

# DESIGNING SWITCHES FOR ROUTING IN CIRCUIT-SWITCHED TREES

A Thesis

Submitted to the Graduate Faculty of the  
Louisiana State University and  
Agricultural and Mechanical College  
in partial fulfillment of the  
requirements for the degree of  
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

by

Dinesh Prasad Venkat Rao  
B.E., University of Madras  
Chennai, India, 2002  
May 2006

## **Acknowledgements**

I would like to extend my deepest thanks to my major professors, Dr. Jerry Trahan, for providing me with constant guidance, provoking suggestions and inspiration throughout the entire period of my thesis work. Right from the time I started working on my thesis, he has been very helpful in organizing my ideas and putting together a very fine piece of work.

I express my gratitude to the members of my committee, Dr. Vaidyanathan Ramachandran and Dr. Ashok Srivatsava for their valuable suggestions and advice. Also, I would like to thank Stephen Lee Bishop for his help and guidance in using the cadence tool.

I dedicate this work to my family members for their prayers, encouragement and financial support. I would like to thank all my friends at LSU for their assistance, wishes, support and encouragement throughout my research, and also throughout the course of my graduate study at Louisiana State University.

## Table of Contents

<b>ACKNOWLEDGMENTS</b> .....	ii
<b>LIST OF TABLES</b> .....	v
<b>LIST OF FIGURES</b> .....	vii
<b>ABSTRACT</b> .....	viii
<b>1: MOTIVATION AND INTRODUCTION</b> .....	1
<b>2: BACKGROUND</b> .....	3
2.1 Introduction.....	3
2.2 Reconfigurable Computing .....	3
2.3 Self-Reconfigurable Gate Array.....	5
2.4 Circuit-Switched Tree .....	6
2.4.1 Based on Number of Destinations .....	8
2.4.2 Based on the Orientation of the Communications.....	9
2.4.3 Based on the Communication Width.....	9
2.4.4 Based on the Property of Incompatibility .....	10
2.5 Patterns of Communication Sets .....	11
<b>3: CONFIGURING CIRCUIT-SWITCHED TREES</b> .....	13
3.1 Configuration Basics for the CST .....	13
3.2 Structure of a CST Switch .....	13
3.3 Configuration of CST for a Width-1 Communication Set. ....	15
3.3.1 Well-Nested Communication Set.....	16
3.3.2 Multicast Communication Set.....	18
3.4 Configuration of CST for Width-w Communication Set.....	21
3.4.1 Internal Structure of a Switch for Width-w Communication Set.....	22
3.4.2 Procedure to Schedule and Construct Paths in Width-w Communication Set .....	23
3.4.3 Well-Nested, Width-w Communication Set .....	25
<b>4: IMPLEMENTATION</b> .....	26
4.1 Introduction .....	26
4.2 Design of Data Unit (DU) in a Switch .....	26
4.3 Design of a Switch for Well-Nested, Right-Oriented, Width-1 Communication Set.....	27
4.4 Design of a Switch for Multicast, Right-Oriented, Width-1 Communication Set.....	30
4.5 Design of a Switch for Well-Nested, Right-Oriented, Width-w Communication	

Set.....	33
4.6 Design of a Switch for Point-to-Point, Right-Oriented, Width-w Communication	
Set.....	37
4.7 Design of a Multi-Pattern Framework .....	38
4.8 Implementation of a CST with Four, Eight, and 16 PEs .....	40
<b>5: IMPLEMENTATION RESULTS .....</b>	<b>42</b>
5.1 Introduction.....	42
5.2 Implementation Environment .....	42
5.3 Implementation Results and Discussion .....	43
5.3.1 Area Analysis.....	50
5.3.2 Frequency Analysis.....	51
5.3.3 Power Analysis.....	52
<b>6: SUMMARY AND FUTURE WORK .....</b>	<b>54</b>
6.1 Summary .....	54
6.2 Future Work .....	54
<b>REFERENCES.....</b>	<b>54</b>
<b>VITA.....</b>	<b>57</b>

## List of Tables

Table 3.1: Status symbol function $f_s$ for a right-oriented, well-nested, width-1 communication set.....	17
Table 3.2: Configuration function $f_c$ for a right-oriented, well-nested, width-1 communication set.....	18
Table 3.3: Status symbol function $f_s$ for the right oriented, multicast, width-1 communication set.....	19
Table 3.4: Configuration function $f_c$ for a right oriented, multicast, width-1 communication set.....	21
Table 4.1: Notation for PE status.....	28
Table 4.2: LUT contents for status symbol function $f_s$ for well-nested, right-oriented, width-1 communication set (LUT1).....	28
Table 4.3: Connections determined by $f_c$ for well-nested, right-oriented,width-1 communication set (LUT2) .....	29
Table 4.4: Module- $f_s$ mapping for status symbol function $f_s$ for multicast, right-oriented communication set.....	31
Table 4.5: Connections determined by $f_c$ for multicast, right-oriented, width-1 communication set (LUT3) .....	32
Table 5.1: Results for a single switch with constraint on time (8-bit data width).....	44
Table 5.2: Results for a single switch with constraint on area (8-bit data width).....	44
Table 5.3: Results for three switches in a CST (four PEs) with constraint on time (8-bit data width) .....	45
Table 5.4: Results for three switches in a CST (four PEs) with constraint on area (8-bit data width) .....	45
Table 5.5: Results for seven switches in a CST (eight PEs) with constraint on time (8-bit data width) .....	46
Table 5.6: Results for seven switches in a CST (eight PEs) with constraint on area (8-bit data width) .....	46
Table 5.7: Results for 15 switches in a CST (16 PEs) with constraint on time (8-bit data	

width) .....	47
Table 5.8: Results for 15 switches in a CST (16 PEs) with constraint on area (8-bit data width) .....	47
Table 5.9: Results for 15 switches in a CST (16 PEs) with constraint on time (2-bit data width) .....	48
Table 5.10: Results for 15 switches in a CST (16 PEs) with constraint on area (2-bit data width) .....	48
Table 5.11: Results for 15 switches in a CST (16 PEs) with constraint on time (32-bit data width) .....	49
Table 5.12: Results for 15 switches in a CST (16 PEs) with constraint on area (32-bit data width) .....	49

## List of Figures

Figure 2.1: Self-reconfigurable gate array architecture.....	6
Figure 2.2: A CST with eight PEs.....	7
Figure 2.3: Multicast communication set.....	8
Figure 2.4: Non-oriented, well-nested communication set.....	9
Figure 2.5: Right-oriented, well-nested, width-2, communication set.....	10
Figure 2.6: Well-nested communication set.....	11
Figure 2.7: Monotonic communication set.....	12
Figure 3.1: A CST with 8 PEs connected through seven switches.....	14
Figure 3.2: Internal structure of a switch.....	15
Figure 3.3: Internal structure of a switch for a width- $w$ communication set.....	21
Figure 4.1: Internal structure of data unit in a switch. ....	27
Figure 4.2: Block diagram of a switch for well-nested, right-oriented, width-1 communication sets.....	30
Figure 4.3: Block diagram of a switch for multicast, right-oriented, width-1 communication sets .....	34
Figure 4.4: Block diagram of the switch for well-nested, right-oriented, width- $w$ communication sets.....	35
Figure 4.5 Multi-pattern framework for a switch accommodating four communication patterns. ....	39
Figure 4.6: Block diagram of the a CST with four PEs. ....	41
Figure 5.9: Graph comparing the area of each design.....	50
Figure 5.10: Graph comparing the frequency of each design.....	51
Figure 5.11: Graph comparing the power dissipation of each design.....	53

## Abstract

Reconfigurable computing provides a fast and flexible solution for intensive computing processes. Thus, it acts as a bridge between software controlled and hardware based processors. The self-reconfigurable gate array (SRGA) is a reconfigurable architecture that allows fast switching between operations on a reconfigurable device. It consists of a 2-dimensional array of processing elements (PEs) connected using a binary tree structure, called a circuit-switched tree (CST). A CST is a balanced binary tree in which leaves represent processing elements (PE) and internal nodes represent switches. The PEs in the CST communicates with each other by configuring the appropriate switches in the communication path for different types of communication patterns. In this thesis, we have designed and implemented digital blocks for the routing algorithms provided by Roy *et al.* [RTV04] for right-oriented communication patterns for width-1 and width- $w$  well-nested sets and width-1 multicast sets. We have extended the work and implemented the algorithm for point-to-point, right-oriented, width- $w$  communication sets. Finally, we have introduced a multi-pattern framework, which accommodates different communication patterns. All the designs are synthesized for 0.25-micrometer technology and area, frequency, and power analyses are performed. The results show the behavior of the designs with four, eight, and 16 PEs. The results prove that the proposed framework occupies less area as compared to the sum of the areas occupied by other communication patterns discussed.

## 1. Motivation and Introduction

In this fast world, performance expectations and the flexibility requirements of computing resources are increasing drastically. Hardware technology, namely ASIC devices, offers fast and efficient performance for specific computations, but lacks re-programmability needed for flexibility. On the other end, software-controlled processors overcome the lack of re-programmability of ASIC devices, but suffer comparatively slow computation performance. Thus, a greater demand exists for devices that are both fast and flexible enough to change structure to perform various operations. Thus the invention of reconfigurable devices addressed the problem. They offer flexibility along the lines of programmed processors and speed along the lines of ASIC devices [JK02, P02].

Different architectures exist for reconfigurable devices [C0H02, BoP02]. The self-reconfigurable gate array [SWMP00] is one such architecture, allowing fast switching between operations on a reconfigurable device. It consists of a 2-dimensional array of processing elements (PEs) connected using a binary tree structure, called a circuit-switched tree (CST). A CST is a balanced binary tree in which leaves represent PEs and internal nodes represent switches [El03]. (Chapter 2 describes the SRGA and CST in detail.) Different types of communication patterns are possible on the CST. In the CST, PEs communicate with each other by configuring the appropriate switches in the communication path. Roy *et al.* [RTV04, RVT05] have given configuration algorithms for different communication patterns.

In this thesis, we have designed and implemented switches for different configuration algorithms presented in Chapter 3. We have also extended the pattern-specific implementations to a more general case and also introduced a multi-pattern framework design that accommodates four communication patterns discussed in this thesis. The proposed framework allows a user to use single chip to route the data for different communication patterns. We have implemented the

CST with four, eight, and 16 PEs for all the communication patterns. We have studied the behavior of the area, propagation delay, and the maximum frequency for the designs.

Chapter 2 provides background information about reconfigurable computing and its concepts, a detailed description of the SRGA and CST, and different classes of communication possible on CSTs. In Chapter 3, we discuss the configuration algorithms for different communication patterns provided by Roy *et al.* [RTV04, RVT05].

Chapter 4 provides our design and implementation of switches for the communication patterns. We introduce a multi-pattern framework, which accommodates different communication patterns. In Chapter 5, we present the synthesis results for the designs. We provide detailed discussion on the area occupied, propagation delay, and the power dissipation of the design for different data width. Chapter 6 provides conclusions and possible future work directions.

## **2. Background**

### **2.1 Introduction**

This chapter briefly introduces reconfigurable computing and its basic concepts. It describes a special architecture developed for reconfigurable devices, known as the self-reconfigurable gate array (SRGA). We have also provided the basic concepts of the circuit-switched tree (CST), different types of communication patterns, and configurations available on the CST.

### **2.2 Reconfigurable Computing**

In a conventional method, designers implement an algorithm in either customized hardware or existing microprocessors. Hardware methods use either hardwired technology (ASIC – Application Specific Integrated Circuits) or circuit board-level design. ASIC devices perform specific tasks. Thus, they are very fast and efficient in performing specific computations. After the process of fabrication, the design can neither alter to perform different operations nor include new features. The process of redesign to enhance an ASIC is difficult and expensive. The same problem applies to board-level design. The second method uses software-controlled processors to implement algorithms. Conventional processors switch among programs to perform different tasks without the need of changing the hardware. The performance (speed), however, is much less compared to ASIC implementation [CoH02, VT04].

Reconfigurable computing represents a new idea in computer philosophy, where the hardware component of a design configures to support multiple tasks. It provides the flexibility of programmed processors and the speed of hardware. Reconfigurable computing devices have two parts, one part being fixed and the other part reconfigurable. The fixed part (sometimes a

microprocessor) controls the reconfiguration aspect of the reconfigurable part. The reconfigurable part is usually a regular array of configurable logic blocks embedded in a fabric of routing resources that is also configurable. Often, a stream of bits specifies the configuration. In the case of Field Programmable Gate Arrays (FPGA), writing configuration bit streams to programmable logic blocks maps the custom logic to the resources on the device [BoP02].

Gerald Estrin was the first person to propose the concept of reconfigurable computing [EI03]. He built a hybrid configurable computing system with standard general processors and configurable hardware.

Programmable devices may be *static* or *dynamic*. Static reconfiguration configures the programmable device for one application; the next reconfiguration process takes place after the completion of the entire application. Rapid prototyping uses static reconfiguration to test the logic before designing an ASIC. Dynamic reconfiguration (run-time reconfiguration) involves changing the configuration, or parts of the configuration of the system, at run time.

In reconfigurable devices, the total time taken to perform an operation (algorithm) is the sum of reconfiguration time and execution time. Reconfiguration time is the time spent reconfiguring the hardware.

In *partial reconfiguration*, a part of the reconfigurable device undergoes reconfiguration, while the remainder performs its function. Reconfiguration time typically increases linearly with the size of the hardware to be reconfigured. Partial reconfiguration allows overlapping reconfiguration time on one part with computation time on another part. This reduces the overall execution time compared to the process of stopping all computations while reconfiguration occurs [CoH02].

In reconfigurable computing, the ability to perform *context switching* further reduces reconfiguration time. A device with this capability stores a number of configuration layers on a

chip and can switch among them within a single or a few clock cycles. This method initially loads into different layers the configuration bits necessary to perform different operations. If the number of configurations needed for a computation is no more than the number of available layers (contexts), then the time for this initial loading does not occur at run time and so does not factor into reconfiguration time. The time taken to switch to a new configuration at run time is much less than the reconfiguration time in partial reconfiguration by several orders of magnitude [LAPPAJ02]. This feature allows data to be stored on the device while other programs (contexts) may operate on the data [SMP99]. Reconfigurable devices with context-switching capabilities are commercially available. Chameleon Inc. introduced the first commercially available device with context switching capabilities, named as the CS2000 RCP series [B04].

Different architectures exist for reconfigurable devices. The self-reconfigurable gate array is one such architecture that allows single-cycle context switching and single-cycle random access of on-chip configurations. The next section explains in detail about the self-reconfigurable gate array architecture [SP01].

### **2.3 Self-Reconfigurable Gate Array**

Sidhu *et al.* [SWMP00, SP02] proposed the *self-reconfigurable gate array* (SRGA) architecture. In this architecture, the processing elements (PEs) are in the form of a 2-dimensional array. Each PE has direct links to its four surrounding PEs and connects to other PEs in its row and column through a binary tree structure. In the binary tree structure, the PEs are at the leaves and identical switches are at the non-leaf nodes. Section 2.4 describes this circuit-switched tree. Figure 2.1 shows an SRGA with a 4X4 array of processing elements. Each PE contains a logic cell and a memory block. A logic cell consists of a flip-flop and 16-bit look up table (LUT). Memory blocks store configuration contexts and data for the configured logic. In order to connect the PEs to communicate between them, configuration bits configure appropriate

switches (discussed in Chapter 4). Since local information available in each PE configures the switches in the binary tree, the reconfiguration operation is fast. Sidhu *et al.* [SWMP00] claimed that context switch and memory access can each be performed in a 10 ns clock cycle.

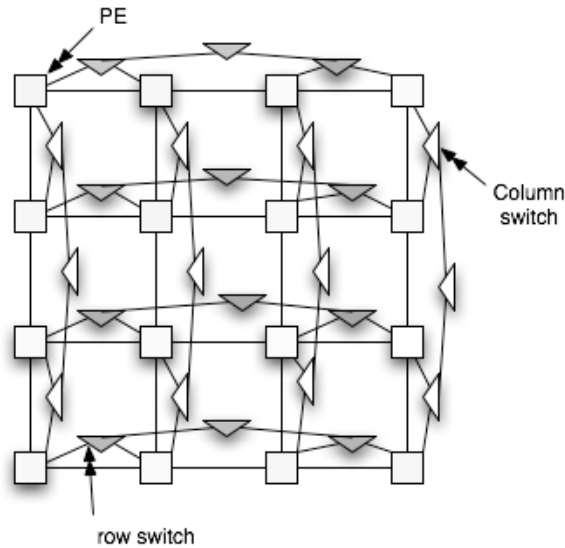


Figure 2.1: Self-reconfigurable gate array architecture.

## 2.4 Circuit-Switched Tree

A *circuit-switched tree* (CST) is a balanced binary tree in which leaves represent PEs and internal nodes represent switches. El-Boghdadi *et al* [EIVTR02, EIVTR03] introduced the concepts in this section. The switches act as gates between processors. In a CST, configuring the appropriate switches in communication path physically connects the source and destination in a communication. The edges in the tree are full duplex (allowing communication to take place in opposite directions simultaneously). Figure 2.2 shows an example of a CST with eight PEs. The seven switches connect the PEs in a row (if a CST has  $n$  number of PEs, then it requires  $(n-1)$  number of switches).

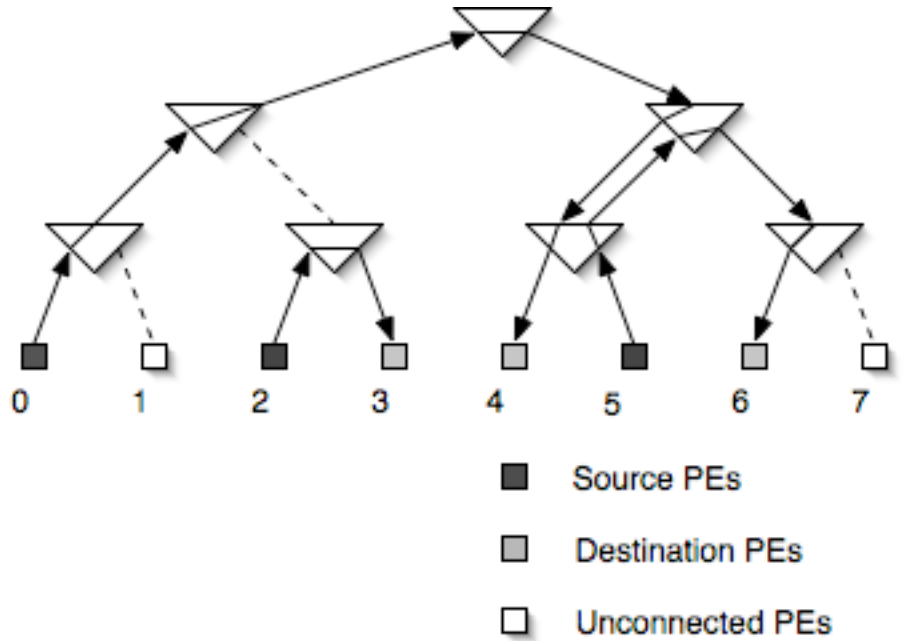


Figure 2.2: A CST with eight PEs.

The PEs in a CST can be source, destination, or neither. The notation  $(s, d)$  represents a point-to-point communication in a CST, where  $s$  is the source PE and  $d$  is the destination PE. In general, there can be more than one destination for a signal source (discussed in detail under Section 2.4.1(2)). In those cases  $(s, D)$  represents a communication, where  $D$  denotes a set of destinations. Because a CST is a tree, each communication has a unique path that connects the source and the destination. In a CST, the path leads upward from the source until it reaches the switch that is the lowest common ancestor of the source and destination, and then the path turns down towards the destination PE. In Figure 2.2, the path for communication  $(0, 4)$  climbs from source 0 up in the CST until it meets the lowest common ancestor with destination PE 4, then heads down towards destination PE 4. Figure 2.2 shows paths using directed arrows.

A collection of individual communications forms a *communication set*. The CST shown in Figure 2.2 has three communications, which form a communication set represented by  $\{(0, 4), (2, 3), (5, 6)\}$ . Some communication sets exhibit patterns that a routing algorithm can exploit to

realize all the paths in a CST. Roy *et al.* [RTV04, RVT05] provide different scheduling and routing algorithms for different communication patterns (discussed in the next chapter). We can divide communication sets based on the characteristics they exhibit, such as (1) number of destinations, (2) orientation, (3) communication width, and (4) incompatibility.

### 2.4.1 Based on Number of Destinations

(1) Point-to-point communication: A *point-to-point communication* has only one source and one destination. One source connects to only one destination. Figure 2.2 gives an example of point-to-point communications. In this communication set  $\{(0, 4), (2, 3), (5, 6)\}$ , each of the three communications has one source (0, 2, and 5) connected to one destination (4, 3, and 6, respectively).

(2) Multicast communication: In a *multicast communication*, a single source has more than one destination. Figure 2.3 gives an example of a multicast communication set. The communication set in the CST shown has two communications. PE 0 is the source of one set, and it has PEs 1, 2, and 4 as destinations, described as  $(0, \{1,2,4\})$ , while  $(5, \{6,7\})$  describes the second communication.

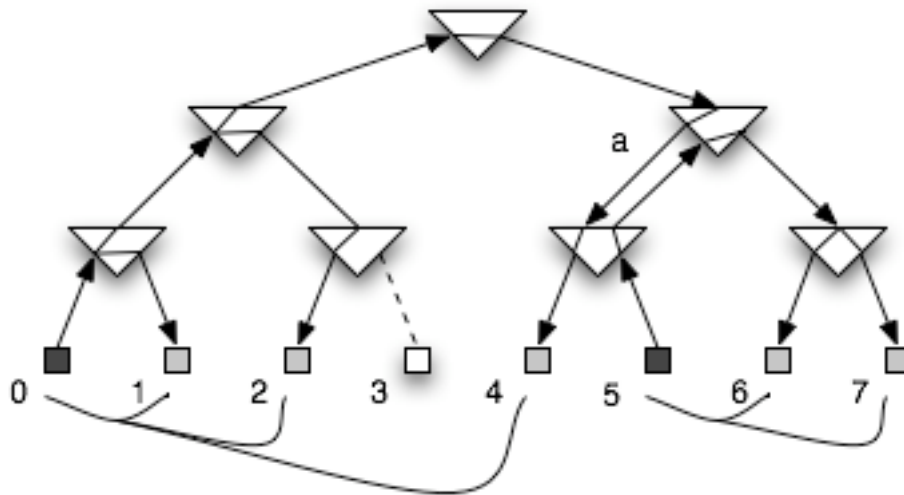


Figure 2.3: Multicast communication set.

### 2.4.2 Based on the Orientation of the Communications

In an *oriented* communication set, all communications have their destinations to the same side of their sources. In a *left-oriented* communication set, the destinations are to the left of the sources. Similarly, in a *right-oriented* communication set, the destinations are to the right of their sources. We can partition any communication set into a left-oriented communication set and a right-oriented communication set. Figures 2.2 and 2.3 show right-oriented communication sets. In the case of a *non-oriented* communication set, some communications are left-oriented and the remainder is right-oriented. Figure 2.4 shows a non-oriented communication set.

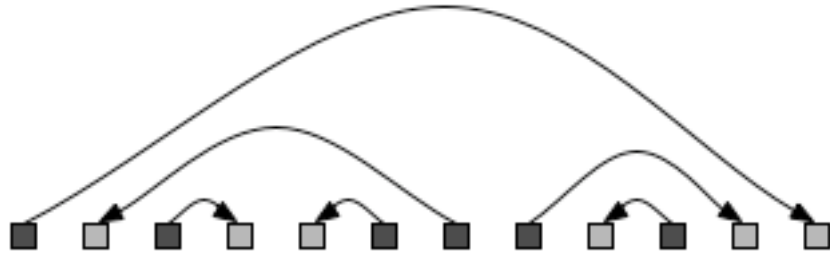


Figure 2.4: Non-oriented, well-nested communication set.

### 2.4.3 Based on the Communication Width

In a CST, the maximum number of communication paths that share an edge in the same direction gives the *width* of the communication set. A CST cannot simultaneously execute two communications that share the same edge in the same direction. Therefore, the width of the communication set has an impact on the number of rounds required to establish the communication paths of all communications in a set. For example, if a communication set has width two, we need two rounds to perform all communications. In general, a communication set with width- $w$  needs a minimum of  $w$  rounds to connect all the communications in the set.

(1) Width-1 communication set: In a width-1 communication set, no two communications share the same edge in the same direction. Thus, routing of the communication

set is possible in one step (that is, one can simultaneously route all the communications in a CST). Figures 2.2 and 2.3 show examples of width-1 communication sets. Two communications share an edge in each figure, but since they are in opposite directions, we can route them simultaneously.

(2) Width- $w$  communication set: In a width- $w$  communication set,  $w$  communications use the same edge in the same direction, requiring a minimum of  $w$  steps to perform all communications on a CST. A width- $w$  communication set is *width-partitionable* if a schedule exists for the CST to perform all communications in  $w$  steps. (Not all communication sets are width-partitionable [EI03, EIVTR02].) Figure 2.5 shows a CST with a width-2 communication set. Both the edges  $a$  and  $b$  have two communications taking place in the same direction.

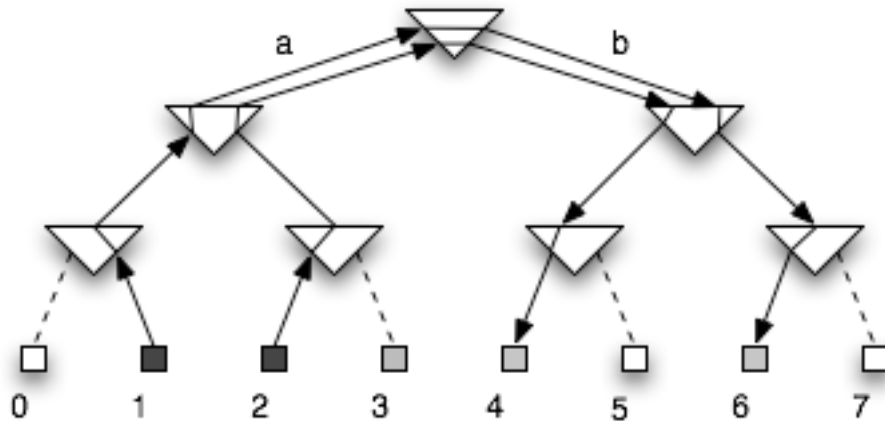


Figure 2.5: Right-oriented, well-nested, width-2, communication set.

#### 2.4.4 Based on the Property of Incompatibility

When two or more communications in a CST use the same edge in the same direction, we say that they are *incompatible*. The CST cannot perform these communications at the same time. Figure 2.5 shows an example communication set, in which (1, 6) and (2, 4) exhibit incompatibility. In this, two communications share edge  $a$  in the same direction. If the two communications go upwards in the CST using the same edge, then it is a *source incompatible*. In

Figure 2.5, (1, 6) and (2, 4) exhibit source incompatibility. Similarly, if two communications come down the CST using a common edge, it is a *destination incompatible*. In Figure 2.5, (1, 6) and (2, 4) also exhibit destination incompatibility because of sharing edge *b*. If no two maximal incompatibles share a common source (or destination), then they are called as *disjoint incompatibles*. If two incompatibles share a common source (or destination), then they are known as *non-disjoint incompatibles*.

## 2.5 Patterns of Communication Sets

Some communication sets exhibit patterns that permit further classification.

(1a) Well-nested communication set: A *well-nested* communication set is a special case of a point-to-point communication set that forms a well-nested parenthesis expression. Figure 2.6 shows a well-nested communication set, with an opening parenthesis symbol “(” representing a source and a closing parenthesis symbol “)” representing a destination. In a well-nested communication set, two possibilities arise. (1) A pair of parentheses (source – destination range) can completely contain another. For example, communication (2, 4) that starts after PE 1 in Figure 2.5 ends before PE 6. (2) The parenthesis pair (source – destination range) can be completely disjoint from another. For example, in Figure 2.6 communications (0, 5) and (6, 12) are completely disjoint. Figure 2.4 also exhibits the property of well-nested communication sets.

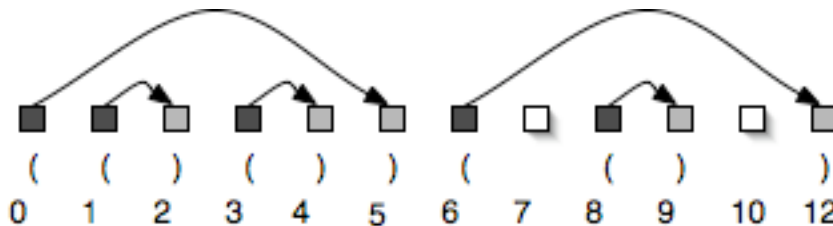


Figure 2.6: Well-nested communication set.

(1b) Monotonic communication set: In this type of communication set, the different communications interleave or concatenate with one other. Figure 2.7 shows an example of a

monotonic set of communications. For a formal definition, refer to El-Boghdadi *et al* [El03, EIVT03].

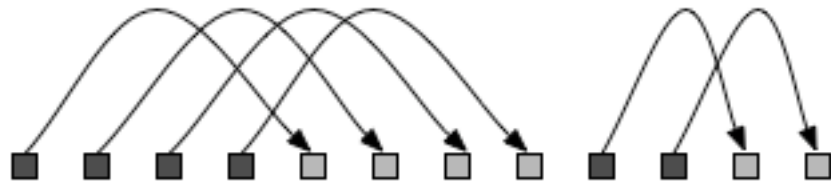


Figure 2.7: Monotonic communication set.

### 3. Configuring Circuit-Switched Trees

#### 3.1 Configuration Basics for the CST

In a CST, the ability to accommodate a communication does not assure a data path connecting the source to the destination PE, unless one configures the CST. Configuring the CST means setting the switches in the CST. Configuring the appropriate switches in the communication path physically connects the source and destination of a communication. Often, the source and destination of a communication will hold a communication ID, enabling the CST to independently construct paths up from each of the source and destination to their lowest ancestor and enabling that switch to recognize that it should connect the paths. Switches in the communication path between the source and destination set their connections based on information received from the parent and child switches. Once the CST has configured all of its switches, it has established data paths for a number of compatible communications. Different communication patterns need different configurations of the CST [RTV04, RVT05].

Each switch (except the root) has connections to a parent, a left child, and a right child. Figure 3.1 shows a CST with eight PEs. The CST has seven identical switches to connect the eight PEs. Switch 4, for example, connects to its parent, switch 6, and to its left and right children, switches 0 and 1, respectively. The root, switch 6, connects to left and right children, but has no parent.

#### 3.2 Structure of a CST Switch

A switch is a collection of units with the function of establishing physical connections among PEs. Figure 3.2 shows the internal structure of a switch. In general, a switch consists of two main units, a *control unit* and a *data unit*. We have designed both the units using combinational blocks. (Section 4.2 will discuss the design and implementation of the switch.)

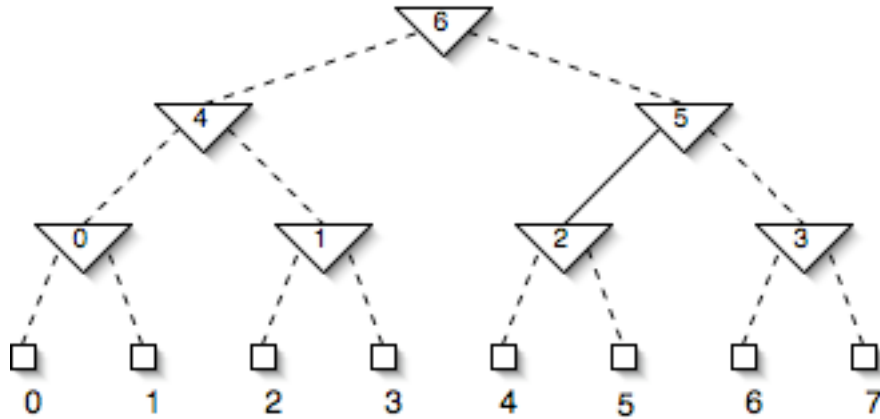


Figure 3.1: A CST with 8 PEs connected through seven switches.

1. The *control unit* accepts two status symbols from its children as inputs. Each status symbol carries information such as status of the PE (whether the PE is a source or destination or neither) and the communication ID. It also receives the status symbol from its parent switch. In general, a control unit has two modes, namely (1) bottom-up mode and (2) bottom-up and top-down mode. It decides on the mode based on the width of the communication set. Generally, a control unit uses Mode 1 operation for a width-1 communication set and Mode 2 for a width- $w$  set. Based on the communication pattern and the status symbol from its children, the control unit performs two functions.

Mode 1: (a) It generates a status symbol and sends it to its immediate parent switch in the CST. (b) It generates configuration signals and sends these to the data unit to configure and establish a physical connection between the appropriate input and output lines in the data unit.

Mode 2: (a) It generates a status symbol and sends it to its immediate parent switch in the CST. (b) It waits until it gets a control signal from its parent to configure the data unit (for more description, see Section 3.4).

- The *data unit* is a combinational block that establishes physical connections between selected input and output data lines. A data unit has three input and three output lines, connecting it to its two children and its parent. The signals Lin, Rin and Pin in Figure 3.2 show the input data lines to the switch from its left child, right child, and parent switch, respectively. Similarly, Lout, Rout, and Pout are the output data lines from the switch to its left child, right child, and its parent, respectively.

Control unit design differs for different communication patterns and classes, but the data unit remains the same.

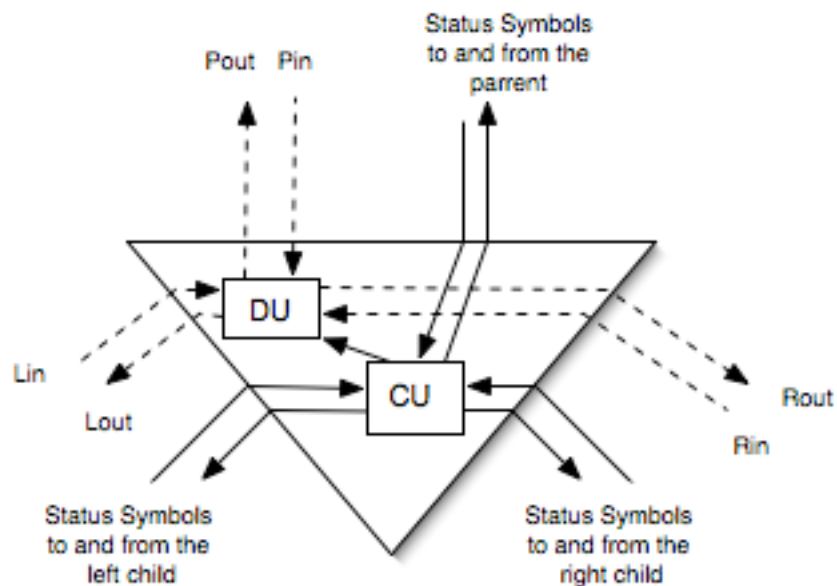


Figure 3.2: Internal structure of a switch.

### 3.3 Configuration of CST for a Width-1 Communication Set

As discussed earlier (Section 2.3.1), in a width-1 communication set, no two communications share a common edge in the same direction. The procedure provided below is a basic framework to establish a data path for a width-1 communication set between source and destination PE pairs, following Roy *et al.* [RTV04]. Sections 3.3.1 and 3.3.2 provide detailed descriptions of the symbols and the configuration for well-nested and multicast communication sets, respectively.

The control unit of for width-1 communication sets operates in Mode 1, bottom-up, so that status symbols flow upwards only.

1. Each PE generates a status symbol announcing its status in the CST. The symbol can be source ( $s$ ), destination ( $d$ ), or either ( $n$ ).
2. These symbols propagate up towards the root and simultaneously configure the switches in the path of communication. The control unit generates and sends a status symbol to its parent using the status symbol function ( $f_s$ ) and a configuration signal to its data unit using the configuration function ( $f_c$ ), based on the input symbols from its left and right children. (Tables 3.1-3.4 show  $f_s$  and  $f_c$  for well-nested and multicast communication sets.) The switch configures itself based on the configuration signal generated by its control unit.
3. The process completes when the root configures itself. The process ignores the status symbol generated by the root switch.

### 3.3.1 Well-Nested Communication Set

This section discusses the algorithm for configuring a CST for an oriented, well-nested communication set given by Roy *et al.* [RTV04]. Tables 3.1 and 3.2 show the status symbol function  $f_s$  and configuration function  $f_c$ , which are to be substituted in the above framework to establish the data path. Oriented, well-nested communication sets on the CST have the property that, when a source symbol and a destination symbol meet at a switch, they belong to the same communication and the switch can connect the data paths. Therefore, such a set does not require communication IDs

Table 3.1: Status symbol function  $f_s$  for a right-oriented, well-nested, width-1 communication set.

$f_s$	<b>s</b>	<b>d</b>	<b>n</b>	<b>b</b>
<b>s</b>		n	s	s
<b>d</b>	b		d	
<b>n</b>	s	d	n	b
<b>b</b>		d	b	b

The control unit in a switch generates the symbol function  $f_s$ . Table 3.1 shows the symbol function  $f_s$  for a right-oriented, well-nested, width-1 communication set. Blank spaces in the table indicate impossible combinations for a right-oriented, well-nested, width-1 communication set. The leftmost column and topmost row in Table 3.1 give the possible left and right input status symbols to the switch from its left and right children, respectively. This holds the same for Tables 3.2-3.4 in this chapter. For example, if a CU (control unit) of a switch receives symbols  $s$  and  $b$  from its left and right children, respectively, then the CU will send  $s$  as the status symbol to its parent.

The control unit in a switch uses configuration function  $f_c$  to generate the configuration signal to configure the data unit. Table 3.2 shows the configuration function for different input combinations of status symbols from the left and right children to the control unit. The data lines flow through the switches as shown in Table 3.2. For example, when a control unit of the switch receives  $b$  signals from its left and right child switches, the CU generates a configuration signal to DU to connect its Pin data line to Lout, Lin to Rout, and Rin to Lout, respectively. The control unit connects the appropriate data lines based on the configuration signal.

Table 3.2: Configuration function  $f_c$  for a right-oriented, well-nested, width-1 communication set.

$f_c$	s	d	n	b
s				
d				
n				
b				

### 3.3.2 Multicast Communication Set

As discussed earlier (Section 2.3.1(2)), a multicast communication has more than one destination for a single source. Thus,  $(s, D)$  represents a multicast communication. A multicast communication set can contain multicast communications. The procedure to establish a data path connection between the PEs in a multicast communication set is mostly similar to that for a well-nested communication set. In multicast communication, however, since there is more than one destination for a single source, it demands additional information. For example, the configuration process does not end when a source encounters a destination as in the case of a well-nested communication set discussed in Section 3.3.1, since a multicast communication has more than one destination. So each destination PE should send a symbol indicating its status as final

destination or not. In Roy *et al.* [RTV04],  $r$  denotes the final (rightmost) destination, and we will also use the same in this thesis. Along with the indication of final destination in a communication, a unique communication ID is necessary to match appropriate sources and destinations [RTV04].

Table 3.3 [RTV04] shows the status symbol function  $f_s$  for width-1 multicast communication sets.

*Notations used:*

$s$  - source,

$d$  - destination,

$m_{\text{communication ID no}}$  - communication ID (where  $a, b, c,$  and  $d$  used in the table refer to communication IDs).

A multicast status symbol is a four-tuple. The first two elements have form  $(s, m_a)$  or  $(-, -)$ . Elements  $(s, m_a)$  imply that the source with ID  $m_a$  of the switch receiving the symbol in the subtree has not matched its rightmost destination, and  $(-, -)$  imply that all sources in the subtree have matched their rightmost destinations. The last two elements have form  $(d, m_b), (r, m_b),$  or  $(-, -)$ . Elements  $(d, m_b)$  imply that non-rightmost destination or destinations with ID  $m_b$  in the subtree have not matched their source,  $(r, m_b)$  imply that the rightmost destination with ID  $m_b$  (and perhaps same non-rightmost destinations with the ID  $m_b$ ) has not matched its source; and  $(-, -)$  imply that all the destinations in the subtree are matched.

Table 3.3: Status symbol function  $f_s$  for the right oriented, multicast, width-1 communication set.

$f_s$	$s, m_c, -, -$	$-, -, d, m_d$	$-, -, -, -$	$-, -, r, m_d$	$s, m_c, d, m_d$	$s, m_c, r, m_d$
$s, m_a, -, -$		If $m_a=m_d$ $(s, m_a, d, m_d)$ else $(s, m_a, d, m_d)$	$s, m_a, -, -$	If $m_a=m_d$ $(-, -, -, -)$ else $(s, m_a, d, m_d)$		$s, m_c, -, -$
$-, -, d, m_b$	$s, m_c, d, m_b$	$-, -, d, m_d$	$-, -, d, m_b$	$-, -, r, m_d$	$s, m_c, d, m_d$	$s, m_c, r, m_d$
$-, -, -, -$	$s, m_c, -, -$	$-, -, d, m_d$	$-, -, -, -$	$-, -, r, m_d$	$s, m_c, d, m_d$	$s, m_c, r, m_d$

(Table Contd..)

$-, -, r, m_b$	$s, m_c, r, m_b$		$-, -, r, m_b$			
$s, m_a, d, m_b$		If $m_a=m_d$ ( $s, m_a, d, m_b$ ) else ( $s, m_a, r, m_d$ )	$s, m_a, d,$ $m_b$	If $m_a=m_d$ ( $-, -, d, m_b$ ) else ( $s, m_a, r, m_d$ )		$s, m_c, d, m_b$
$s, m_a, r, m_b$		$s, m_a, r, m_b$	$s, m_a, r, m_b$	$-, -, r, m_b$		$s, m_c, r, m_b$

For example, consider the control unit of a switch receiving  $(s, m_a, d, m_b)$  and  $(-, -, r, m_d)$  as input from its left and right children, respectively. Two cases are possible for this input combination. (1)  $m_a = m_d$ : In this case the source from the left child matches the rightmost destination from the right child, thus the switch will connect them. (Table 3.4 shows the configuration of the switch for this example.) Thus, the control unit sends the remaining destination symbol from the left child  $(-, -, d, m_b)$  to its parent switch. (2)  $m_a \neq m_d$ : Since the source ID from the left child and the destination ID from the right children do not match, the control unit sends the status symbol  $(s, m_a, r, m_d)$  to its parent switch. In this case the communication ID  $m_b$  has to be equal to  $m_d$  since we deal with width-1 communication sets. It forward symbols  $r$  to the parent instead of  $d$ , to inform it that the rightmost destination for the communication ID  $m_d$  is present in this sub-tree.

Table 3.4 shows the configuration function  $f_c$  for different possible input combinations from left and right children. The control unit in the switch evaluates  $f_c$  and configures the data unit. For example, from Table 3.4, consider the left and right child inputs as  $(s, m_a, d, m_b)$  and  $(-, -, r, m_d)$ , respectively. As discussed earlier in this section, two cases exist. (1)  $m_a = m_d$ : The left input data line connects to the right output data line, since the source matches the rightmost destination. The parent input connects to the left output to establish the destination path from its parent to its child switch. (2)  $m_a \neq m_d$ : The switch establishes three connections. It connects the parent input data lines to the outputs to the children to establish destination paths between the parent and both

children switches. It also connects the left input line to the parent output line to send the unmatched source symbol upward.

Table 3.4: Configuration function  $f_c$  for a right oriented, multicast, width-1 communication set.

$f_c$	$s, m_c, r, r$	$r, r, d, m_d$	$r, r, r, r$	$r, r, r, m_d$	$s, m_c, d, m_d$	$s, m_c, r, m_d$
$s, m_a, r, r$						
$r, r, d, m_d$						
$r, r, r, r$						
$r, r, r, m_d$						
$s, m_a, d, m_d$						
$s, m_a, r, m_d$						

Blank spaces in Table 3.4 indicate impossible combinations for a right-oriented, multicast communication set.

### 3.4 Configuration of CST for Width- $w$ Communication Sets

In a width- $w$  communication set,  $w$  communications share a common edge in the same direction. Sections 2.3.4 and 2.4.3(2) offer a detailed description of width- $w$  communication sets.

In a width- $w$  communication set, to establish data paths between the PEs for all communications,



$a_L, b_L$  and  $a_R, b_R$  – The set of IDs sent to left and right children, during the top-down phase.

### 3.4.2 Procedure to Schedule and Construct Paths in Width- $w$ Communication Sets

We provide a framework below to connect a set of non-conflicting data paths between communicating PEs in a width- $w$  communication set following Roy *et al.* [RVT05]. The signals flow level-by-level from PEs to root, and then from root to PEs in a CST. The CST must execute at least  $w$  rounds of the steps below for a width- $w$  communication set. We specialize the framework for the special case of a well-nested communication set in Section 3.4.3. We implement the switch for a general, width- $w$ , point-to-point communication set in Section 4.6.

A general framework for a right-oriented, width- $w$ , communication set switch:

Bottom-up phase:

1. Each communication has a pre-assigned, unique ID in the communication set to differentiate among communications.
2. The PEs sends status information (which includes communication ID) to their parents. Each symbol flows up towards the root until it meets, a match. As shown in Figure 3.3, the control unit of each switch receives a set of source IDs,  $S_L$  and  $S_R$ , and destination IDs,  $D_L$  and  $D_R$ , from its left and right children, respectively. The control unit performs the computation process and sends a set of source and destination IDs,  $S$  and  $D$ , to its parent, where  $S = (S_L - D_R) \cup S_R$  and  $D = D_L \cup (D_R - S_L)$ . The expression  $(S_L - D_R) \cup S_R$  has two parts. Set difference  $(S_L - D_R)$  gives the IDs in set  $S_L$  that are not present in set  $D_R$ . This finds the unmatched source IDs. It drops all the matched IDs and creates a set of unmatched IDs. Then it takes set union between unmatched sources from the left and set  $S_R$ . Because the communications are right-oriented, no source in  $S_R$  finds its match at this switch. Similar ideas hold good for computing destination set  $D$ .

Top-down phase:

3. After the root switch executes Step 2, it initiates the top-down phase. In a non-root switch, when it receives status symbols from its parents, it executes its top-down phase

**Pseudocode for top-down phase:**

initialize  $a_L, b_L, a_R, b_R$  to null

disconnect all the ports in the switch

if  $a = b = \text{null}$  {

    If  $S_L \cap D_R \neq \emptyset$  {

$e \leftarrow$  any one element of  $S_L \cap D_R$

        Connect Lin to Rout

$a_L \leftarrow e$

$b_R \leftarrow e$

    }

} else

    if  $a \in S_R$  {

        connect Rin to Pout

$a_R \leftarrow a$

    }

    if  $b \in D_L$  {

        connect Pin to Lout

$b_L \leftarrow b$

    }

    if  $a \in (S_L - D_R)$ {

        connect Lin to Pout

$a_L \leftarrow a$

    }

    if  $b \in (D_R - S_L)$ {

        connect Pin to Rout

$b_R \leftarrow b$

    }

    if  $a_L = b_R = \text{null}$  and  $S_L \cap D_R \neq \emptyset$  {

$e \leftarrow$  any one element of  $S_L \cap D_R$

        Connect Lin to Rout

$a_L \leftarrow e$

$b_R \leftarrow e$

    }

}

step. Based on the IDs from the parent and the status symbols ( $S_L$ ,  $S_R$ ,  $D_L$ , and  $D_R$ ) received in the previous phase from the children, the control unit instructs the data unit to configure the switch and sends a pair of IDs for its left ( $a_L$  and  $b_L$ ) and right ( $a_R$  and  $b_R$ ) children. Based on  $a$  and  $b$ , the switch can have more than one configuration as shown in the pseudo-code. Below we have given the pseudocode for a non-root switch in this step as explained in Roy *at el.* [RVT05].

### 3.4.3 Well-Nested, Width- $w$ Communication Sets

A well-nested, width- $w$  communication set is a special case of point-to-point communication sets that does not need a unique communication ID. The other property of a well-nested, width- $w$  communication set is that it is width-partionable [RVT05]. The step-by-step procedure to establish data paths between the PEs in a communication set is similar to the general case discussed in Section 3.4.2. The configuration and control symbol generation by the control unit follows the pseudocode shown above. In a well-nested communication set, the value of nesting depth of the communication is assigned as communication ID for that communication [RVT05, DV04].

## 4. Implementation

### 4.1 Introduction

In this chapter, we discuss the design and implementation of a switch (discussed in Chapter 3) in a CST. This chapter provides the design of a switch for four different patterns of right-oriented communication set: (1) well-nested, width-1, (2) multicast, width-1, (3) well-nested, width- $w$ , and (4) general point-to-point, width- $w$ . Finally, we introduce a multi-pattern framework design that accommodates multiple communication patterns. This framework performs routes for different communication patterns by switching among different control units. This framework reduces the area occupied, propagation delay, and the power dissipated in a chip (see Chapter 5 for the results).

Recall that the switch basically consists of two blocks, namely, control unit (CU) and data unit (DU). The function of the control unit differs for different communication patterns. We design the control unit separately for each communication pattern. The design of the data unit remains the same, however, for all communication patterns.

### 4.2 Design of Data Unit in a Switch

The data unit (for details, see Section 3.3) in a switch physically connects the data lines, thus establishing communication paths between source and destination PEs. We have designed the data unit using combinational logic. We obtained the basic idea of the design from Sidhu *et al.* [SWMP00]. Figure 4.1 shows the design of the data unit for the switch. The design contains three 2 X 1 multiplexers. The *enable* (negative logic) and *select* signals control the operation of the multiplexers.  $Lin$ ,  $Rin$ , and  $Pin$  are the input data lines, and  $Lout$ ,  $Rout$ , and  $Pout$  are the output data lines of the data unit. When enabled, the multiplexer connects the selected inputs to its output. For example, consider the second mux in Figure 4.1. Three cases exist. Case 1: The

mux does not perform any operation when enable = 1. Case 2: Connect data line Lin to output Rout when enable = 0 and select = 0. Case 3: Connect Pin to Rout when enable = 0 and select = 1.

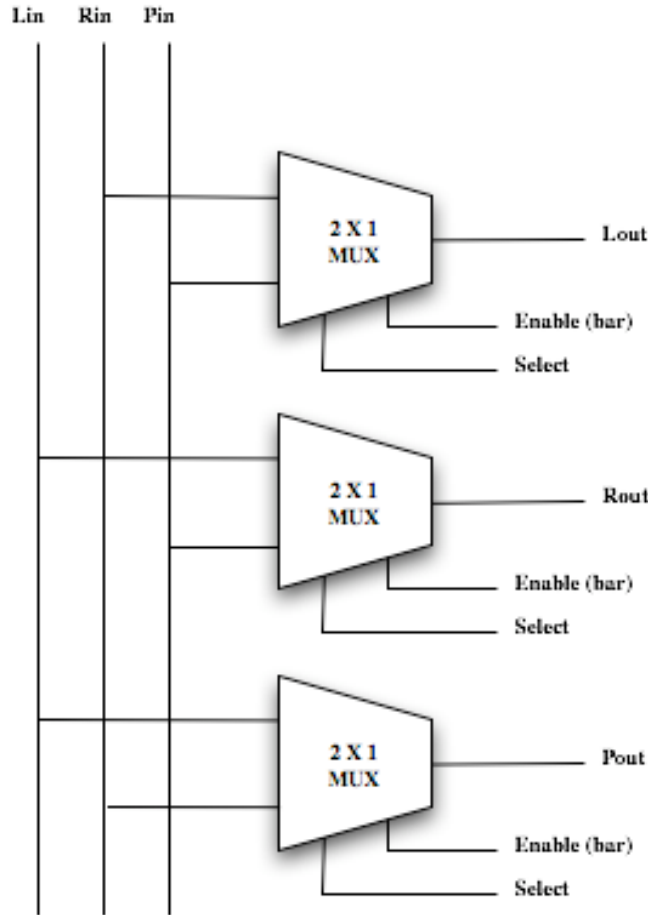


Figure 4.1: Internal structure of data unit in a switch.

The design of the data unit in a switch remains the same for all the communication patterns discussed in the next section.

#### 4.3 Design of a Switch for Well-Nested, Right-Oriented, Width-1 Communication Sets

The control unit for a well-nested, width-1 communication set operates in Mode 1 (defined in Section 3.2). The control unit uses a *bottom-up* approach, where the control unit in the switch generates and sends a status symbol (based on status symbol function  $f_s$  specified in Table 3.1) to the parent switch and sends a configuration signal (based on the configuration

function  $f_c$  specified in Table 3.2) to the data unit simultaneously. Section 3.3.1 provides a detailed description of the configuration algorithm for well-nested, right oriented, width-1 communication sets [RTV04].

For implementation purposes, we have assigned binary values to represent the status (source ( $s$ ), destination ( $d$ ), both source and destination ( $b$ ), or either ( $n$ )) of each PE in the CST as shown in Table 4.1. We have implemented the status symbol function  $f_s$  (Table 3.1) and configuration function  $f_c$  (Table 3.2) in the form of look-up tables (LUT). The advantage of using LUTs in a design is a lack of complexity and thus less propagation delay. Table 4.2 shows  $f_s$  for well-nested, right-oriented communication sets in the form of LUT contents. It shows the possible input combinations from the left and right children to the control unit and provides the output. Table 4.2 also shows the output in two different forms (binary and status symbol). The symbol  $x$  denotes a don't care condition. Similarly, Table 4.3 shows  $f_c$ . The output column shows the connections among the input and output data lines.

Table 4.1: Notation for PE status.

Status Symbols	Binary value assigned
n	00
s	01
d	10
b	11

Table 4.2: LUT contents for status symbol function  $f_s$  for well-nested, right-oriented, width-1 communication set (LUT1).

Input signals				Output		
<i>Status symbol from left child</i>		<i>Status symbol from right child</i>		<i>Binary representation</i>		<i>Symbol representation</i>
0	0	0	0	0	0	$n$
0	0	0	1	0	1	$s$

(Table Contd..)

0	0	1	0	1	0	$d$
0	0	1	1	1	1	$b$
0	1	0	0	0	1	$s$
0	1	0	1	x	x	$x$
0	1	1	0	0	0	$n$
0	1	1	1	0	1	$s$
1	0	0	0	1	0	$d$
1	0	0	1	1	1	$b$
1	0	1	0	x	x	$x$
1	0	1	1	x	x	$x$
1	1	0	0	1	1	$b$
1	1	0	1	x	x	$x$
1	1	1	1	1	1	$b$

Table 4.3: Connections determined by  $f_c$  for well-nested, right-oriented,width-1 communication set (LUT2).

Input Signal			Switch connections
<i>Status symbol from left child</i>	<i>Status symbol from right child</i>	<i>Symbol representation of Input</i>	
00	00	nn	No Connection
00	01	ns	$(R_{in} \rightarrow P_{out})$
00	10	nd	$(P_{in} \rightarrow R_{out})$
00	11	nb	$(P_{in} \rightarrow R_{out})$ $(R_{in} \rightarrow P_{out})$
01	00	sn	$(L_{in} \rightarrow P_{out})$
01	01	ss	No Connection
01	10	sd	$(L_{in} \rightarrow R_{out})$
01	11	sb	$(L_{in} \rightarrow R_{out})$ $(R_{in} \rightarrow P_{out})$
10	00	dn	$(P_{in} \rightarrow L_{out})$
10	01	ds	$(P_{in} \rightarrow L_{out})$ $(R_{in} \rightarrow P_{out})$
10	10	dd	No Connection
10	11	db	No Connection
11	00	bn	$(P_{in} \rightarrow L_{out})$ $(L_{in} \rightarrow P_{out})$
11	01	bs	No Connection
11	10	bd	$(P_{in} \rightarrow L_{out})$ $(L_{in} \rightarrow R_{out})$
11	11	bb	$(P_{in} \rightarrow L_{out})$ $(L_{in} \rightarrow R_{out})$ $(R_{in} \rightarrow P_{out})$

Figure 4.2 shows the block diagram of a switch for well-nested, right-oriented, width-1 communication sets. The block accepts status symbols from its left and right children and (1) generates a status symbol for its parent and (2) configures the data unit. LUT1 implements  $f_s$  using Table 4.2, and LUT2 implements  $f_c$  using Table 4.3, generating three-bit enable and three-bit select signals for the data unit (one bit for each mux in the data unit.)

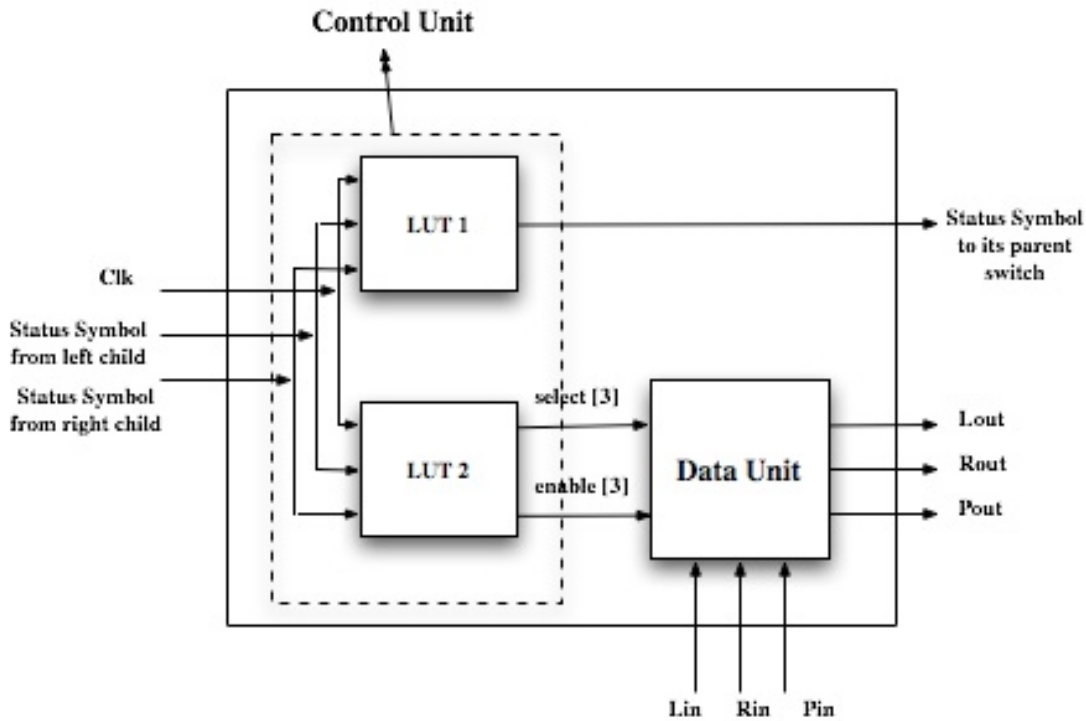


Figure 4.2: Block diagram of a switch for well-nested, right-oriented, width-1 communication sets.

#### 4.4 Design of a Switch for Multicast, Right-Oriented, Width-1 Communication Sets

For multicast, as in the above section, we have implemented  $f_s$  (Table 3.3) as module- $f_s$  and  $f_c$  (Table 3.4) as LUT3. Section 3.3.2 described each multicast status symbol as a four-tuple, but we use a three-tuple, coding source and destination information together. During implementation of module- $f_s$  and LUT3, we have used 11 bits binary to represent status symbols. The first 3 bits [10:8] in the status symbol (moving from MSB) denote the status of the PEs, among which the 1<sup>st</sup>

bit [10] indicates whether it is a final destination (the bit is 1 if final (rightmost) destination, 0 otherwise). The remaining two bits [9:8] denote PE status as shown in Table 4.1. We used four bit to represent each communication ID, so among the remaining eight bits [7:0], four-bits [7:4] holds the source ID and four-bits [3:0] holds the destination ID. Thus, this design applies when the number of distinct communication IDs in a set is below 16.

Tables 4.4 and 4.5 describe module- $f_s$  and LUT3. The first three bit field in each input and output string in Table 4.4 represents the status symbol. The remaining two four bit fields represent the source and destination IDs. For example, the seventh row in Table 4.4 has L [11:0] = 010\_0000\_xxxx and R [11:0] = 111\_xxxx\_0000 as left and right child inputs. The output for this combination from the table is 011\_R [7:4]\_L [3:0] where L [3:0] and R [7:4] specify the bit locations in the left and right child inputs. This means that the destination ID of the left input and the source ID of the right input from the destination ID and the source ID of the output, respectively. Similarly, the Table 4.5 shows the configuration function, in which LUT3 generates a configuration signal based on its two input status symbols L [11:0] and R [11:0] from its left and right children, respectively (for description of the configuration signal see Section 4.2).

Table 4.4: Module- $f_s$  mapping for status symbol function  $f_s$  for multicast, right-oriented communication set.

Input						Output		
Status symbol from left child (L)			Status symbol from right child (R)			Binary format		
<b>001</b>	<b>xxxx</b>	<b>0000</b>	<b>001</b>	<b>xxxx</b>	<b>0000</b>			
<b>001</b>	<b>xxxx</b>	<b>0000</b>	<b>010</b>	<b>0000</b>	<b>xxxx</b>	<b>001</b>	L [7:4]	<b>0000 *</b>
						<b>011</b>	L [7:4]	<b>R [3:0]</b>
<b>001</b>	<b>xxxx</b>	<b>0000</b>	<b>000</b>	<b>0000</b>	<b>0000</b>	<b>001</b>	L [7:4]	<b>0000</b>
<b>001</b>	<b>xxxx</b>	<b>0000</b>	<b>110</b>	<b>0000</b>	<b>xxxx</b>	<b>000</b>	<b>0000</b>	<b>0000*</b>
						<b>111</b>	L [7:4]	<b>R [3:0]</b>
<b>001</b>	<b>xxxx</b>	<b>0000</b>	<b>011</b>	<b>xxxx</b>	<b>xxxx</b>			
<b>001</b>	<b>xxxx</b>	<b>0000</b>	<b>111</b>	<b>xxxx</b>	<b>xxxx</b>	<b>001</b>	L [7:4]	<b>0000</b>
010	0000	xxxx	001	xxxx	0000	011	R [7:4]	L [3:0]
010	0000	xxxx	010	0000	xxxx	010	0000	R [3:0]
010	0000	xxxx	000	0000	0000	010	0000	L [3:0]
010	0000	xxxx	110	0000	xxxx	110	0000	R [3:0]
010	0000	xxxx	011	xxxx	xxxx	011	R [7:4]	R [3:0]
010	0000	xxxx	111	xxxx	xxxx	111	R [7:4]	R [3:0]

(Table Contd..)

000	0000	0000	001	xxxx	0000	001	R [7:4]	0000
000	0000	0000	010	0000	xxxx	010	0000	R [3:0]
000	0000	0000	000	0000	0000	000	0000	0000
000	0000	0000	110	0000	xxxx	110	0000	R [3:0]
000	0000	0000	011	xxxx	xxxx	011	R [7:4]	R [3:0]
000	0000	0000	111	xxxx	xxxx	111	R [7:4]	R [3:0]
110	0000	xxxx	001	xxxx	0000	111	R [7:4]	L [3:0]
110	0000	xxxx	010	0000	xxxx			
110	0000	xxxx	000	0000	0000	110	L [7:4]	0010
110	0000	xxxx	110	0000	xxxx			
110	0000	xxxx	011	xxxx	xxxx			
110	0000	xxxx	111	xxxx	xxxx			
011	xxxx	xxxx	001	xxxx	0000			
011	xxxx	xxxx	010	0000	xxxx	011	L [7:4]	L [3:0]*
						011	L [7:4]	R [3:0]
011	xxxx	xxxx	000	0000	0000	011	L [7:4]	L [3:0]
011	xxxx	xxxx	110	0000	xxxx	010	0000	L [3:0]*
						111	L [7:4]	R [3:0]
011	xxxx	xxxx	011	xxxx	xxxx			
011	xxxx	xxxx	111	xxxx	xxxx	011	R [7:4]	L [3:0]
111	xxxx	xxxx	001	xxxx	0000			
111	xxxx	xxxx	010	0000	xxxx	111	L [7:4]	L [3:0]
111	xxxx	xxxx	000	0000	0000	111	L [7:4]	L [3:0]
111	xxxx	xxxx	110	0000	xxxx	110	0000	L [3:0]
111	xxxx	xxxx	011	xxxx	xxxx			
111	xxxx	xxxx	111	xxxx	xxxx	111	R [7:4]	L [3:0]

Note:

1. If the communication IDs of the left and right children match, then use the starred output in Tables 4.4 and 4.5.
2. The blank spaces in Table 4.4 indicate impossible input combinations.
3. Entry x denotes don't care.

Table 4.5: Connections determined by  $f_c$  for multicast, right-oriented, width-1 communication set (LUT3).

Input Signal (Hexadecimal format)		Switch connections
Status symbol from left child	Status symbol from right child	
1x0	1x0	No Connection
1x0	20x	$P_{out}(L_{in} \rightarrow P_{out}) / R_{out}(L_{in} \rightarrow R_{out})^*$ $P_{out}(L_{in} \rightarrow P_{out}) / R_{out}(P_{in} \rightarrow R_{out})$
1x0	000	$P_{out}(L_{in} \rightarrow P_{out})$
1x0	60x	$R_{out}(L_{in} \rightarrow R_{out})^*$ $P_{out}(L_{in} \rightarrow P_{out}) / R_{out}(P_{in} \rightarrow R_{out})$
1x0	3xx	No Connection
1x0	7xx	$P_{out}(R_{in} \rightarrow P_{out}) / R_{out}(L_{in} \rightarrow R_{out})$
20x	1x0	$P_{out}(R_{in} \rightarrow P_{out}) / L_{out}(P_{in} \rightarrow L_{out})$
20x	20x	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(P_{in} \rightarrow R_{out})$
20x	000	$L_{out}(P_{in} \rightarrow L_{out})$

(Table Contd..)

20x	60x	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(P_{in} \rightarrow R_{out})$
20x	3xx	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(P_{in} \rightarrow R_{out}) / P_{out}(R_{in} \rightarrow P_{out})$
20x	7xx	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(P_{in} \rightarrow R_{out}) / P_{out}(R_{in} \rightarrow P_{out})$
000	1x0	$P_{out}(R_{in} \rightarrow P_{out})$
000	20x	$R_{out}(P_{in} \rightarrow R_{out})$
000	000	No Connection
000	60x	$R_{out}(P_{in} \rightarrow R_{out})$
000	3xx	$R_{out}(P_{in} \rightarrow R_{out}) / P_{out}(R_{in} \rightarrow P_{out})$
000	7xx	$R_{out}(P_{in} \rightarrow R_{out}) / P_{out}(R_{in} \rightarrow P_{out})$
60x	1x0	$P_{out}(R_{in} \rightarrow P_{out}) / L_{out}(P_{in} \rightarrow L_{out})$
60x	20x	No Connection
60x	000	$L_{out}(P_{in} \rightarrow L_{out})$
60x	60x	No Connection
60x	3xx	<b>No Connection</b>
60x	7xx	<b>No Connection</b>
3xx	1x0	No Connection
3xx	20x	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(P_{in} \rightarrow R_{out}) / P_{out}(L_{in} \rightarrow P_{out}) *$ $L_{out}(P_{in} \rightarrow L_{out}) / P_{out}(L_{in} \rightarrow P_{out}) / R_{out}(L_{in} \rightarrow R_{out})$
3xx	000	$L_{out}(P_{in} \rightarrow L_{out}) / P_{out}(L_{in} \rightarrow P_{out})$
3xx	60x	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(L_{in} \rightarrow R_{out}) *$ $L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(P_{in} \rightarrow R_{out}) / P_{out}(L_{in} \rightarrow P_{out})$
3xx	3xx	No Connection
3xx	7xx	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(L_{in} \rightarrow R_{out}) / P_{out}(R_{in} \rightarrow P_{out})$
7xx	1x0	No Connection
7xx	20x	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(L_{in} \rightarrow R_{out}) / P_{out}(L_{in} \rightarrow P_{out})$
7xx	000	$L_{out}(P_{in} \rightarrow L_{out}) / P_{out}(L_{in} \rightarrow P_{out})$
7xx	60x	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(L_{in} \rightarrow R_{out})$
7xx	3xx	No Connection
7xx	7xx	$L_{out}(P_{in} \rightarrow L_{out}) / R_{out}(L_{in} \rightarrow R_{out}) / P_{out}(R_{in} \rightarrow P_{out})$

Figure 4.3 shows the design of a switch for multicast, right-oriented, width-1 communication sets. The module- $f_s$  and LUT3 blocks implement Tables 4.4 and 4.5. The design uses a comparator to compare IDs. The switch operates in Mode 1 (see Section 3.2 for details). The switch is clocked if it is placed in the bottom level of the CST. The switches placed above the bottom level are triggered by the output status symbols from the previous level.

#### 4.5 Design of a Switch for Well-nested, Right-oriented, Width-w Communication Sets

The control unit for well-nested, width-w communication sets works in Mode 2 (see Section 3.2 for more details). Figure 3.3 shows the internal structure of a switch, and Section 3.4 gives a detailed description. In the first, *bottom-up*, phase, the switch receives status symbols from its children. Based on this, the control unit sends a status symbol to its parent.

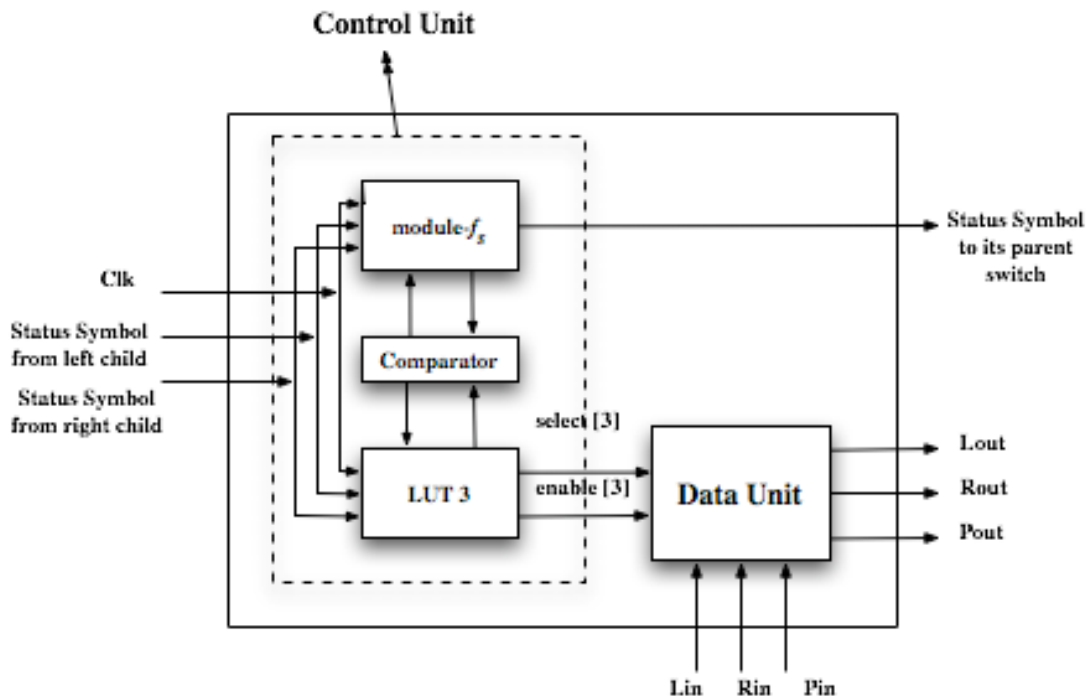


Figure 4.3: Block diagram of a switch for multicast, right-oriented, width-1 communication sets.

Symbols step up the tree level by level until meeting matches. In the *top-down* phase, the switch receives symbols from its parent and then it generates and sends symbols to its children. At the same time, it configures appropriate data lines in the data unit based on the configuration information generated by the control unit.

Figure 4.4 shows the block diagram of the switch for well-nested, right-oriented, width- $w$  communication sets. The design of the control unit consists of different blocks such as status symbol generator, comparator, LUT4, and two blocks for taking set intersection for two different ranges of signal size (signal size here means number of bits in a signal).

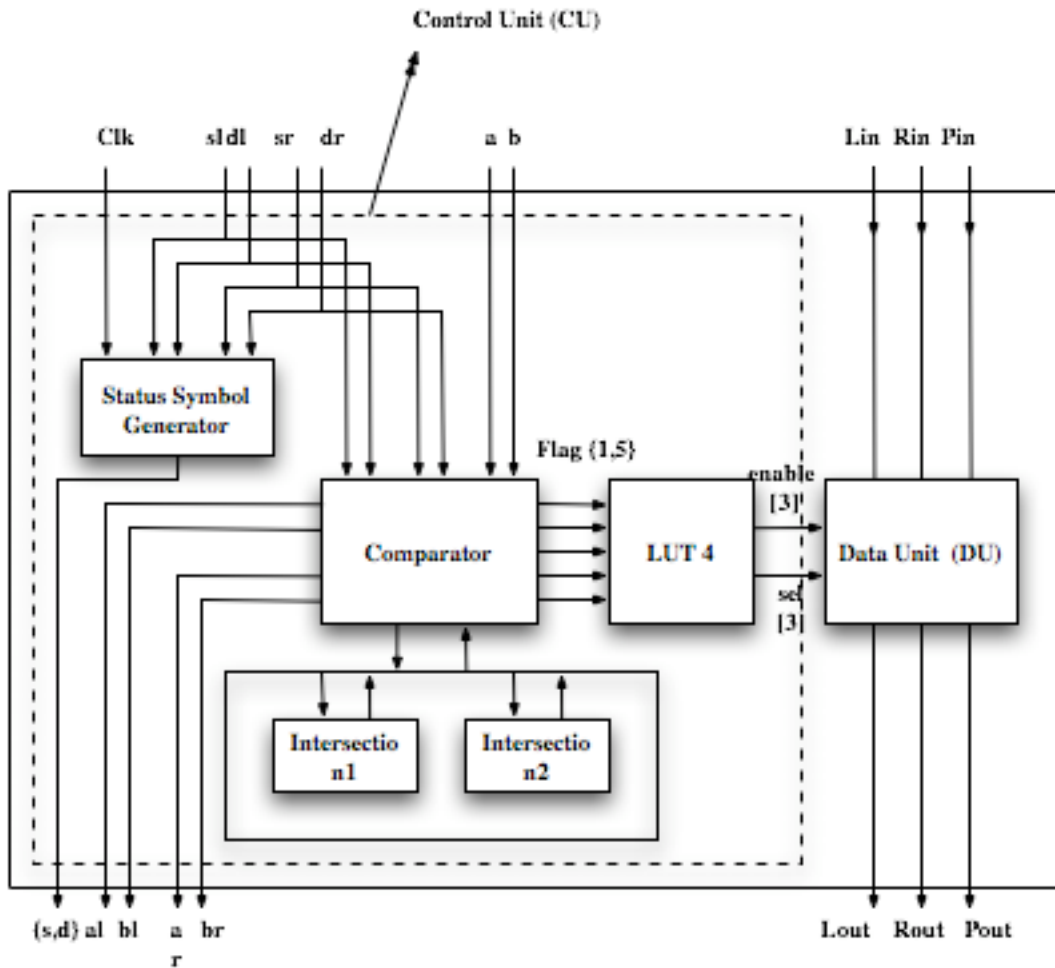


Figure 4.4: Block diagram of the switch for well-nested, right-oriented, width- $w$  communication sets.

The sets of sources and destinations passed from child switch to parent switch exhibit special properties for well-nested, oriented sets. When assigning the nesting depth of a communication as its ID [RTV05, DV04], the set of source (destination) indices forms a contiguous range. When a switch removes matching IDs and combines remaining source (destination) IDs from its children, the set of indices remains a contiguous range. Consequently, we can specify such a set by its lowest and highest IDs. For example, let  $[1, 5]$  denotes the range of IDs with the values  $1, 2, \dots, 5$ .

Bottom-up phase: The *status symbol generator* block is responsible for generating symbols  $S$  and  $D$  to be sent to its parent during the bottom-up phase. This block accepts the sets of source and destination IDs from its left and right children as input. It then generates  $S$  and  $D$  according to  $S = (S_L - D_R) \cup S_R$  and  $D = D_L \cup (D_R - S_L)$  discussed in Section 3.4.2. The set union is a concatenation of ranges, according to the property discussed above.

Let  $S_L = [s_{L1}, s_{L2}]$ ,  $D_R = [d_{R1}, d_{R2}]$ , and  $S_R = [s_{R1}, s_{R2}]$ . From the properties discussed earlier, sets  $S_L$  and  $D_R$  arriving a switch have to be in one of three cases, if both sets are non-empty.

1.  $s_{L1} = d_{R1}$  and  $s_{L2} = d_{R2}$
2.  $s_{L1} < d_{R1}$  and  $s_{L2} = d_{R2}$
3.  $s_{L1} > d_{R1}$  and  $s_{L2} = d_{R2}$

In all the three cases, the upper range of source and destination set has to match, due to the well-nested property. Case 1: Every source ID and destination ID match. Case 2: The range of the source set is bigger than that of the destination set, so some source IDs do not match. Case 3: The range of the source set is smaller than that of the destination set, so some destinations do not match. Thus using the properties and the cases discussed above, we have simplified the pseudocode. This code explains the operation performed to obtain set  $S$  (where  $S = (S_L - D_R) \cup S_R$ ) by the status symbol generator. An analogous procedure applies to  $D$ .

Top-down phase: The *comparator* with the *intersection* blocks implement the pseudocode provided in Section 3.4.2. The *comparator* block accepts status symbols  $a$  and  $b$  from its parent and compares them with the sets of source and destination IDs received from its children during the bottom-up phase. Block *intersection1* finds the intersection between

**Pseudo code**

```

if ( $S_L = \emptyset$ ) {
     $S = S_R$ 
}
else {
    if ( $D_R = \emptyset$ ) {
         $S = [S_{L1}, S_{R2}]$ 
    }
    else {
        if ( $s_{L1} \geq d_{L1}$ ) {
             $S = S_R$ 
        }
        else {
             $S = [S_{L1}, S_{R2}]$ 
        }
    }
}

```

the source set  $S_L$  and destination set  $D_R$ . Block *intersection2* checks whether the symbol  $a$  belongs to  $S_L$  or  $S_R$ . Based on the comparison results the comparator module generates flags for each configuration specified in the pseudocode. Using LUT4 these flags generate enable and select signals.

During implementation we have used nine bits to represent a status symbol. The MSB is 1 if the set is empty, 0 otherwise. The remaining eight bits represent the lower and the upper limits of the range of IDs (four bits for each).

#### **4.6 Design of a Switch for Point-to-Point, Right-Oriented, Width-w Communication Sets**

The design of a switch for point-to-point, right-oriented, width- $w$  communication sets implements the general scheduling and configuring procedure in Section 3.4.2. The point-to-point communication set does not necessarily exhibit the properties discussed in Section 4.5, as a well-nested communication set is a special case of a point-to-point communication set. Thus, we

cannot represent the source and destination sets ( $S_L$ ,  $S_R$ ,  $D_L$ , and  $D_R$ ) as ranges of IDs. These are sets with arbitrary IDs.

Bottom-up phase: In this case the source and the destination sets are arbitrary sets rather than ranges. Thus, the pseudocode in Section 4.5 does not apply. Thus the design of the *status symbol generator* in this section differs from the design in the previous section. In this, the status symbol generator has to check the presence of each and every ID in set  $S_L$  with every ID in the set  $D_R$  to compute  $(S_L - D_R)$ . Then it takes the set union between sets  $S_R$  and  $(S_L - D_R)$ . Here the union of sets is not the concatenation of ranges as in the previous section. An analogous procedure applies to obtain  $D$ . Thus the status symbol generator module becomes more complex and occupies more area.

Top-down phase: The *comparator* with the *intersection* blocks implements the pseudocode provided in Section 3.4.2. The comparator accepts status symbols  $a$  and  $b$  from its parent and compares them with the sets of source and destination IDs. Again in this case to find whether  $a$  belongs to  $S_L$  or  $S_R$  (similarly to find whether  $b$  belongs to  $D_L$  or  $D_R$ ), we have to compare  $a$  with every ID in  $S_L$  and  $S_R$ . Thus, the design becomes complex. The intersection block finds the set intersections described in the pseudocode of Section 3.4.2.

As the number of communications in the CST increases, the number of communication IDs used will increase (each communication has a unique ID). In this case, the number of IDs against which the symbol  $a$  (or  $b$ ) has to be compared also increases. This accounts for the overall increase in the complexity of the design.

#### **4.7 Design of a Multi-pattern Framework**

We propose a multi-pattern framework that facilitates user selection of the switch function for different communication patterns. We have given a prototype for the four types of

communication patterns discussed in Sections 4.3-4.6. Figure 4.5 shows the multi-pattern framework block diagram for the proposed idea.

The framework shown is for a single switch. We have one common data unit for all patterns. The design has different control units for different communication patterns. The design of each control unit for different communication patterns in the framework is identical to the one discussed in Sections 4.3-4.6. The output of the framework changes based on the select line (*Comm\_select*). The framework generates configuration signals (enable and select signals) for all patterns. A 4 X 1 multiplexer selects one configuration signal based on the two-bit *Comm\_select* signal. Thus, a single switch can perform the operation of four different switches. The area occupied by our framework is much less compared to the sum of the areas occupied by the switches of different communication patterns (see Chapter 5).

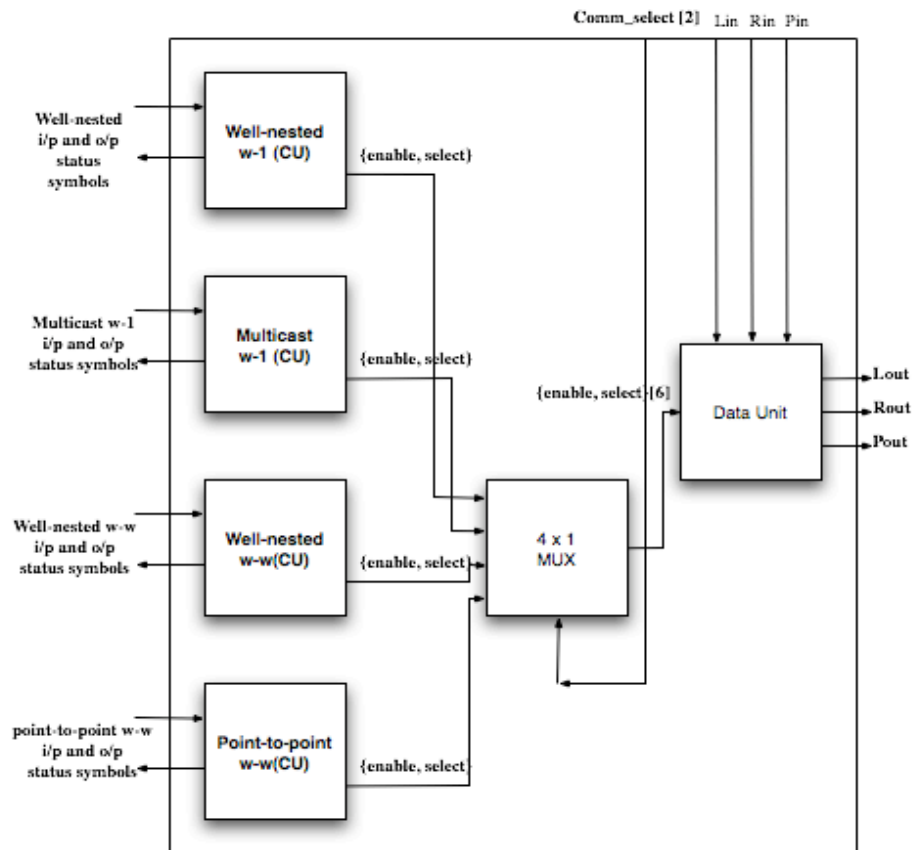


Figure 4.5 Multi-pattern framework for a switch accommodating four communication patterns.

#### 4.8 Implementation of a CST with Four, Eight, and 16 PEs

We have implemented the CST with four, eight, and 16 processors separately for all the four communication patterns discussed in this thesis, namely (1) well-nested, right-oriented, width-1, (2) multicast, right-oriented, width-1, (3) well-nested, right-oriented, width- $w$ , and (4) point-to-point, right-oriented, width- $w$ . We have extended our work to implement four, eight, and 16 PEs for the proposed multi-pattern framework.

Figure 4.6 shows a CST with four PEs. The implementation contains three identical switches. It has two levels, with two switches at the bottom level and the remaining switch (the root switch) at the top level. The switches at the bottom level trigger based on the input clock signal and high-level switch triggers based on the output generated by its previous level switches. Figure 4.6 shows only the data paths of the CST. The parent output of switch 0 connects with the left input of switch 2. Similarly, the parent output of switch 1 connects the right input of switch 2. The left and right output lines of switch 2 connect to the parent input lines of its respective children. The design ignores the parent input and output of the root.

We obtain the implementation of a CST with four PEs for a particular communication pattern by replacing the switches in Figure 4.6 with the switch design of that communication pattern.

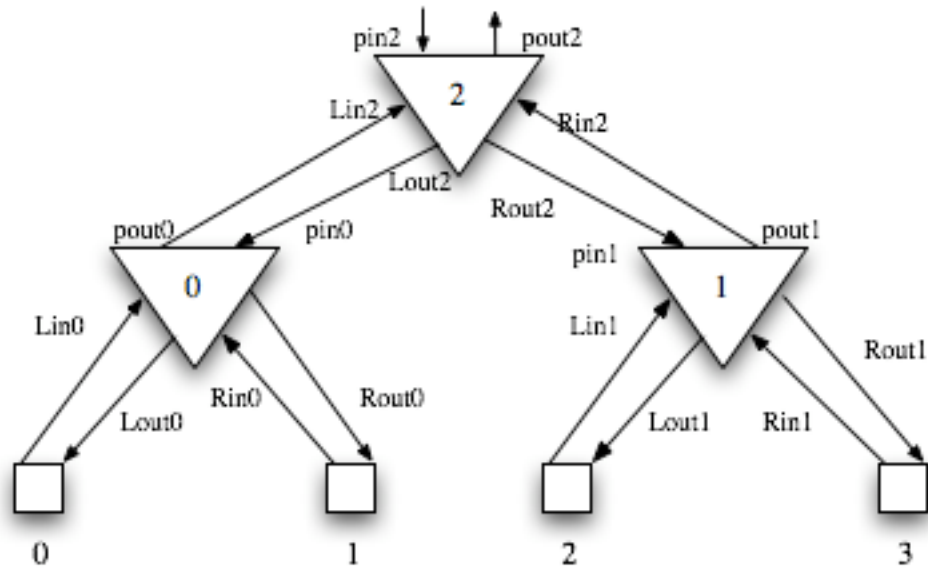


Figure 4.6: Block diagram of the a CST with four PEs.

## 5. Implementation Results

### 5.1 Introduction

In this chapter, we provide the simulation and synthesis results of implementations for a single switch and for CSTs with four, eight, and 16 PEs (that is three, seven, and 15 switches). We have mapped our ASIC design with 0.25 $\mu$ m technology to obtain area, delay, and power measurements. We then discuss the results.

### 5.2 Implementation Environment

We have used *Verilog hardware description language* (Verilog HDL) to describe the design. We have written each block in the design as a separate *module*. The main module instantiates submodules to obtain a complete design. The implementation uses *Cadence* suite EDA (Electronic Design Automation) tools. We have compiled, simulated, and tested the design using Cadence simulator *NC-Verilog*.

The next step is to synthesize the design. Synthesis is the important step in the ASIC design methodology in which the conceptual HDL design definition is converted into physical circuit representation for the specified target technology [Cadence 02]. We have converted our Verilog HDL codes into realizable digital circuits using the Cadence synthesis tool, *Ambit Build Gates* or *PKS (Physically Knowledgeable Synthesis)*. We have mapped our design with the *timing library file* and *layout exchange library file* for 0.25 $\mu$ m technology. We have executed the process of synthesis separately with the time constraint and area constraint on the design [Cadence 03]. The *time*, *area*, and *power* reports for each design are generated using the synthesis tool. We have also written a shell script to automate the process of synthesis.

### 5.3 Implementation Results and Discussion

In this section we present the results obtained after synthesizing our designs for different right-oriented communication patterns, namely (1) well-nested, width-1, (2) multicast, width-1, (3) well-nested, width- $w$ , (4) point-to-point, width- $w$ , and (5) the proposed multi-pattern framework.

For each design we have obtained the timing, area, and power reports.

Calculation of frequency from the timing report:

$$\text{Required time} = \text{phase shift} - \text{external delay},$$

where *phase shift* is the time period of the clock in the design and *external delay* is the time taken for the input signal to reach the input pin from the external point. The designer sets the phase shift during the synthesis process.

*Slack time* is the remaining time left in the clock time period after the signal reaches the output.

For a design to operate properly, the slack time has to be positive.

$$\text{Slack time} = \text{required time} - \text{arrival time}$$

where *arrival time* is the actual time at which the input signal arrives at the output.

$$\text{Propagation delay} = \text{phase shift} - \text{slack time}$$

$$\text{Frequency} = 1 / \text{propagation delay}$$

The synthesis tool estimates the slack time and required time based on the complexity of the design and the technology used.

The tables shown below give the results of area (sum=square microns), frequency (MHz), and power (mW) for different designs for 8-bit data width.

Table 5.1: Results for a single switch with constraint on time (8-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	900	2.66E-03	2.11	474
Multicast (width -1)	2186	8.64E-03	3.61	278
Well-nested (width- $w$ )	2047	0.0191	4.43	225.7
Point-to-point (width- $w$ )	2522	0.0216	4.61	217
Multi-pattern framework	5564	0.0439	4.49	222.7

Table 5.2: Results for a single switch with constraint on area (8-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	905	2.88E-03	2.12	471.7
Multicast (width -1)	2175	8.79E-03	3.94	253.8
Well-nested (width- $w$ )	2021	0.0177	3.73	268
Point-to-point (width- $w$ )	2472	0.0201	3.78	269.5
Multi-pattern framework	5612	0.0476	4.68	213.7

Table 5.3 Results for three switches in a CST (four PEs) with constraint on time (8-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	2331	0.0124	2.55	392.2
Multicast (width -1)	3303	8.4E-03	6.37	157
Well-nested (width- $w$ )	4411	0.0342	3.99	250.6
Point-to-point (width- $w$ )	6115	0.0714	4.45	224.8
Multi-pattern framework	11715	0.0854	6.93	144.3

Table 5.4: Results for three switches in a CST (four PEs) with constraint on area (8-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	2330	0.0125	2.52	396.8
Multicast (width -1)	3321	7.071E-03	6.58	152
Well-nested (width- $w$ )	4280	0.0326	3.82	261.7
Point-to-point (width- $w$ )	6174	0.0695	4.8	208.4
Multi-pattern framework	11413	0.082	7.20	139

Table 5.5: Results for seven switches in a CST (eight PEs) with constraint on time (8-bit data width).

Communication Type	Area (sum)	Power (mW)	Time (ns)	Max. Frequency (MHz)
Well-nested (width -1)	5535	0.0283	3.78	265
Multicast (width -1)	8010	0.0199	9.4	106.3
Well-nested (width- $w$ )	10636	0.1448	4.87	205.3
Point-to-point (width- $w$ )	19379	0.4982	5.46	183.2
Multi-pattern framework	30059	0.2973	9.83	101.8

Table 5.6: Results for seven switches in a CST (eight PEs) with constraint on area (8-bit data width).

Communication Type	Area (sum)	Power (mW)	Time (ns)	Max. Frequency (MHz)
Well-nested (width -1)	5535	0.0290	3.71	270.2
Multicast (width -1)	7858	0.0179	9.67	103.4
Well-nested (width- $w$ )	10438	0.1469	5.40	185
Point-to-point (width- $w$ )	19372	0.4870	5.67	176.3
Multi-pattern framework	29624	0.3030	9.42	106

Table 5.7: Results for 15 switches in a CST (16 PEs) with constraint on time (8-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	11971	0.0605	5.06	178.6
Multicast (width -1)	17227	0.044	12.24	81.7
Well-nested (width- $w$ )	22810	0.3955	7.13	140.25
Point-to-point (width- $w$ )	44639	1.3507	6.44	155.3
Multi-pattern framework	71542	1.29	15.34	66

Table 5.8: Results for 15 switches in a CST (16 PEs) with constraint on area (8-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	11972	0.0608	5.14	194.5
Multicast (width -1)	17045	0.0412	12.01	83.4
Well-nested (width- $w$ )	22320	0.3910	7.66	130.5
Point-to-point (width- $w$ )	44650	1.3921	6.77	144.7
Multi-pattern framework	71242	1.3984	14.54	68.7

The tables shown below give the results of area (sum=square microns), frequency (MHz), and power (mW) for different designs for 2-bit data width.

Table 5.9: Results for 15 switches in a CST (16 PEs) with constraint on time (2-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	10010	0.0168	4.57	218.8
Multicast (width -1)	14246	0.0396	11.70	85.4
Well-nested (width- $w$ )	19082	0.3763	7.53	132.8
Point-to-point (width- $w$ )	41720	1.3232	6.45	155
Multi-pattern framework	69111	1.29	14.25	70.17

Table 5.10: Results for 15 switches in a CST (16 PEs) with constraint on area (2-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	10003	0.0172	4.73	211.4
Multicast (width -1)	14095	0.0166	12.22	81.83
Well-nested (width- $w$ )	19530	0.3717	7.52	132.9
Point-to-point (width- $w$ )	41458	1.2794	6.65	150.38
Multi-pattern framework	68425	1.425	14.26	70.1

The tables shown below give the results of area (sum=square microns), frequency (MHz), and power (mW) for different designs for 32-bit data width.

Table 5.11: Results for 15 switches in a CST (16 PEs) with constraint on time (32-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	13071	0.2323	5.29	189
Multicast (width -1)	26891	0.789	14.37	69.58
Well-nested (width-w)	31953	0.3763	7.53	132.8
Point-to-point (width-w)	54107	1.3608	6.68	149.7
Multi-pattern framework	71356	1.500	15.36	65.1

Table 5.12: Results for 15 switches in a CST (16 PEs) with constraint on area (32-bit data width).

Communication Type	Area (sum)	Power (mW)	Time delay (ns)	Max. Frequency (MHz)
Well-nested (width -1)	13062	0.2306	5.26	190.5
Multicast (width -1)	26806	0.0757	14.01	71.3
Well-nested (width-w)	31939	0.4264	7.77	128.7
Point-to-point (width-w)	53990	1.3308	6.61	151.28
Multi-pattern framework	71316	1.511	15.45	64.72

Graphs are plotted as shown in Figures 5.9-5.11 comparing the results for a CST with four, eight, and 16 PEs for area, frequency, and power dissipation.

### 5.3.1 Area Analysis

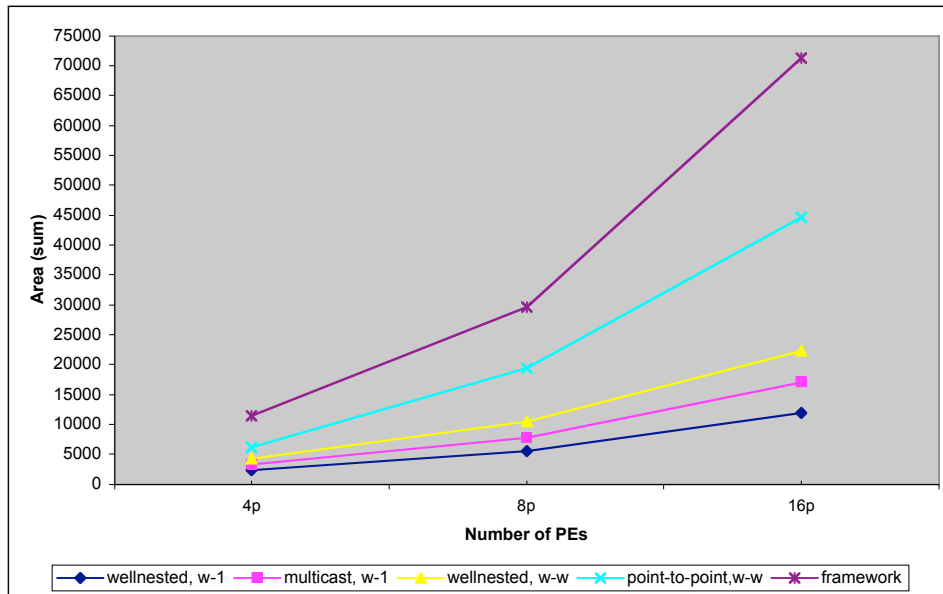


Figure 5.9: Graph comparing the area of each design.

The area increases with number of PEs for each communication pattern design, from a CST with four PEs to a CST with 16 PEs. We would expect the area of a CST with four PEs (three switches) to be greater than or equal to three times the area of a single switch. But from the results that is not true. The reason is that the synthesis tool flattens the identical modules to optimize the area. Since we perform the basic operation of the synthesis, the tools use the default in-built algorithm to position the modules in the design. The CST with 4 PEs has three identical switches, thus the tool overlaps some of the identical modules to reduce the area.

We would expect that the area of the CST with eight PEs should be greater than or equal to seven-thirds ( $7/3=2.33$ ) times the area of the four PEs designs. Similarly, we would expect that the area of the CSTs with 16 PEs should be greater than or equal to five ( $15/3=5$ ) times the

area of the four PEs designs. The results satisfy both the above expectations. The areas of the point-to-point communication set and multi-pattern design shows larger increase than other communication sets. This is due to the increase in the complexity of the design of the control unit for general, point-to-point communication sets as the number of communication IDs increases. The multi-pattern framework shows similar behavior since it includes the point-to-point communication set control unit.

As expected the sum of the areas of each communication pattern in a CST is more than the area required for the proposed multi-pattern framework. Thus, our framework performs the operation of all the four communication patterns in less area than implementing each switch separately.

Changing the data width from 8-bit to 2-bit and 32-bits affects the area. As expected the area for the 2-bit is less than 8-bit and the area of 8-bit is lesser than the area of 32-bit.

### 5.3.2 Frequency Analysis

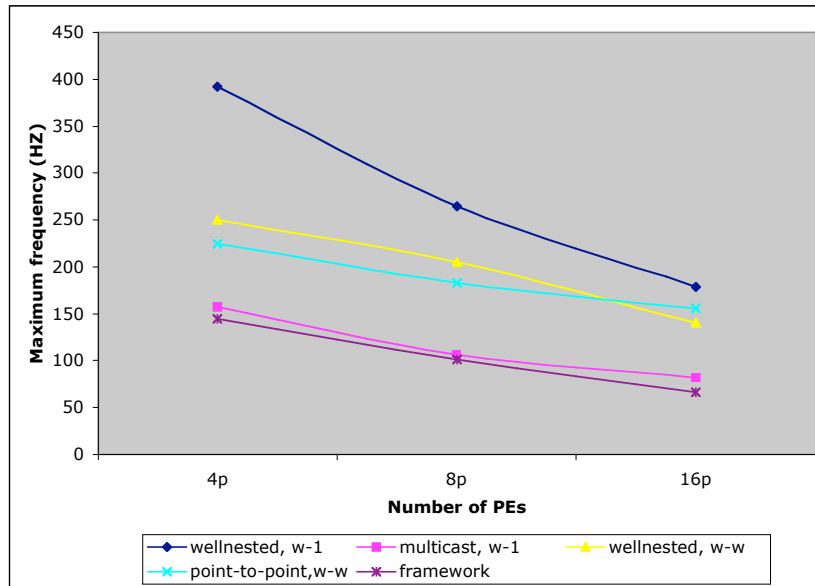


Figure 5.10: Graph comparing the frequency of each design.

The graph in Figure 5.10 shows a steady decrease in frequency as the number of PEs increases in the CST. We would expect the frequency to drop by half for a CST with four PEs compared to a single switch design, since the number of levels in the four PE CST is two. The results do not show this behavior. One of the reasons is because of the way the synthesis tool places modules to optimize the design to meet the time constraint (similar to the discussion in the area analysis section). The other reason is that not all switches in the CST are triggered using clock. Switches in the lower level are clocked and the switches above them trigger based on the input obtained from the previous level. If we compare the results for eight PEs and 16 PEs with those for four PEs, the delay increases constantly from two levels to three and four levels. The multicast width-1 design shows more delay as compared to other communication patterns except the design of the multi-pattern framework. This is because of the complexity of the implementation of the module shown in Table 4.4. The framework design requires more propagation delay than multicast width-1, since it includes the multicast design in it. As expected, the multi-pattern framework is the slowest compared to the individual communication patterns for four, eight, and 16 PEs. As expected the time delay for the 2-bit is less than 8-bit and the 8-bit is lesser than 32-bit.

### **5.3.3 Power Analysis**

The graph in Figure 5.11 shows the power dissipation (mW) for different implemented designs. The power dissipation increases with the number of PEs. The line for the multi-pattern framework and the point-to-point, right-oriented, width- $w$  communication set design has steepest increases due to the increase in number of connections and the complexity of the design. As expected the power dissipated for the 2-bit is less than 8-bit and the 8-bit is lesser than 32-bit.

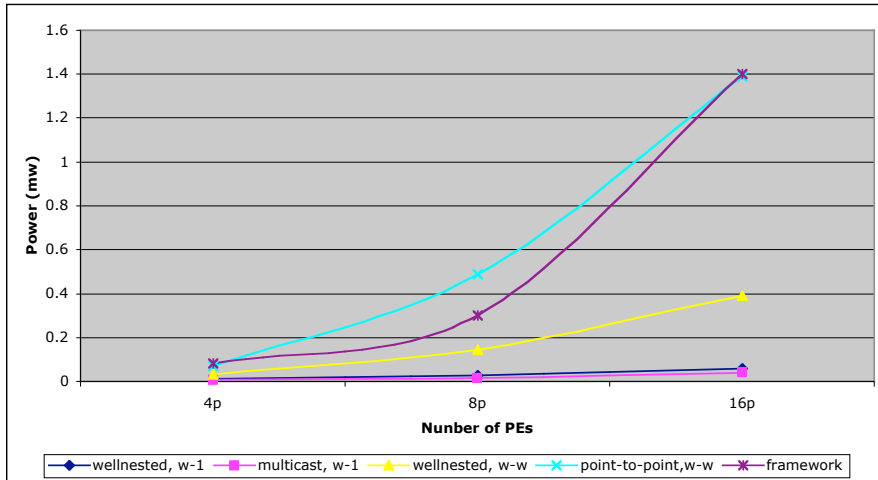


Figure 5.11: Graph comparing the power dissipation of each design.

## 6. Summary and Future Work

### 6.1 Summary

In this thesis, we have provided the designs and implementations of the routing algorithms for different communication patterns in a CST given by Roy *et al.* [RTV04, RVT05]. We have extended the work and implemented the algorithm for the point-to-point communication set. Finally, we have introduced a multi-pattern framework, which accommodates different communication patterns. We obtained the synthesis results by mapping our design with 0.25-micrometer technology. The results show that the proposed framework occupies less area as compared to the sum of the area occupied by other communication patterns discussed. We have successfully routed the data for different communication patterns in one clock cycle. (Chapter 5 provides the area, frequency and power analyses.) The results show that as the number of PEs in a CST increases, the area and power dissipation increase and the frequency drop gradually.

### 6.2 Future Work

This thesis work gives way to a number of future research directions. In this thesis, we have implemented only the right-oriented classes of communication set. One of the directions could be implementing a design that could accommodate both oriented and non-oriented communication sets. The control unit of the point-to-point, width- $w$  communication set is complex (due to the comparator), so one can try to reduce the complexity of the design. The framework presented in Chapter 4 has only four communication patterns, so accommodating more communication patterns and studying the behavior would be another future direction.

## References

- [BoP02] K. Bondalapati and V. K. Prasanna (2002), "Reconfigurable Computing Systems," *Proc. IEEE*, vol. 90, no. 7, pp. 1201-1217.
- [B04] A. Bindal (2004), "Synthesis and Timing Verification Tutorial," <http://www.engr.sjsu.edu/abindal/tutorials.htm>, Computer Engineering Department, San Jose State University.
- [CoH02] K. Compton and S. Hauck (2002), "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171-210.
- [Cadence 02] PKS User Guide, Product Version 5.0, May 2002.
- [Cadence 03] Command Reference for BuildGates Synthesis and Cadence PKS, Product Version 5.0.11, August 2003.
- [DV04] H. P. Dharamesena and R. Vaidynathan (2004), "The Mesh with Binary Tree Network: An Enhanced Mesh with the Low Bus-Loading". *The Journal of Interconnection network*, vol. 5, no. 2, pp. 131-150.
- [E103] H. M. El-Boghdadi (2003), "On Implementing Dynamically Reconfigurable Architectures," Ph.D. Thesis, Dept. Electrical and Computer Engr., Louisiana State University.
- [EIVTR02a] H. M. El-Boghdadi, R. Vaidyanathan, J. L. Trahan, and S. Rai (2002), "Implementing Prefix Sums and Multiple Addition Algorithms for the Reconfigurable Mesh on the Reconfigurable Tree Array," *Proc. 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, vol. 3, pp. 1068-1074.
- [EIVTR02b] H. M. El-Boghdadi, R. Vaidyanathan, J. L. Trahan, and S. Rai (2002), "On the Communication Capability of the Self-Reconfigurable Gate Array Architecture," *Proc. 2002 Reconfigurable Architectures Workshop*.
- [EIVTR03] H. M. El-Boghdadi, R. Vaidyanathan, J. L. Trahan, and S. Rai (2003), "On Designing Implementable Algorithms for the Linear Reconfigurable Mesh," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications*, pp. 241-246.
- [JK02] J. Torresen and K. A. Vinger (2002), "High Performance Computing by Context Switching Reconfigurable Logic." *Proc. 16th European Simulation Multiconference (ESM-2002)*, pp. 207-210.
- [P02] K. Puttegowda (2002), "Context Switching Strategies in a Run-Time Reconfigurable System," MSc. Thesis, Bradley, Dept. Electrical and Computer Engr.
- [LPPAJ02] D. I. Lehn, K. Puttegowda, J. H. Park, P. Athanas, and M. Jones (2002), "Evaluation of Rapid Context Switching on a CSRC Device," *Proc. Int'l. Conf. Engineering of Reconfigurable Systems and Algorithms (ERSA'02)*, pp. 154-160.

- [RTV04] K. Roy, J. L. Trahan, and R. Vaidyanathan (2004), "Configuring the Circuit Switched Tree for Point-to-Point and Multicast Communication," *Proc. IASTED Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, pp. 392-397.
- [RVT05] K. Roy, R. Vaidyanathan, and J. L. Trahan (2005), "Configuring the Circuit Switched Tree for Multiple Width Communications," *Proc. Workshop on Advances in Parallel and Distributed Computational Models (APDCM'05)*.
- [SMP99] R. P. S. Sidhu, A. Mei, and V. K. Prasanna (1999), "String Matching on Multicontext FPGAs using Self-Reconfiguration," *Proc. 1999 ACM/SIGDA 7th Int'l. Symp. Field Programmable Gate Arrays (FPGA '99)*, pp. 217-226.
- [SP01] R. Sidhu and V. K. Prasanna (2001), "Fast Regular Expression Matching using FPGAs," *Proc. 2001 IEEE Symp. Field-Programmable Custom Computing Machines*.
- [SP02] R. Sidhu and V. K. Prasanna (2002), "Efficient Metacomputation Using Self-Reconfiguration," *Proc. 12th Int'l. Workshop Field-Programmable Logic and Applic. (Lect. Notes Comput. Sci. # 2438)*, pp. 698-709.
- [SWMP00] R. Sidhu, S. Wadhwa, A. Mei, and V. K. Prasanna (2000), "A Self-Reconfigurable Gate Array Architecture," *Proc. 10th Int'l. Workshop Field-Programmable Logic and Applic. (Lect. Notes Comp. Sci. # 1896)*, pp. 106-120.
- [VT04] R. Vaidyanathan and J. L. Trahan (2004), *Dynamic Reconfiguration: Architectures and Algorithms*, Kluwer Academic/Plenum Publishers, New York.

## **Vita**

Dinesh Prasad Venkat Rao was born in India in 1981. He completed his high school in 1998 at Alpha Matriculation Higher Secondary School, Chennai, India. He entered the Department of Electronics and Communication Engineering, University of Madras, Chennai, India, in 1998 and obtained the degree of Bachelor of Engineering in June 2002.

He worked as a Research Associate for a year at Anna University, Chennai, India. In August 2003 he entered the graduate program in the Department of Electrical and Computer Engineering at Louisiana State University. During his enrollment in the graduate school, he served as a Graduate Assistant in Department of Psychology and Center for Computation and Technology as a Database programmer.