional IPs govern the number of bits required to denote each of address length, source address and destination address fields. Table 2 shows the header flit length (number of bits) in different technology nodes for the BFT architecture. The header flit length can expressed as:

$$Header\_length = 2_{type} + 2_{VCID} + 4_{address\_length} + 4_{packet\_length} + S_{source\_address} + D_{destination\_address} \quad [bits]$$

where $S_{source\_address}$, $D_{source\_address}$ denote the length of the source and destination address fields of the header flit, and *Header_length* is the total length of the header flit, in bits. In order to determine the number of bits needed for the source and destination address fields, we need to know the maximum number of 100K gates IP blocks that can be integrated on a NoC. Assuming a 20mm x 20mm chip size, the size of a 2-input NAND gate being 11 μm^2 in TSMC 0.18μm CMOS technology and a scaling factor of 0.7 for successive technology nodes [3], one can calculate the number of digital IP blocks that can be fitted on a chip. From these, due to the properties of BFT topology (the upper bound of the number of switches is half the number of leaf nodes), one third will be switches and the rest will be functional IPs. The distribution of the number of switches and functional IPs is given in Table 1. Accordingly, the length of the source/destination address can be calculated as log_2 (Number of functional IPs).

Table 1: Distribution of IP blocks and switches in successive technology nodes.

Technology node	Max. Number of IPs	Number of Functional IPs	Number of Switches
130nm	500	333	167
90nm	1000	666	337
65nm	2500	1666	834
45nm	7500	5000	2500
32nm	10000	6666	3334

Table 2: Flit Lengths – BFT.

Technology node	Type	VCID	Add. Length	Packet Length	Source Address	Destination Address	Header Flit Length
130 nm	2	2	4	4	9	9	30
90 nm	2	2	4	4	10	10	32
65 nm	2	2	4	4	11	11	34
45 nm	2	2	4	4	13	13	38
32 nm	2	2	4	4	13	13	38

The length of the data flits is kept equal to the length of the header flit. To estimate the silicon area consumed by the buffers, we developed a VHDL model of a switch, having four virtual channels using a fully static, standard cell-based, CMOS 0.18 μm technology. Simulation results shown in Fig. 27 indicate that the throughput is relatively independent of buffer depth. Therefore, to save silicon area the depth of the FIFO buffer is kept as one flit.

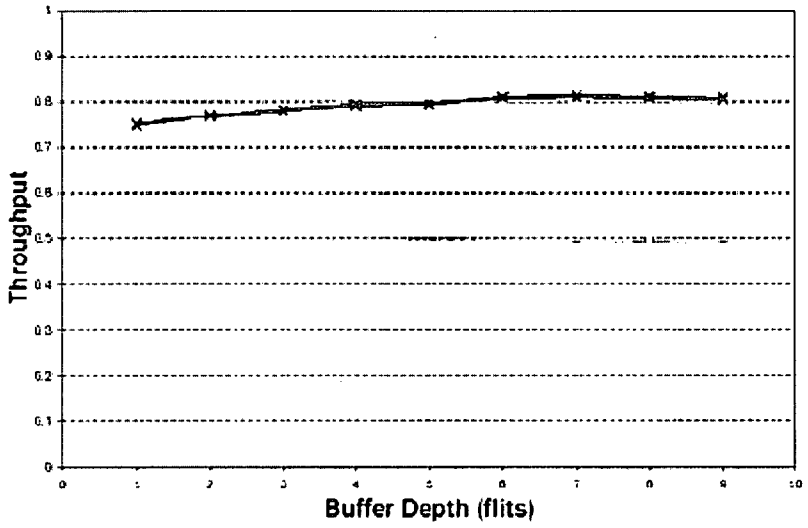

Fig. 27: Buffer depth impact on throughput.

The switches have two main components, the storage buffer, and logic to implement routing, flow control. The storage buffers are the FIFOs at the inputs and outputs of the switches. Using data from Tables 1 and 2, we can estimate the amount of silicon area consumed by the infrastructure IP blocks (switches) in different technology nodes for the BFT interconnect architecture. The procedure for area calculation is straightforward:

- the size of a switch in 0.18μm technology is known as 35,500 2-input NAND gates;

- the scaling factor for successive technologies is 0.7 according to [3];
- the size of the switch is assumed to be proportional with the number of bits required for the virtual channels (FIFO buffers);
- the number of switches is given in Table 1;
- the size of the FIFO buffers (in bits) is given in Table 2;
- the area overhead due to switches is calculated with the expression:

Area = No. of switches x No. of virtual channels x Flit size

[equivalent 2-input NAND gates]

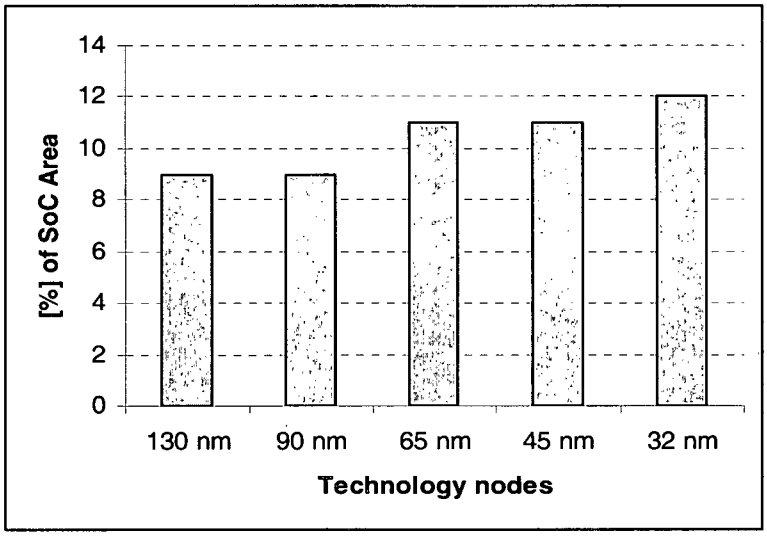

Fig. 28: I²P Area Overhead.

The total silicon area consumed by switches amounts from 9% in 130 nm technology, to 12% in 32nm technology node, for a 20mm x 20mm total chip area. Given the advantages that such an architecture offers in terms of parallel programming capability and latency, the area overhead is within reasonable limits.

3.5 Summary

In this chapter we have detailed the main design considerations for the infrastructure blocks, i.e., switches, of a network on chip, here considering the BFT (Butterfly Fat-Tree) topology in particular. The main system level parameters governing physical implementation of switches have been identified and their effect on design decisions analyzed. The major building blocks of a switching element were described. Silicon area overhead for a complete system in different technology node was shown to be between 9% (130nm) and 12% (32nm).

4 Chapter IV

Inter-Switch Wire Delay Analysis

We begin this chapter by outlining the trends that develop in SoC design in the context of semiconductor technology evolution. We then show a simple model for the inter-switch wire length in BFT topology; based on this model we evaluate inter-switch delays. Finally, for comparison, the scalability of shared medium (bus) topology is analyzed from a delay point of view and a simple metric is developed to quantify it.

4.1 Soc Microarchitecture Trends and Assumptions

In a conventional digital ASIC design flow, several iterations of logic synthesis and physical design are required before convergence to design specifications is achieved. During synthesis, the capacitances of the global wires are generally unknown, and *wire-load models* are typically used as estimators. The accuracy of such estimations is generally acceptable for short wires, but increasingly unacceptable as the wire delays reach levels where they constitute a significant portion of the critical path delay.

For IP blocks consisting of 50-100K gates, such interconnect delay estimation related problems can be reasonably well tackled by existing CAD tools [16]. Moreover, various publications show that global wires in blocks of 50-100K gates tend to scale with technology [17] [18]. Therefore, problems in ultra deep submicron processes arising from non-scalable global wire delay and poor back annotation mechanisms can be assumed to be readily surmountable when these are limited to such blocks. There is plenty of evidence in support of IP blocks amounting to such sizes. For example, a 32-bit DSP

core can have around 115K gates [19]; a MPEG2 decoder can consist of approximately 60K gates [19], and a general purpose 32-bit RISC microprocessor can amount to around 50K gates [19].

We are already at a point where a few new designs coming out from industry consist of up to 100 embedded processors [20]. By extension to the above, we conjecture that the trend for future SoC integration will be based on a hierarchical design paradigm where an increasing number of IP blocks consisting of 100K gates will be integrated according to a specific interconnect template. One possible interconnect template is the butterfly fat-tree as shown in Fig. 10.

Assuming IP blocks consisting of 100K gates and a constant chip size [3] of 20mm x 20mm, Table 2 shows the maximum number of IP blocks that can be integrated in a single SoC in different ITRS technology nodes [18]. In the foregoing, we assume that such blocks would be integrated according to a butterfly fat-tree microarchitecture template, described in Chapter 2. As reported in Table 3, as the number of IP blocks increases, the number of required levels in the butterfly fat-tree also increases. Table 3 also reports the number of required BFT levels for each technology node. In the next section, we show that the increased number of levels does not negatively impact the achievable clock cycle rates for the SoC.

Table 3: Maximum number of IP blocks (100K gates/IP block) [18] and corresponding fat-tree levels as technology scales

Technology Node	Max. No. of IPs	No. of BFT levels
130 nm	500	6
90 nm	1000	7
65 nm	2500	9
45 nm	7500	10
32 nm	10000	11

4.2 Interconnect Models and Trends

The demand for high levels of integration in semiconductor industry has resulted in an aggressive shrinking of the devices, with the added bonus of increased device speeds resulting from smaller channel lengths. Interconnect delay, which was formerly insignificant, is rapidly becoming a bottleneck due to degrading performance trends with scaling [17] [21]. Longer wires due to a larger chip size, coupled with smaller and more closely packed interconnects (smaller pitch) is leading to a continuous increase in resistance and capacitance, forcing longer RC interconnects delays with each generation. Several solutions have been proposed at different levels: physical design, circuit, and material level. The physical design solution is to progressively increase the number of metal layers in the future. This leads to more relaxed dimensions for longer wires at the top metal levels. However, an excessive increase in the number of metal layers inflates process complexity and cost. At the circuit level, the most common solution is repeater insertion to mitigate the increase in global wire delay [21]. The major penalty of repeater insertion is area and power consumption. Finally, the material-based solution consists of replacing aluminium and silicon-dioxide with copper and low dielectric constant (low-k) materials, respectively. The effect is the increase of speed by reducing the resistance and capacitance per unit length.

These solutions alone are not enough to allow the continuation of the existing design paradigm [22]. In the following, we will briefly introduce simple models for capacitance and resistance of metal traces, which will help us apply the repeater insertion methodology for inter-switch wires in the BFT topology.

Resistance

For metal traces with rectangular cross-section the resistance is calculated as:

$$R = \frac{\rho}{T} * \frac{L}{W} = R_{sq} * \frac{L}{W} = R_w * L \tag{4.1}$$

where ρ is the metal resistivity. It has been shown [22] that the resistivity of global metal traces is not constant, but rises slowly with shrinking of feature size; in fact, resistivity varies from 2 μΩ-cm in 0.18μm technology, to 4 μΩ-cm in 32 nm technology [22].

Capacitance

The capacitance per unit length, needed for delay calculations, is obtained using a simple parallel plate model consisting of inter and intra-level components, along with a fringe component, as shown in Fig. 29 [23].

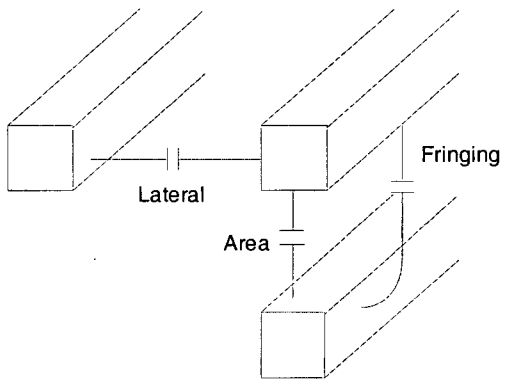

Fig. 29: Lateral, fringing and parallel plate components of the wire capacitance.

It is interesting to observe that as feature size shrinks, the parallel plate component of wire capacitance decreases slowly, while the lateral and fringe components remain almost constant. The overall effect is that after 65 nm technology node, lateral and fringe capacitance will dominate and the total wire capacitance per unit length will remain almost constant [22]. Accordingly, capacitance can be calculated with the formula:

$$C = 2C_{fringe} + 2C_{area} + 2C_{lateral} \tag{4.2}$$

When dealing with global wires, it is appropriate to use a distributed RC model for more accurate calculation. Assuming that the wire is divided into n sections, and applying a simple Elmore delay method, the delay of the wire in Fig. 30 can be estimated as [30]:

$$D = R_w C_w \cdot \frac{n(n+1)}{2n^2} \cdot L^2 \xrightarrow[n\to\infty]{} \frac{R_w C_w}{2} L^2 \tag{4.3}$$

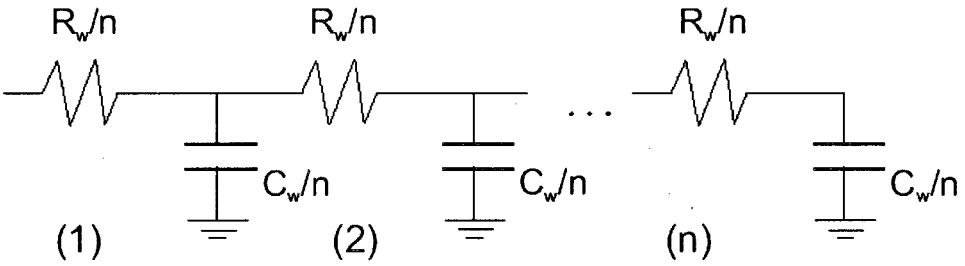

Fig. 30: Distributed wire model.

However, a more accurate expression for the delay of a wire considering the distributed model is [21]:

$$D = 0.4 R_w C_w L^2 \tag{4.4}$$

Repeater Insertion

From the delay equation *(4.4)* , it is evident that delay increases quadratically with wire length. This dependence can be reduced to a linear one by inserting repeaters [21]. By breaking the wire into N sections and inserting N repeaters of size M times the minimum inverter, the new delay can be calculated as:

$$D_{buffered} = Nt_{inv} + \left(C_G R_w M + \frac{C_w R_{eqn}}{M} \right) L + 0.4 R_w C_w \frac{L^2}{N} \qquad (4.5)$$

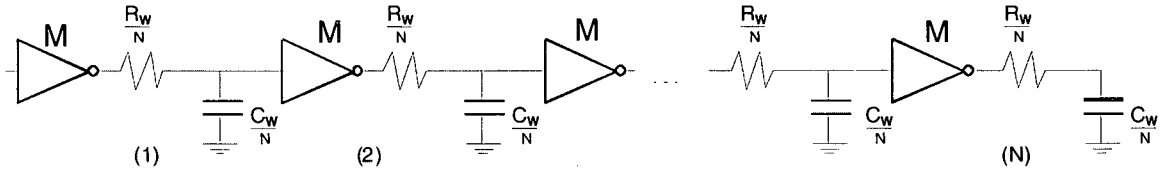

Fig. 31: Buffer insertion.

Here, we have used the following notations:

- t_{inv} – delay of an inverter driving its own parasitic capacitance
- C_G – gate capacitance of a minimum size inverter with equal rise/fall time
- R_{eqn} – equivalent resistance of the *n*-type diffusion region in Ω/□

Fig. 32 plots the unbuffered and buffered wire delay for upper metal layers in 0.13μm technology using the corresponding resistance and capacitance parameters from Table 4. The horizontal line represents the 15*FO4* limit taken as reference according to ITRS 2001. It can be noticed that the maximum length of global wire that can be theoretically utilized in this specific technology is around 10 mm; in this case the wire delay represents 100% of the critical path of the signal transported through it.

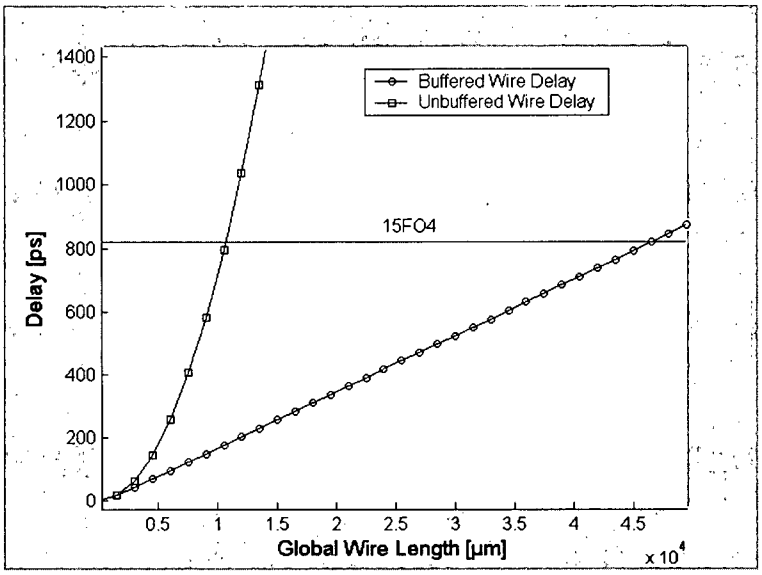

Fig. 32: Unbuffered and buffered global wire delay in 0.13-µm technology.

To explain why the reference limit of 15*FO4* is considered here, one must take into account the trend of the clock cycle of high-performance processors. In order to increase the achievable clock cycle and, consequently, the performance of processors, designers had two main resources: at device level, technology improvements leading to smaller feature size with better gate delays as an immediate consequence, and, at circuit level, the amount of combinational logic within a pipeline stage was reduced. This trend is shown in Fig. 17 [3]. It is projected that the normalized clock cycle will saturate somewhere in the range of 10 – 15 *FO4* delay units. The main reason is the fact that around 4*FO4* units is the overhead of the clocked elements (latches, flip-flops), while the rest up to 10 – 15 *FO4* can be used for combinational logic.

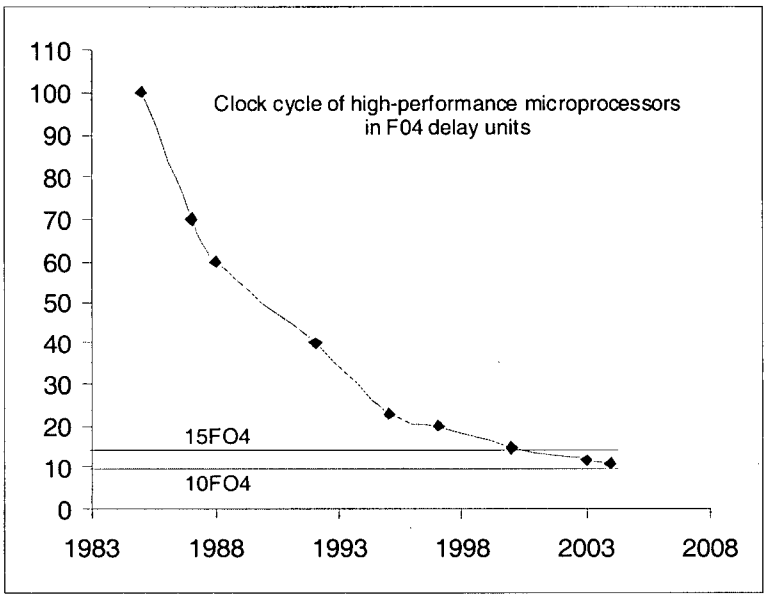

Fig. 33: Clock cycle of high-performance microprocessors in normalized units of FO4.

4.3 Inter-Switch Wire Delay in BFT Architecture

The wire length between switches in the butterfly fat-tree architecture depends on the levels of the switches. For ease of analysis, we will use the simplified layout of the BFT shown in Fig. 34 to determine the inter-switch wire-length expression. Let L_{chip} be the size of the chip on one side, assuming a square silicon die, and let $Area$ be the area of the chip, $Area = L_{chip}{}^2$. In the case of the BFT topology with 64 leaves, the wire lengths between successive levels of switches can be calculated as:

$$w_{1,0} = \frac{L_{chip}}{2^3} \tag{4.6}$$

$$w_{2,1} = \frac{L_{chip}}{2^2} \tag{4.7}$$

$$w_{3,2} = \frac{L_{chip}}{2} \tag{4.8}$$

where $w_{1,0}$ is the length of the physical channel between the IP blocks and the first level of switches, $w_{2,1}$ is the length of the physical channel between switches of level two and one and so on.

In general, the inter-switch wire length is given by the following expression:

$$w_{a+1,a} = \frac{\sqrt{Area}}{2^{levels-a}} \tag{4.9}$$

where $w_{a+1,a}$ is the length of the wire spanning the distance between level a and level $a+1$ switches, where a can take integer values between *0* and (***levels-1***), with ***levels*** being the number of BFT levels in the particular interconnect implementation.

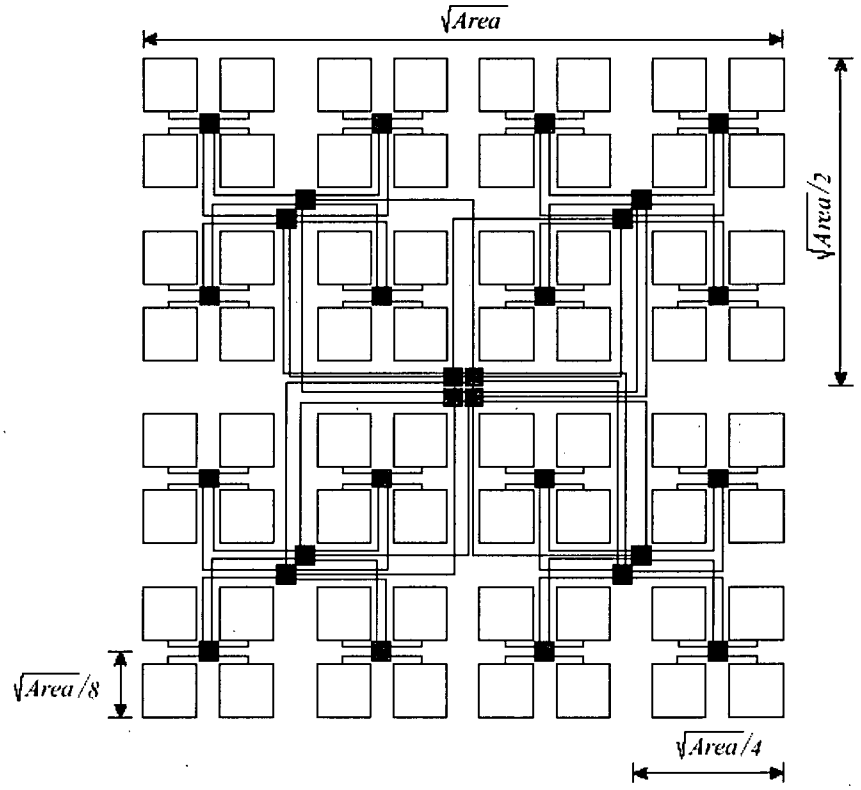

Fig. 34: Inter-switch wire lengths in a 64-IP BFT.

Table 4 shows the inter-switch wire length in mm for different technology nodes. X denotes that the particular inter-switch wire is not present in the concerned technology node. The maximum die size is assumed to remain unchanged at 20 mm, assumption supported by ITRS 2001 projections [3].

Table 4: Inter-switch wire lengths in mm

Technology node	No. of levels	$W_{11,10}$	$W_{10,9}$	$W_{9,8}$	$W_{8,7}$	$W_{7,6}$	$W_{6,5}$	$W_{5,4}$	$W_{4,3}$	$W_{3,2}$	$W_{2,1}$
130 nm	6	X	X	X	X	X	10.000	5.000	2.500	1.250	0.625
90 nm	7	X	X	X	X	10.000	5.000	2.500	1.250	0.625	0.312
65 nm	9	X	X	10.000	5.000	2.500	1.250	0.625	0.312	0.156	0.078
45 nm	10	X	10.000	5.000	2.500	1.250	0.625	0.312	0.156	0.078	0.039
32 nm	11	10.000	5.000	2.500	1.250	0.625	0.312	0.156	0.078	0.039	0.019

We can compute the intrinsic *RC* delay [21] of a wire according to the equation below:

$$D_{unbuffered} = 0.4 R_w C_w L^2 \qquad (4.10)$$

where R_w and C_w are the resistance and capacitance per unit length of the wire, respectively, and L is the wire length. The minimum conceivable clock cycle time considering a highly pipelined design style can be assumed to equal the value of 15FO4, with FO4 defined as the delay of an inverter driving four identical ones [24]. In different technology nodes, FO4 can be estimated as $425*L_{min}$ [ps] where L_{min} is the minimum feature size in each technology node [3]. For long wires, the intrinsic delay will easily exceed this 15FO4 limit. In those cases, the delay can, at best, be made to increase

linearly with wire length by inserting buffers. If the wire is divided into N segments and a total of N inverters inserted, then the total delay of the buffered wire will be according to the following expression [21]:

$$D_{buffered} = Nt_{inv} + \left(C_G R_w M + \frac{C_w R_{eqn}}{M} \right) L + 0.4 R_w C_w \frac{L^2}{N} \tag{4.11}$$

where t_{inv} is the delay of an inverter sized for equal rise and fall propagation delays, and can be approximated as t_{inv}=FO4/5. M is the size of the inverters, C_G is the gate capacitance of the minimum size inverter with equal rise and fall times, R_{eqn} is the large signal resistance of n-type transistor in Ω/□. Differentiating $D_{buffered}$ with respect to N and equating to zero yields the optimum number of segments [21]:

$$N = \sqrt{\frac{0.4 R_w C_w L^2}{t_{inv}}} \tag{4.12}$$

R_w can be calculated according to the following formula:

$$R_w = \frac{\rho}{TW} \tag{4.13}$$

where ρ is the resistivity of the metal wire (here assumed to be 2.2Ωμm for copper), and T and W are the wire thickness and width, respectively.

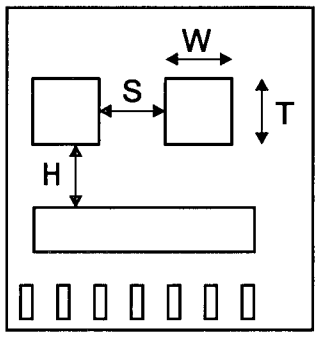

Fig. 35: Cross section of multiple metal layers.

C_w can be calculated according to the following equation [22]:

$$C_w = 2\varepsilon_d \varepsilon_0 \left(\frac{1+2\left(T/W\right)^2}{\left(T/W\right)} \right) + C_{fringe} \tag{4.14}$$

where ε_d is the dielectric constant, ε_0 is the permittivity of free space. C_{fringe} is the fringing capacitance assumed to be constant and equal to 0.04fF/μm in all technology nodes [22]. In our calculations of R_w and C_w, the inter-level dielectric thickness *(H)*, top level metal thickness *(T)*, intra-level dielectric thickness *(S)*, and top level wire width *(W)*, are all assumed to be the half pitch [22] for the given technology node, as shown in Fig. 35. Specific values for R_w, C_w and t_{inv} are shown in Table 5 for successive technology nodes.

Table 5: Values of R_w, C_w, t_{inv} and FO4 in different technology nodes

Technology node	R_w [Ω/μm]	C_w [fF/μm]	t_{inv} [ps]	FO4 [ps]
130 nm	0.06	0.30	11.05	55.25
90 nm	0.12	0.22	7.65	38.25
65 nm	0.20	0.20	5.50	27.5
45 nm	0.44	0.20	3.82	19.1
32 nm	0.73	0.20	2.70	13.5

We used the values of R_w, C_w, and t_{inv} from Table 4 to calculate unbuffered and buffered global wire delay in different technology nodes.

Figs. 36 and 37 report the unbuffered and buffered global wire delay variation with wire length in successive technology nodes, with D130 denoting 130 nm technology, D90 denoting 90 nm etc., and 15FO4 130 denoting 15 times the delay of an inverter driving four identical inverters in 130 nm, 15 FO 90 that for 90 nm etc.

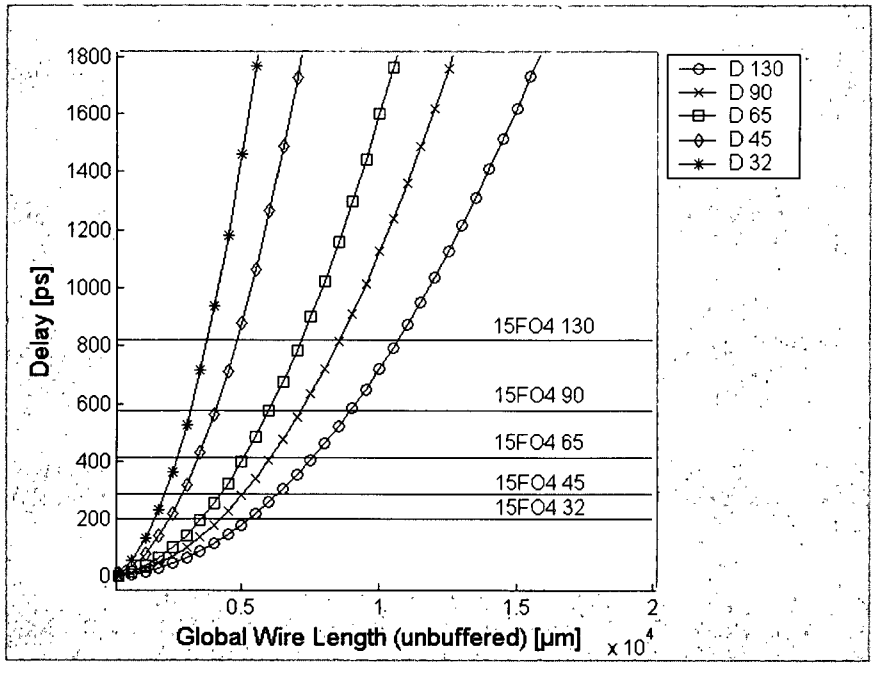

Fig. 36: Unbuffered global wire delay in different technology nodes.

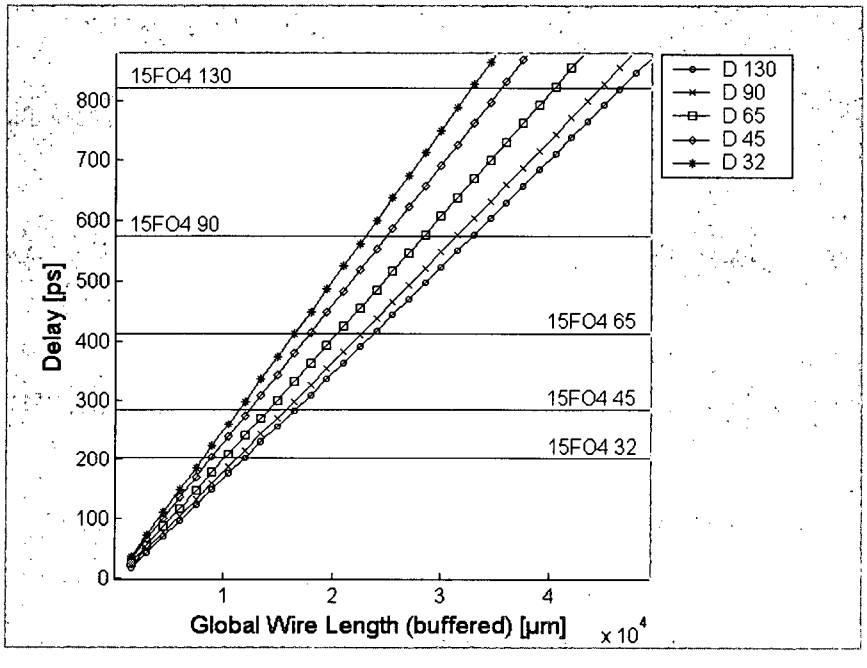

Fig. 37: Buffered global wire delay in different technology nodes.

From Figs. 36 and 37, the length of global wires (inter-switch connections), which require buffering, can be determined. Shading in Table 5 highlights these. From Table 5,

it can be noticed that most of the inter-switch wires need not be buffered. Consequently, the inter-switch propagation delay always remains within one clock cycle. This facilitates achieving system-level timing closure and brings out one advantage of switch-based SoC design, i.e., global wires requiring buffering can be identified in early stages of the design cycle.

Another advantage that emerges from our scheme is that the inter-switch wire delay, and hence, the clock cycle, are largely independent of the number of IP blocks in the system. In our networked SoC, the only global wires are those that span distances between switches. As the inter-switch wire delay (either buffered or unbuffered) does not exceed one clock cycle (i.e., 15FO4 delay units), these switch-based SoCs do not suffer from the global wire delay problem that arises in ultra deep submicron technologies.

An important point is that our inter-switch wire length and delay analysis and its results do not strongly depend on the IP block size assumption. If the number of gates in the IP blocks were to largely exceed 100K gates, or were much smaller than 50K, then the total number of IP blocks in an SoC would scale accordingly, i.e., inversely to the size of IP blocks. Consequently, only the number of levels in our template would change. The inter-switch wire length and delay would remain largely unaffected.

4.4 Wire Delay in a Shared Medium SoC

In this section we analyze the effects on delay of connecting IP blocks to a bus. In a bus-based SoC, multiple IP blocks share the transmission media. As the number of connected IP blocks increases, the capacitance attached to the bus wires increases correspondingly. This negatively impacts propagation delay, and, ultimately, the

achievable clock cycle. This thus limits the number of IP blocks that can be connected to the bus, and thereby the system scalability.

Each attached IP block will capacitively load the bus wires. For ease of analysis (but without loss of generality), we assume this extra capacitance to be evenly distributed along the wire and model it as a parasitic capacitance.

As many existing on-chip buses are multiplexer - based [10] [11] [12], as shown in Fig. 38, they are basically unidirectional and can therefore easily be buffered.

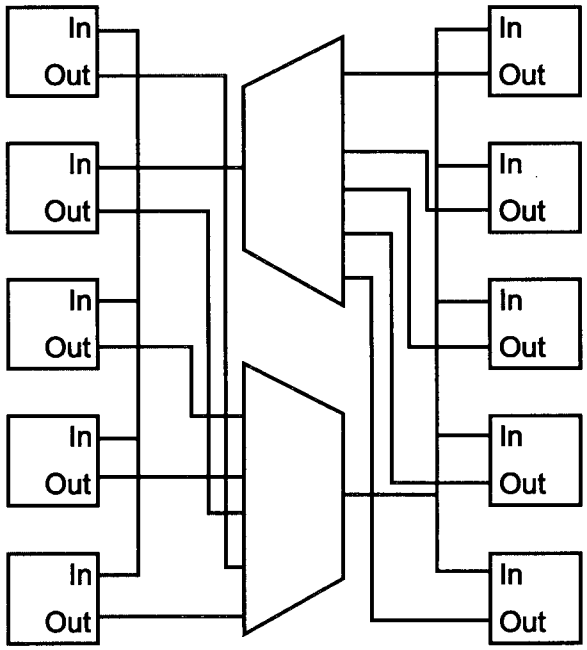

Fig. 38: Multiplexer-based bus architecture.

We consider the length of bus wires, L_{bus}, to equal the maximum unbuffered wire length at each technology node as shown in Fig. 37, as this length can be driven within one clock cycle. Attaching IP blocks to a bus adds an equivalent capacitance of C_p per unit length of wire. As a result, the driving capability of the bus will be negatively affected, and buffer insertion is required to accommodate multiple IPs while satisfying a

propagation delay within one clock cycle. If a bus wire is divided into N segments, then each wire segment will have a capacitance of $(C_w + C_p)$ per unit length and the delay in the buffered bus wire can be obtained by modifying equation (4.11). The delay in this case will be as follows:

$$D_{buffered,bus} = N_{bus} t_{inv} + \left(C_G R_w M + \frac{(C_w + C_p) R_{eqn}}{M} \right) L_{bus} + 0.4 R_w (C_w + C_p) \frac{L^2_{bus}}{N_{bus}} \quad (4.15)$$

Similarly to equation (4.12), the optimum number of sections will be given by the following:

$$N_{bus} = \sqrt{\frac{0.4 R_w \left(C_w + C_p\right) L_{bus}{}^2}{t_{inv}}} \quad (4.16)$$

From equation (4.15) one can determine how much parasitic capacitance can be added to a bus wire before D_{bus} exceeds one clock cycle for a specific wire length in successive technology nodes, assuming the clock cycle to be 15FO4. The value of C_p can be considered as a metric for the scalability of a bus-based system as it relates to how many IP blocks can be appended to a bus before the delay exceeds one clock cycle. Decreasing bus wire length increases the value of admissible C_p, but the physical size of IP blocks will limit the scaling down of bus wire lengths. On the other hand, if bus wire lengths are increased, then wire capacitance will dominate and result in decreasing the allowable C_p and hence the number of IPs possibly appended to the bus. In Fig. 39 we illustrate the effect of bus wire length on C_P for different technology nodes. From the latter, the bus driving capability decreases exponentially with bus wire length.

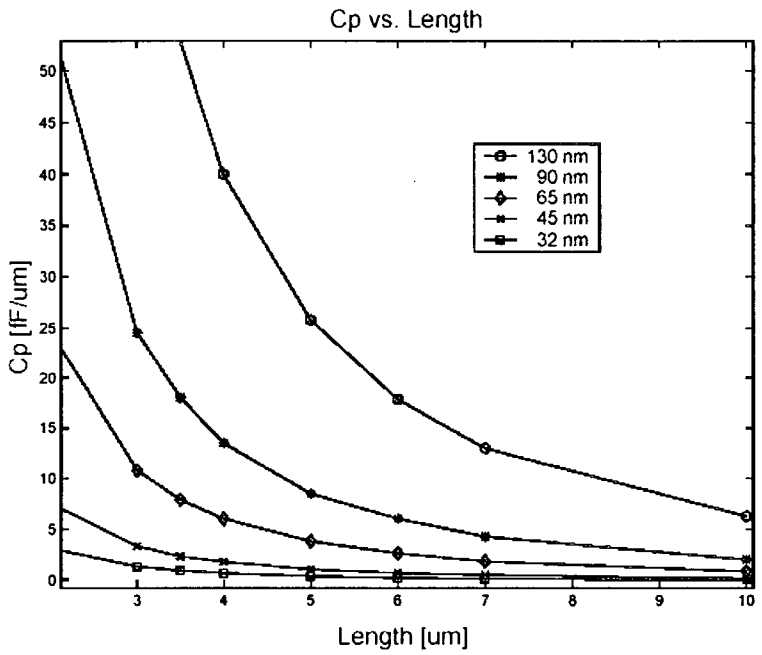

Fig. 39: Variation of C_p with bus wire length.

Fig. 39 illustrates the scalability problem associated with bus-based SoCs. For a fixed bus length, there is an upper limit on the parasitic capacitance (due to attached IP blocks) that can be accommodated if the bus delay is to be less than one clock cycle. As a result, there is a corresponding upper limit to the number of IPs that can be connected to a bus. Furthermore, in order to meet such delay requirements, the value of allowable parasitic capacitance decreases exponentially with bus length. As a result, the number of IP cores that can be added to the bus decreases.

However, due to heterogeneous nature of constituent IP cores in a SoC (embedded processors, DSPs, MPEG decoders, memories etc.), it is not possible to quantify the number of IPs that can be connected to the bus a priori. By knowing C_p and the types of IPs that need to be integrated for a particular application, we are able to determine whether timing closure is achievable when connecting these IPs to a bus.

If the $15FO4$ constraint on the clock cycle is relaxed and thereby increased, then the permissible values of C_p and hence the number of attached IP blocks also increases as shown in Fig. 40. This implies that by stretching the clock cycle, more IP blocks can be added to the bus at the cost of overall system speed degradation.

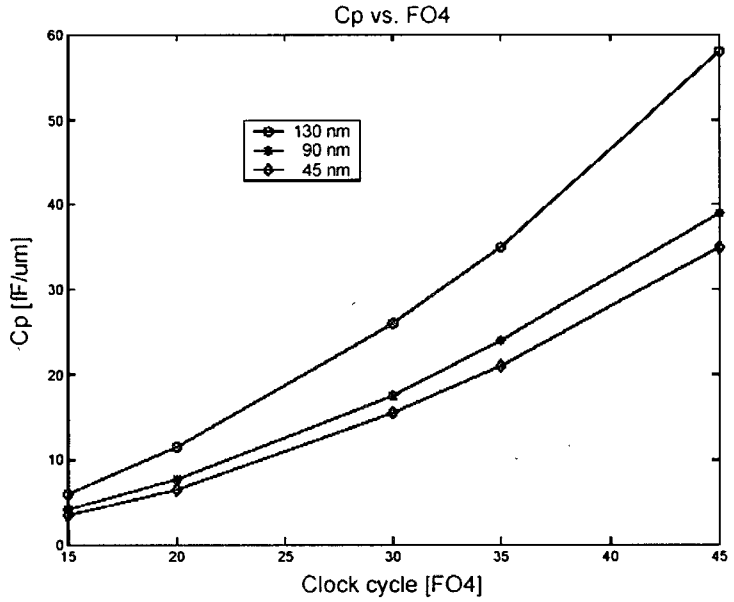

Fig. 40: Variation of C_p with clock cycle for different technology nodes.

The length of bus wires is difficult to predict in early stages of the design cycle. Hence, typically, system-level timing closure can only be reached post-layout and after several iterations.

In contrast, in the case of a networked SoC, the system size does not imply any extra loading on the inter-switch wire. As a result, variations in system size have little effect on the achievable clock cycle. That is, the inter-switch wire delay is largely insensitive to system size and only depends on the levels of the switches in the butterfly fat-tree architecture.

4.5 Summary

This chapter specifically analyzes global wire delays in a new switch-based interconnect architecture for future generations of SoCs. The butterfly fat-tree architecture was assumed as a system-level interconnect template. As this is a highly structured and regular architecture, the inter-switch wire delay can be estimated accurately, in initial phases of the design cycle. It was shown that it is possible to constrain this delay to be within one clock cycle, where the latter is, in turn, dictated by the technology dependent parameter limit governed by *15FO4*.

The delay in a bus-based SoC depends on the number of connected IP blocks. To further quantify this dependency, we proposed the parasitic capacitance, C_P , as a metric, which, in turn, is directly proportional to the number of IPs attached to a bus. indicated upper limits on the value of C_p, and therefore on the number of IPs, for different forthcoming technology nodes, and showed how these limits decrease exponentially against increases in bus wire length.

Looking forward in time, where numerous (hundreds or thousands) IP blocks consisting of 50-100K gates will need to be integrated, single bus-based interconnect templates will face serious limitations. We envisage that multiple forms of buses connected through a hierarchical architecture will ultimately converge to some form of network as the one proposed and analyzed here. As a result, we propose that future design processes start with a network architecture in mind. This will allow for better interconnect delay predictions. The ultimate effect will be the shortening of the design cycle and a reduced number of iterations.

5 Chapter V

Intra-Switch Delay Analysis

Together with inter-switch delay, the intra-switch delay component dictates the performance of any interconnect template, with respect to the maximum achievable clock rate. In this chapter we provide a detailed analysis of the intra-switch delays by using the method of logical effort. First, we consider the pipelined nature of the switch and explain what are the factors governing each pipeline stage. Then, we develop delay models for each pipeline stage based on the detailed gate-level design of the blocks involved in corresponding stages. The delay numbers are provided in technology independent units of FO4, thus lending an insight of what the effect of technology evolution will be on the performance of the BFT architecture coupled with wormhole routing.

5.1 Introduction to Logical Effort

The method of logical [21][24] effort is an easy way to estimate delay in CMOS circuits. It is founded on a simple model of the delay of a single CMOS logic gate. The model accounts for the delays caused by the capacitive load driven by the logic gate and for the topology of the logic gate. As the load increases, the delay also increases, but delay also depends on the logic function of the gate. Inverters, which perform minimum logical processing of a signal, drive loads best and are often used as amplifiers to drive large capacitances. Logic gates that compute other functions require more transistors, some of which are connected in series, making them poorer than inverters at driving

currents. The method of logical effort quantifies these effects to simplify delay analysis for individual logic gates and multistage logic networks. The complete method for a multistage logic path involves two steps: the first step is the determination of the optimum number of stages in the path, and the second step is delay calculation (with gate sizing as a side effect, but most important from a designer's perspective). In the following, we will give the basics of logical effort method, used further to analyze the delays in the pipeline stages of switches in BFT architecture.

The effect of a particular fabrication process is isolated by expressing delays in terms of a basic delay unit t_{inv} particular to that process. t_{inv} is the delay of an inverter driving an identical one with no parasitics.

The delay incurred by a logic gate is comprised of two components: a fixed part called the *parasitic delay* (p) and a part that is proportional to the load the gate is driving, called the *effort delay* or *fan out* delay (f). The total delay, measured in units of t_{inv} , is the sum of the fan out and parasitic delays:

$$d = f + p \tag{5.1}$$

The fan out portion of the delay, f, is characterized by two terms: the *logical effort (LE)* captures the properties of the logic gate, while the fan out, also called *electrical effort*, (FO) characterizes the load. The fan out portion of the delay is the product of these two factors:

$$f = LE * FO \tag{5.2}$$

The logical effort (LE) expresses the effect of the gate's topology on its ability to produce output current. The electrical effort FO describes how the electrical environment of the

logic gate affects performance and how the size of the transistors in the gate determines its load driving capability. FO is defined as:

$$FO = \frac{C_{out}}{C_{in}} \tag{5.3}$$

where C_{out} is the capacitance that loads the output of the logic gate and C_{in} is the capacitance presented by the input terminal of the logic gate.

Combining Eq. (5.1) and (5.2) we obtain the basic equation that models the delay through a single logic gate, in units of t_{inv}:

$$d = LE * FO + p \tag{5.4}$$

The logical effort of a gate is defined as the number of times worse it is at delivering output current than would be an inverter with identical input capacitance. It is important to note that the logical effort of a gate does not depend on the size of the gate, but only on its topology. There are two options to calculate the logical effort of a gate:

1. As the ratio between the input capacitance of that gate and the input capacitance of an inverter that produces the same output current;
2. As the ratio between the delay of the gate and the delay of an inverter with the same input capacitance.

In most CMOS processes, the PMOS transistor width is larger than the NMOS transistor width to account for different carrier velocity, when circuits are designed for equal rise and fall times. $\gamma = W_p/W_n$ is the ratio of PMOS to NMOS width in an inverter for equal conductance (equal rise and fall times). In our analysis, for simplicity, we will assume $\gamma = 2$. Under these assumptions, the logical effort of different logic gates can be calculated and it is given in Table 6.

Table 6: Logical effort and parasitics of usual logic gates

Gate type	Logical effort	Formula	Parasitic
NAND (n inputs)	Total	$\frac{n(n+\gamma)}{1+\gamma}$	np_{inv}
	Per input	$\frac{(n+\gamma)}{1+\gamma}$	
NOR (n inputs)	Total	$\frac{n(1+n\gamma)}{1+\gamma}$	np_{inv}
	Per input	$\frac{(1+n\gamma)}{1+\gamma}$	
Multiplexer (n inputs)	Total	$4n$	$2np_{inv}$
	d (data), s (select)	2, 2	
XOR, XNOR (n inputs)	Total	$n^2 2^{n-1}$	$n2^{n-1} p_{inv}$
	Per bundle	$n2^{n-1}$	
C-element (n inputs)	Total	n^2	np_{inv}
	Per input	n	

For a path comprised of multiple logic gates, the logical effort along the path, called *path logical effort* (LE_p) is calculated as the product of *LE* of all gates along the path:

$$LE_p = \Pi LE_{gate} \tag{5.5}$$

The *path effective effort* or *path fan out* FO_p can be defined as the ratio of the load capacitance of the last gate of the path to the input capacitance of the first gate:

$$FO_p = \frac{C_{load}}{C_{g1}} \tag{5.6}$$

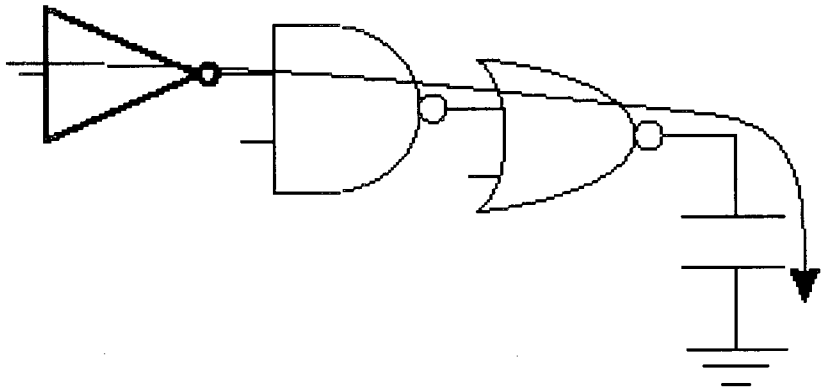

Fig. 41: A critical path with three stages.

When fan out occurs at the output of a node and some of the available drive current is directed along the analyzed path, and some branches out of the path, to account for logical fan out within the logical path, we use *branching factor (BF)* of a logic gate:

$$BF = \frac{C_{on-path} + C_{off-path}}{C_{on-path}} \tag{5.7}$$

The *path branching effort* BE_p is defined as the product of the branching factors along that path:

$$BE_p = \Pi BF_{gate} \tag{5.8}$$

Finally, the total path effort PE can be defined as:

$$PE = FO_p \Pi BF_{gate} LE_{gate} = FO_p BE_p LE_p \tag{5.9}$$

The gate effort that minimizes the path delay, called *stage effort* (*SE*), is calculated according to:

$$SE = \sqrt[N]{PE} \tag{5.10}$$

where N is the number of stages (gates) on that path. The minimum delay through the path can therefore be calculated as:

$$D = N * SE + \Sigma P \tag{5.11}$$

That is, the delay is the sum two components: a fan out related delay and a parasitic delay. In the following subsections, we will use this methodology to calculate the minimum achievable delay of the three pipeline stages of the switches in the BFT interconnect network. We will make the assumption that $\gamma = 2$ to simplify the analysis.

5.2 Intra-Switch Pipeline Stages and Delay Analysis

The switch has six ports, four children ports denoted by *C0, C1, C2* and *C3* respectively, and two parent ports, denoted by *P0* and *P1*, respectively as shown in Fig. 42.

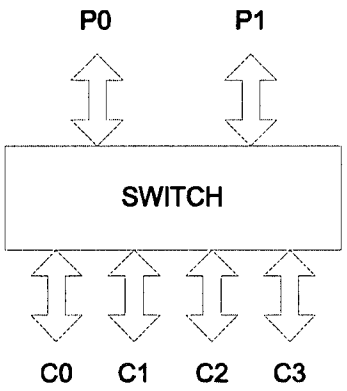

Fig. 42: Switch with 6 ports.

In order to have a considerably high throughput, we use a virtual channel switch, where each port of the switch has four parallel buffers [5]. The different components of the

switch are shown in Fig. 43. It mainly consists of two arbiters, a routing block and a chain of multiplexers/demultiplexers.

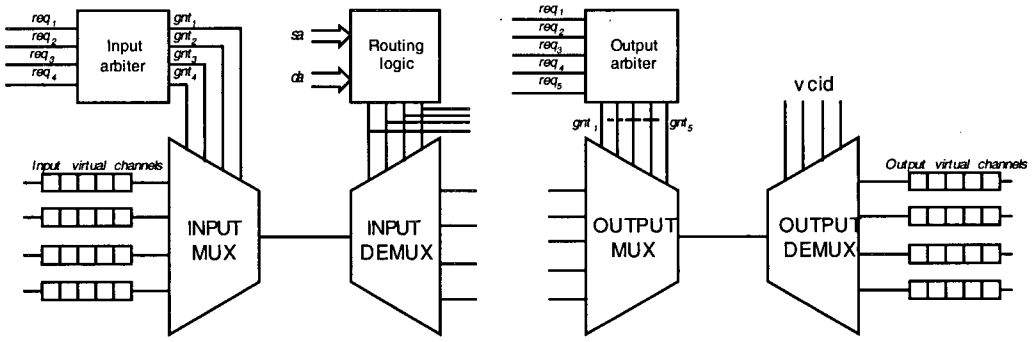

Fig. 43: Block diagram of a switch port.

Each physical input port has more than one virtual channel, uniquely identified by its virtual channel identifier (VCID) [25]. Flits may simultaneously arrive at more than one virtual channel. As a result, an arbitration mechanism is therefore necessary to allow only one virtual channel to access a single physical port. As there are four virtual channels corresponding to each input port, we need a 4:1 arbiter at the input. Similarly, flits from more than one input port may simultaneously try to access a particular output port. Consequently, on the output side, we need a 5:1 arbiter since among the six ports of the switch, any five may try to access a particular output port [4]. The routing logic block determines the output port to be taken by an incoming flit.

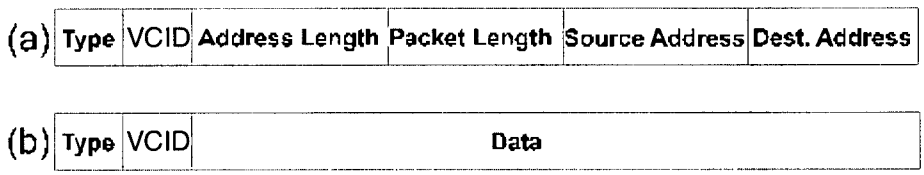

Fig. 44: Flit structure (a) Header flit; (b) Data and Tail flits.

The packets consist of a header flit, one or more data flits and a tail flit. The header, data, and tail flit structures are as shown in Fig. 44. The first field denotes the flit type, namely *header, data* or *tail.* The second field contains the virtual channel identifier (VCID). The third field denotes the address length, which is dependent on the number of SoC IP blocks. The fourth field contains packet length information, i.e., the number of flits in the corresponding packet. The next two fields give source and destination addresses. The flit length is constant but the total number of flits in a packet will vary according to the contents of the packet length field. One packet will consist of a sequence of flits starting with a header flit, followed by a set of data flits (this set may be void eventually) and ended by a tail flit.

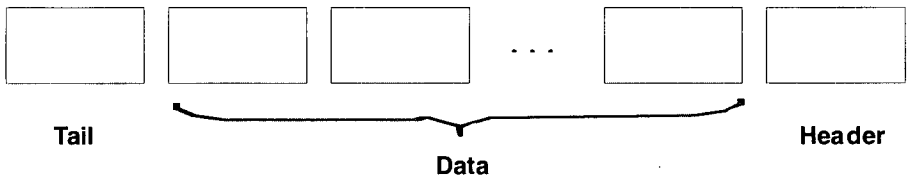

Fig. 45: Packet consisting of header, data and tail flits.

The operation of the switch consists of one or more processes depending on the nature of the flit. In the case of a ***header*** flit, the sequence of the processes is: (1) ***Input Arbitration;*** (2) ***Routing;*** and (3) ***Output Arbitration***. In the case of ***body*** flits, ***Switch Traversal*** replaces the routing process as the routing decision based on the header information is maintained for the subsequent body flits. The blocks involved in the input arbitration process are the 4:1 arbiter and the input multiplexer; similarly, the blocks in the output arbitration process are the 5:1 arbiter and the output multiplexer. The routing process is performed by the combination of the routing block and the input demultiplexer. The switch traversal mainly involves the chain of four multiplexers and demultiplexers.

Each of these processes occurs in different clock cycles. From a signal propagation point of view, each process is a pipeline stage on the critical path of the data flow. Consequently, we need to calculate the delays incurred in these processes.

The arbiter circuit mainly consists of a priority matrix, which stores the priorities [26] of the requesters and grant generation circuits, granting resources to requesters. The matrix arbiter stores priorities between n requestors in a binary n-by-n matrix. Each matrix element *[i, j]* records the binary priority between each pair of inputs. For example, suppose requestor i has a higher priority than requestor j, then the matrix element *[i, j]* will be set to 1, while the corresponding matrix element *[j, i]* will be 0. A requestor will be granted the resource if no other higher priority requestors is bidding for the same resource. Once a requestor succeeds in being granted a resource, its priority is updated and set to be the lowest among all requestors.

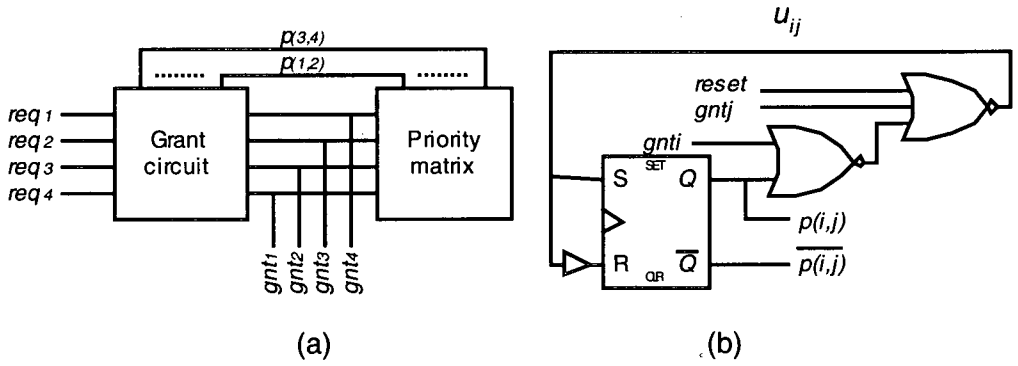

Fig. 46: (a) Block diagram of an arbiter; (b) one element of the priority matrix.

Fig. 46 shows the block diagram of the arbiter, consisting of the grant generation circuit and the priority matrix. As for the input side, there are four virtual channels competing for the resources and the grant circuit generates four grant signals denoted by gnt_1 to gnt_4. The Boolean expressions for the grant signals are given as follows:

$$gnt_1 = req_1\left(\overline{req_2} + p_{12}\right)\left(\overline{req_3} + p_{13}\right)\left(\overline{req_4} + p_{14}\right)$$
$$gnt_2 = req_2\left(\overline{req_1} + \overline{p_{12}}\right)\left(\overline{req_3} + p_{23}\right)\left(\overline{req_4} + p_{24}\right)$$
$$gnt_3 = req_3\left(\overline{req_1} + \overline{p_{13}}\right)\left(\overline{req_2} + \overline{p_{23}}\right)\left(\overline{req_4} + p_{34}\right)$$
$$gnt_4 = req_4\left(\overline{req_1} + \overline{p_{14}}\right)\left(\overline{req_2} + \overline{p_{24}}\right)\left(\overline{req_3} + \overline{p_{34}}\right)$$

where req_i is the request signal from virtual channel i and p_{ij} denotes the priority of virtual channel i over virtual channel j, with $i, j \in [1,4]$.

We use the method of logical effort to determine the delay involved in the input arbitration process. The delay will be given in terms of *FO4*, with *FO4* defined as the delay of an inverter driving four identical ones. The critical path of the input arbiter circuit is shown in Fig. 47.

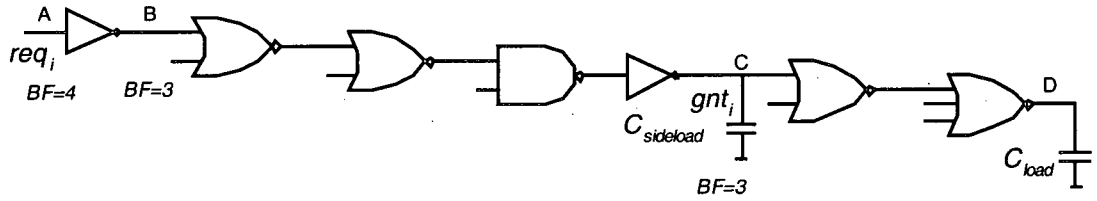

Fig. 47: Critical path of the input arbiter

From Boolean expressions of grant signals, it is clear that req_i and $\overline{req_i}$ fan out to four and three places in the grant circuits and therefore the branching factors at points A and B are four and three, respectively. The grant signals control a multiplexer to select a specific virtual channel. Considering an 8-bit data bus, these grant signals are the control inputs of eight multiplexers as shown in Fig. 48.

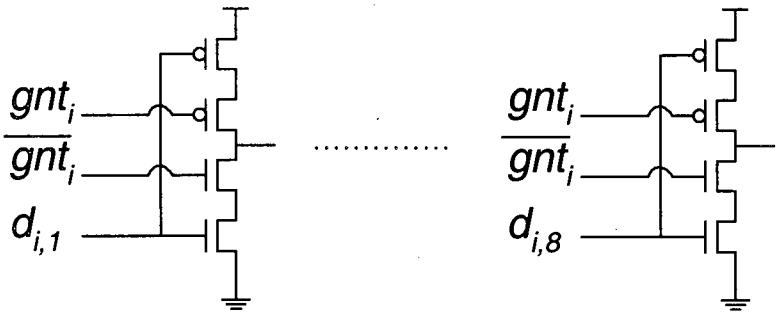

Fig. 48: Grant signals as control inputs of the mux.

This will give rise to a side load capacitance equivalent to $C_{sideload}$ = (8+8/3) times the minimum size inverter input capacitance at point C according to Fig. 47. As each grant signal splits to three elements of the priority matrix we have a branching factor of three at point C. From Fig. 46(b), it is evident that the signal u_{ij} is driving a NAND gate and inverter considering that the flip-flop consists of a pair of cross-coupled NAND gates. Consequently, the load capacitance at point D will be equivalent to three minimum-sized inverter gate capacitances. The load capacitance at the point C is considered as a side load. All the capacitances are expressed relative to the input capacitance of a minimum sized inverter.

We use the notations in Table 7 in determining the delay.

Table 7: Logical Effort – Summary of parameters [21][24]

Term	Expression
Logical Effort of a gate	LE_i
Logical Effort of a path	$LE_p = \Pi LE_i$
Fan Out	$FO = C_{out} / C_{in}$
Branching Factor	BF_i
Branching Effort	$BE = \Pi BF_i$
Path Effort	$PE = (LE_p)(BE)(FO)$
Stage Effort	$SE = (PE)^{1/N}$, N = *No. of stages in the path*
Parasitic Delay of a gate	P_i *(Intrinsic delay due to its own internal capacitance)*
Parasitic Delay of the path	$P = \Sigma P_i$
Delay of a path	$D = (SE)(N) + P$

The values of the logical efforts and parasitic delays of the gates used in our design are shown in Table 8.

Table 8: Logical Effort and parasitic delays of the relevant gates [24]

Gate Type	Logical Effort	Parasitic Delay [t_{inv}]
Inverter	1	1
NOR2	5/3	2
NOR3	7/3	3
XOR2	2	4
NAND2	4/3	2
MUX (Fig. 25)	2	2

From Table 7, the determination of the delay of a path is straightforward. It mainly involves determining the optimal stage effort. In the case of the input arbiter circuit as shown in Fig. 47, in addition to the output load C_{out} there is a side load at point C. Consequently, this amounts to two stage efforts, one characterizing the circuit behaviour from point C to the output load, and the other from the input to point C. To determine the first one, we eliminate the side load and find SE= 2.8 according to Table 6.

Considering SE = 2.8 and C_{load} = 3, we calculate the input capacitances at the point D as

$$C_D = \frac{LE_{3inputsNOR} \times BF_D \times C_{load}}{SE} = 2.5 \qquad (5.12)$$

Considering C_D as the load capacitance, we can calculate the input capacitance at point C. Using a similar equation as (4.5) we get C_C = 1.49. Consequently, in the calculation of the SE of the first 5 stages, we consider the total load capacitance at point C as

$$C_{load,C} = C_{sideload} + 1.49 = 12.16 \qquad (5.13)$$

Again, following Table 7 with a fan out of 12.16 yields the stage effort of the first five stages to be SE = 4.38.

The parasitic delay of the path is $P=13t_{inv}$ (according to Tables 7 and 8). Combining both stage efforts and the parasitic delay we get the delay of the input arbiter as

$$D_{input\_arbiter} = 5\times 4.38 + 2\times 2.8 + 13 = 40t_{inv} = 8FO4 \qquad (5.14)$$

The delay of the input multiplexer will be given as

$$D_{input\_MUX} = 2 + 2 = 4t_{inv} = 0.8FO4 \qquad (5.15)$$

Combining the latter two, the delay in the input arbitration process is

$$D_{input\_arbitration} = D_{input\_arbiter} + D_{Input\_MUX} = 8.8FO4 \qquad (5.16)$$

The first step in the implementation of the routing logic involves the comparison (XOR) of the source and destination addresses taking the most significant ($M= (log_2N - 2^l)$) bits, where N is the number of functional IP blocks in the system and l denotes the level number of the switch. Subsequently, the result of the comparison is checked, i.e., whether a "1" results from the XOR operation. As a result of these two logical operations the critical path of the routing block is as shown in Fig. 49.

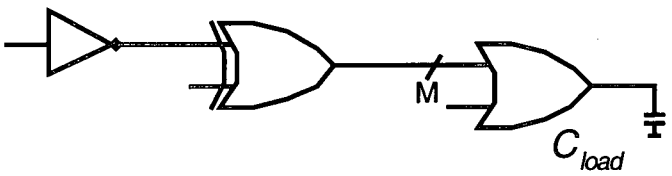

Fig. 49: Critical path of the routing block.

The final M-input OR gate of Fig. 49 is modeled as a tree of 2-input NOR gates [24], also indicated in Fig. 25. If k is the number of levels in the NOR tree then $2^k = M$. The logical effort of this M- input OR tree is

$$LE_{OR}(M) = (LE)_{NOR2}^{\log_2 M} = M^{\log_2 \frac{5}{3}} = M^{0.7} \tag{5.17}$$

The output of the routing logic block fans out to an input demux control inputs and to the input of a 5:1 arbiter. According to the circuits shown in Figs 47 and 48, the output load of the routing block will be equal to C_{load} = (8+8/3+1)=11.67 times the minimum size inverter input capacitance and the fan out will be 11.67. Hence, the stage effort is given as

$$SE_{routing} = \left(2 \times M^{0.7} \times 11.67\right)^{\frac{1}{3}} \tag{5.18}$$

and the delay of the routing block will be

$$D_{routing\_block} = 3 \times \left(2 \times M^{0.7} \times 11.67\right)^{\frac{1}{3}} + (\log_2 M)(P_{NOR2} + P_{INV}) + P_{inv} + P_{XOR2} \tag{5.19}$$

The parasitic terms P_{NOR2}, P_{INV}, P_{XOR2} are equal to 2, 1 and 4 respectively, according to Table 7.

By adding the delay corresponding to the demux to the delay of the routing block, we get the total delay associated with the routing process expressed by

$$D_{routing} = D_{routing\_block} + D_{Input\_DEMUX} \tag{5.20}$$

$D_{routing\_block}$ will depend on M, which in turn depends on the system size. From Table 2, the value of M varies from 7 (in 130 nm node) to 11 (in 32 nm node). Consequently, $D_{routing\_block}$ varies from 4 *FO4* (130 nm) to 6 *FO4* (32 nm). As a result, $D_{routing}$ varies from 5 *FO4* (130nm) to 7 *FO4* (32 nm).

Similarly to the input arbitration process, the delay involved in the output arbitration can be expressed as

$$D_{output\_arbitration} = D_{output\_arbiter} + D_{output\_MUX} = 10.3FO4 \quad (5.21)$$

For the switch traversal process the delay is computed considering the chain of four input and output muxes and demuxes. The output of the final demux drives the latches of the virtual channels as shown in the Fig. 43. Considering that the latches consist of a pair of cross-coupled NAND gates, the load capacitance is equivalent to two minimum-sized inverter gate capacitances, and hence, the fan out will be 2. Following the same method as in the case of the input arbiter we get the stage effort (*SE*) of this mux-demux chain to be 2.38. Finally, the delay of the switch traversal process can be expressed according to the following:

$$\begin{aligned} D_{switch\_traversal} &= (4 \times SE) + P_{Input\_MUX} + P_{Input\_DEMUX} + \\ & P_{Output\_DEMUX} + P_{Output\_MUX} = 17.5t_{inv} = 3.5FO4 \end{aligned} \quad (5.22)$$

The parasitic terms P_{Input_MUX}, P_{Input_DEMUX}, P_{Output_DEMUX} and P_{Output_MUX} are all equal to 2, according to Table 7 and Fig. 48.

We developed a VHDL model for the switch and synthesized it using Synopsys' synthesis tool in CMOS 0.18μ technology. We compared results obtained from the theoretical analysis with those given by Synopsys' tool. These comparisons are reported in Table 9.

Table 9: Comparison of Delays: Calculated vs. Synopsys' tool-generated

Process	Delay (LE analysis)	Delay (Synopsys' Tool)
Input Arbitration	8.8 *FO4*	9.1*FO4*
Routing	5*FO4*	6.2*FO4*
Output Arbitration	10.3*FO4*	8.6*FO4*
Switch Traversal	3.5*F04*	5*FO4*

Table 9 shows that the delays estimated by our logical effort analysis match closely those obtained from Synopsys' tools. However, it should be taken into account that the numbers on the second column reflect a full custom design approach (with respect to gate sizing) while the synthesis tool used did not have the option of modifying the gate size to optimize the delay of paths depending on the load. Our detailed analysis shows an improvement over the existing highly pipelined virtual channel routers [27]. This indicates that the delay associated with each processes involved with the operation of the switch is well below the limit of 15*FO4* and can therefore be driven by a clock with this period or less.

5.3 Summary

This chapter analyzes the delays associated to the pipeline stages involved in the switch operation. We have shown that there are three pipeline stages on the signal critical path. For header flits, these stages are: input arbitration, routing, and output arbitration. For data (body and tail) flits, routing is replaced by switch traversal. For each of these processes, the corresponding delays were calculated using the method of logical effort. The results indicate that all delays of the switch pipeline stages are within 15 *FO4* delay units, and, consequently, the switch can be driven by a clock cycle with this period. The principal conclusion of this chapter is that the switches in the BFT interconnect template will not be a bottleneck for data transmission rate, as the maximum rate at which functional IP block will generate output signals corresponds to 15*FO4* delay units.

6 Chapter VI

Conclusions and Future Work

6.1 Summary

In this thesis, we have investigated the timing characteristics of an on-chip, switch-based interconnect template, namely the butterfly fat-tree. The underlying hypothesis was that a paradigm change will happen in the SoC design methodology, where multiple, heterogeneous IP blocks, consisting of around 100K gates, will be integrated together using a structured interconnect template. The functional IP blocks will exchange data in the form of packets, divided into smaller flow control units (flits) of constant size. From one functional IP to another, the flits will traverse multiple pipeline stages, according to the routing algorithm that is implemented (least common ancestor determination and turnaround routing in the BFT graph). Each pipeline stage will be represented by either an inter-switch wire or a process in a switch. Depending on the nature of the flit (header or data), the processes that will be executed by the switch will be:

- for header: input arbitration, routing, output arbitration;
- for data: input arbitration, switch traversal, output arbitration.

Then, the delays associated to each of the pipeline stages were determined. For inter-switch wires, we developed a wire length model based on the layout of the interconnect template, and calculate the delay of each of these segments. When necessary, it was pointed out which of the inter-switch segments need to be optimized by inserting buffers. The analysis was extended for successive technology nodes using technology dependent parameters such as copper resistivity, gate oxide capacitance, fringing capacitance of metal tracks.

For the active devices of the interconnect network, i.e., the switches, we determined the delays involved in their pipeline stages using the method of logical effort. By expressing the delays of the processes in technology independent units, it was possible to isolate the effect of technology scaling.

The analysis shows that the individual delay of each of the pipeline stages can be made to fit within the limit of $15FO4$ suggested by ITRS as appropriate for the clock cycle of high performance SoC. Given the likelihood of having multiple clock domains in a large SoC, our understanding of the $15FO4$ rule is that data exchange between any two functional IP blocks should be possible at this rate, with the penalty of an increased setup time required by the header flit to reach the destination node. But after this setup time, once a header arrives at the destination, the incoming packets can be absorbed at a clock speed governed by $15FO4$ delay units, required for signals to traverse one pipeline stage.

The non-scalability of buses as on-chip interconnects was quantified with respect to the achievable clock cycle. It was shown that the amount of capacitance (corresponding to the attached functional IP blocks) that can be added to a bus is extremely limited, under the clock cycle constraints mentioned above. Also, a simple method of evaluating the feasibility of buses for specific on-chip applications was presented: it has been shown that the total capacitance added to the bus has to be less than a predefined, technology specific value.

The method can be adapted for any on-chip interconnect topology, with minor changes involved by the specific design parameters such as topology, number of virtual channels, arbitration scheme.

6.2 Contribution of the Work

The goal of this research is to analyze and characterize a specific on chip interconnect architecture with respect to timing. Demonstrating the issues that affect the system level timing is a key component in designing high-speed, high-performance on-chip data transmission mechanisms. We assume that the interconnect network should not limit the system's speed of operation. A methodology was developed to evaluate the maximum achievable clock rate of the SoC communication fabric. The basic question that this thesis answers is the following: given a certain topology of an on-chip network, what is the minimum clock rate at which it is possible to move data across the chip? In our case, the topology is the butterfly fat-tree and the clock rate is $15FO4$ delay units as specified in ITRS 2001 document. The movement of signals from one IP block to another is pipelined, and the pipeline stages consist of both passive (metal wires) and active (intelligent switches) devices. Analyzing the delay of each stage and placing the work in the context of future technology nodes, it is possible to accurately quantify the raw performance (measured in maximum achievable clock rate) of a networked SoC. We also quantified the non-scalability of bus-based systems on chip, and showed a method to decide when the transition from a bus-based to a network-centric design style is required.

The innovative contributions of this thesis are summarized as follows:

- Development of a wire-length model for the BFT (butterfly fat-tree) topology and associated delays for future technology nodes.
- Development of a technology independent delay model for the active devices (switches) and the corresponding pipeline stages.

- Provision of a quantitative formulation for the non-scalability of bus-based systems and indications for how designers should decide on the transition point between bus-based and network-based design style.

6.3 Future Work

6.3.1 Power Analysis

This work is an important step toward the complete realization, characterization and evaluation of the on-chip interconnection networks for large SOCs. After characterizing the BFT interconnect template from a timing/delay point of view, the next logical step is to analyze and parameterize its power dissipation. Based on the pipeline stages on the data path of the flits from source to destination, four components of power dissipation can be identified:

- *wire power:* power dissipated in driving messages along the inter-switch wires;
- *arbitration power:* Power is dissipated during the input/output arbitration steps;
- *routing power:* power required to make a routing decision for the header flit;
- *switching power:* power dissipated in the switch traversal process by the data flits.

Power dissipation in the interconnect network can be viewed in two ways: a) a *message-centered* view and b) a *switch-centered* view. The message-centered view focuses on the amount of power dissipated in driving a message through the network, from a source node to a destination node. Consider a message spread across a number of D links. In a steady state view of the network and at a particular moment of time, the power required to drive the message is equal to the power required to make the routing

decision, P_r, and the power required to drive the data flits through the network. Since the message is spread over D links and each link is concurrently switching, the message-centered power can be expressed as:

$$P_{msg} = P_r + D(P_S + P_W) \tag{6.1}$$

where P_r is the power required for routing decision, P_s is the power dissipated during switch traversal, and P_w is the power dissipated by a flit traversing a wire segment

A *switch-centered* view focuses on power dissipated in switch per message. The power per switch is the power it takes to process a message. This results in:

$$P_{switch} = (P_r / D) + P_S + P_W \tag{6.2}$$

These different views capture power dissipation in the network at two different levels. The message-centered view of power dissipation is helpful when analyzing the network from an application perspective. The switch centered view is useful when the network is analyzed from a technology standpoint.

6.3.2 Interfacing

Another important problem in this NoC project is the interfacing between the functional IP blocks and the communication template. To solve this problem, and considering the heterogeneous nature of the functional IP blocks in the SoC, we shall start from the assumption that all the cores are OCP-compatible. This is not a very restrictive requirement, since most of the existing digital blocks can be converted to the OCP protocol [28]. An example of OCP compatible cores and their corresponding interfaces is given in Fig. 27. A first step in interfacing the IP blocks is making them OCP compatible. The result of this is the fact that we will have to deal with a clearly

specified, standard set of signals corresponding to the master/slave instances of the OCP interface. This set of signals will be packetized by a second interface, which will sit between the OCP instances and the communication fabric. The interface will have two functions:

1: injecting/absorbing the flits leaving/arriving at the functional IP blocks;

2: packetizing/depacketizing the signals coming from/reaching to OCP compatible cores in form of messages/flits;

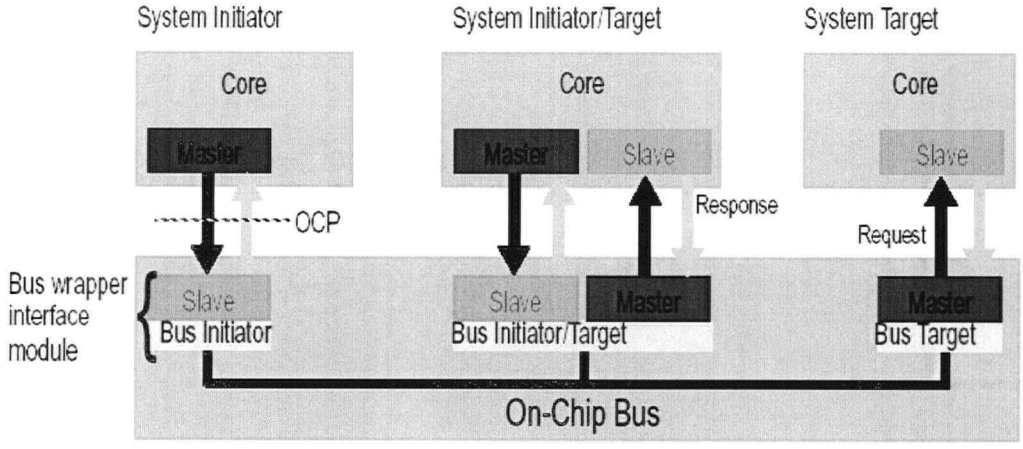

Fig. 50: Example of OCP interfaces [28].

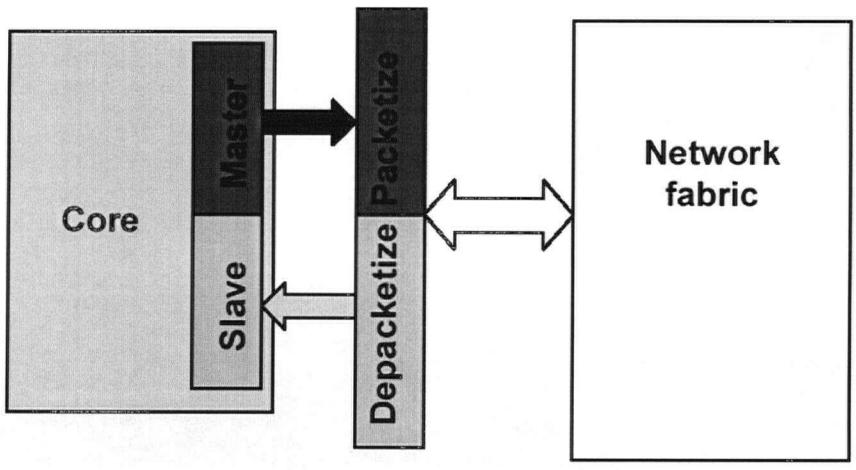

Fig. 51: Packetization/depacketization interfaces.

REFERENCES

[1] W. J. Dally, B. Towles, "Route Packets, not Wires: On-Chip Interconnection Networks", *Proceedings of DAC 2001*, pp.684-689.

[2] P. Guerrier, A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections", *Proceedings of Design, Automation and Test in Europe Conference and Exhibition 2000*, pp. 250 -256.

[3] http://public.itrs.net/Files/2002Update/Home.pdf.

[4] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh, "Design of a Switch for Network on Chip Applications", *Proceedings of ISCAS*, pp. 217-220.

[5] P. Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, "Architecture Evaluation for Communication-Centric SoC Design", submitted to *ISCAS 2004*.

[6] S. Kumar, et al, " A Network on Chip Architecture and Design Methodology," *Proceedings of ISVLSI 2002*, pp. 117-124.

[7] A. Adriahantenaina, A. Greiner, "Micro-network for SoC : Implementation of a 32-port SPIN network" , *Proceedings of DATE 2003*, pp. 1128-1129.

[8] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh, "High-Throughput Switch-Based Interconnect for Future SoCs", *Proceedings of 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications*, June 30-July 2, 2003, Calgary, Canada, pp. 304-310.

[9] J. Duato, S. Yalamanchili, L. Ni, *"Interconnection Networks: An Enginering Approach"*, Morgan Kauffman, 2002.

[10] [AMBA Bus specification, http://www.arm.com.

[11] CoreConnect Specification, http://www-3.ibm.com/chips/products/coreconnect/.

[12] Wishbone Service Center, http://www.silicore.net/wishbone.htm.

[13] F. Petrini, M. Vanneschi, "k-ary n-trees: High Performance Networks for Massively Parallel Architectures", *Proceedings of the 11th International Parallel Processing Symposium, IPPS'97*, Geneva, Switzerland, April 1997, pp. 87-93.

[14] C. E. Leiserson, "Fat-Trees: Universal Networks For Hardware-Efficient Supercomputing", *IEEE Transactions on Computers*, October 1985, C-34(10):892-901.

[15] A. DeHon, "Compact, Multilayer Layout for Butterfly Fat-Tree", *Twelfth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2000)*, July 9-12, 2000, pp. 206-215.

[16] D. Sylvester, K. Keutzer, "Impact of Small Process Geometries on Microarchitectures in Systems on a Chip", Proceedings of the IEEE, Vol. 89, No. 4, April 2001, pp. 467-489.

[17] M.A. Horowitz, et al., "The Future of Wires", *Proceedings of the IEEE*, Volume: 89 Issue: 4, April 2001 pp. 490–504.

[18] P. Wielage, K. Goossens, "Networks on Silicon: Blessing or Nightmare?", *Euromicro Symposium on Digital System Design*, Dortmund, Germany, September 2002, pp. 196-200.

[19] Design and Reuse website, http://www.us.design-reuse.com/sip/.

[20] P. Magarshack, P.G. Paulin, "System-on-Chip Beyond the Nanometer Wall", *Proceedings of DAC'03*, June 2-6, 2003, Anaheim, USA, pp. 419-424.

[21] D. A. Hodges, H. G. Jackson and R. Saleh, "*Analysis and Design of Digital Integrated Circuits*", Third Edition, McGraw-Hill, 2003.

[22] K. C. Saraswat, et al., "Technology and Reliability Constrained Future Copper Interconnects – Part II: Performance Implications," *IEEE Transactions on Electron Devices*, Vol. 49, No. 4, April 2002 pp. 598-604.

[23] C. Yuan, T. Trick, "A Simple Formula for the Estimation of the Capacitance of Two-Dimensional Interconnects in VLSI Circuits", *IEEE Electron Device Letters*, vol. EDL-3, 1982, pp. 391-393.

[24] I. Sutherland, B. Sproull and D. Harris, *"Logical Effort: Designing Fast CMOS Circuits"*, Morgan Kaufmann, 1999.

[25] W. J. Dally, "Virtual-Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, March 1992, pp. 194-205.

[26] H.Wang, L-S Peh and S. Malik, "A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers", *Proceedings of the 10th Symposium on High Performance Interconnects (Hot Interconnects)*, Stanford, CA, August 2002, pp. 21-27.

[27] L. Peh, W. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers", *The Seventh International Symposium on High Performance Computer Architecture*, Jan. 2001, pp. 255-266.

[28] Open Core Protocol, www.ocpip.org.

[29] R.I. Greenberg, Lee Guan, "An Improved Analytical Model for Wormhole Routed Networks with Application to Butterfly Fat-Trees ", *Proceedings of the 1997 International Conference on Parallel Processing*, pp.: 44 – 48.

[30] J. Rabaey, A. Chandrakasan, B. Nikolic, *"Digital Integrated Circuits - A Design Perspective (2nd Ed)"*, Prentice Hall, 2003.

[31] C. Grecu, P. Pande, A. Ivanov, R. Saleh, "A Scalable Communication-Centric SoC Interconnect Architecture", to appear in the *Proceedings of ISQED 2004.*