



National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file / Votre référence

Our file / Notre référence

NOTICE

AVIS

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

If pages are missing, contact the university which granted the degree.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

A Field-Programmable Gate Array Architecture Optimized for Datapaths

by

Donald W. Cherepacha

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Graduate Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada

© Copyright by Donald W. Cherepacha 1994



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-315-96285-2

Canada

Name Donald W. Cherepachuk

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Engineering - Electronics and Electrical

SUBJECT TERM

0554

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0351
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern 0401
Literature 0294
Classical 0255
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Marketing 0501
Canadian Studies 0503
Economics 0505
General 0508
Agricultural 0509
Commerce-Business 0510
Finance 0511
History 0358
Labor 0366
Theory 0351
Folklore 0578
Geography 0351
Gerontology 0578
History 0578
General

Ancient 0279
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0616
International Law and Relations 0617
Public Administration 0614
Recreation 0452
Social Work 0626
Sociology 0627
General 0938
Criminology and Penology 0631
Demography 0628
Ethnic and Racial Studies 0629
Individual and Family Studies 0630
Industrial and Labor Relations 0700
Public and Social Welfare 0344
Social Structure and Development 0709
Theory and Methods 0999
Transportation 0453
Urban and Regional Planning 0453
Women's Studies

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and Nutrition 0476
Animal Pathology 0359
Food Science and Technology 0478
Forestry and Wildlife 0479
Plant Culture 0480
Plant Pathology 0817
Plant Physiology 0777
Range Management 0746
Wood Technology
Biology 0306
General 0287
Anatomy 0308
Biostatistics 0309
Botany 0379
Cell 0329
Ecology 0353
Entomology 0369
Genetics 0793
Limnology 0410
Microbiology 0307
Molecular 0317
Neuroscience 0416
Oceanography 0433
Physiology 0821
Radiation 0778
Veterinary Science 0472
Zoology
Biophysics 0786
General 0760
Medical
Earth Sciences 0425
Biogeochemistry 0996
Geochemistry

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Polynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology 0354
Occupational Health and Therapy 0381
Ophthalmology 0571
Pathology 0419
Pharmacology 0572
Pharmacy 0382
Physical Therapy 0573
Public Health 0574
Radiology 0575
Recreation

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
General 0486
Agricultural 0487
Analytical 0488
Biochemistry 0738
Inorganic 0490
Nuclear 0491
Organic 0494
Pharmaceutical 0495
Physical 0754
Polymer 0405
Radiation
Mathematics 0605
Physics 0986
General 0606
Acoustics 0608
Astronomy and Astrophysics 0748
Atomic 0607
Atmospheric Science 0798
Fluid and Plasma 0759
Molecular 0609
Nuclear 0610
Optics 0752
Radiation 0756
Solid State 0611
Statistics 0463
Applied Sciences 0346
Applied Mechanics 0984
Computer Science

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451



THE UNIVERSITY OF TORONTO LIBRARY
MANUSCRIPT THESIS - MASTER'S
AUTHORITY TO DISTRIBUTE

NOTE: The AUTHOR will sign in one of the two places indicated. It is the intention of the University that there be NO RESTRICTION on the distribution of the publication of theses save in exceptional cases.

- a) Immediate publication in microform by the National Library is authorized.

Author's signature Don Leppa Date Sept. 29, 1994

- OR -

- b) Publication by the National Library is to be postponed until _____ 19 ____
(normal maximum delay is two years).

Author's signature _____ Date _____

This restriction is authorized for reasons which seem to me, as Chair of the
Graduate Department of _____, to be sufficient.

Signature of Graduate Department Chair _____

Date _____

BORROWERS undertake to give proper credit for any use made of the thesis, and to obtain the consent of the author if it is proposed to make extensive quotations, or to reproduce the thesis in whole or in part.

Signature of Borrower	Address	Date

A Field-Programmable Gate Array Architecture Optimized for Datapaths

Donald W. Cherepacha

Master of Applied Science

Department of Electrical and Computer Engineering

University of Toronto

1994

Abstract

This thesis presents a new Field-Programmable Gate Array architecture, called the Datapath-FPGA (DP-FPGA), which targets datapath intensive applications. By operating on data as a number of identically programmed 4-bit slices, the implementation density for these circuits is improved over the use of general-purpose FPGAs. The interconnection network incorporates distinct sets of resources for routing control and data signals. These features reduce circuit area by sharing configuration logic among 4-bit slices, lowering the total number of storage cells required.

This thesis discusses the requirements of logic block and routing structures that can be used to implement typical circuits containing a number of regularly structured datapaths of various sizes, which may include a small number of irregularities. Experimental results from a study of lookup table based datapath logic blocks show that the block with the smallest estimated area contains the following features: a 16-bit lookup table with four read ports, a dedicated carry chain using a bi-directional 4-bit carry skip circuit, a 4-bit register with enable and direct input capabilities, and four 3-state output drivers. Further estimates of DP-FPGA implementation area predict that the area of a design's datapath can be reduced by a factor of approximately two.

Acknowledgments

This thesis has provided me with a tremendous learning experience. I would first like to thank Professor David Lewis. His guidance, technical advice and friendship have been invaluable. In addition, I am grateful to Professors Jonathan Rose and Paul Chow, who were always available to answer my questions and add helpful input to this work. They, along with David Lewis and numerous professors and students, have made the University of Toronto a world leader in the area of FPGAs. I am proud to have been part of this group.

Thanks to the guys in LP392 who created an enjoyable working atmosphere. As well, thanks to the gang at LSI Logic who never missed an opportunity to remind me of deadlines come and gone.

I am truly grateful for the support that I have received from my family over the years. I consider myself a very lucky person to have such a family. There are no words which can sufficiently describe how much I appreciate all that my parents have done for me.

Finally, I would like to thank my fiance Julie, whose encouragement, patience and understanding have been an inspiration to me. I look forward to a lifetime of love and happiness together.

Financial support for this work by the Natural Sciences and Engineering Research Council of Canada and MICRONET was greatly appreciated.

Table of Contents

1	Introduction	1
1.1	Thesis Motivation	1
1.2	The Datapath-FPGA	3
1.3	Research Approach	4
1.4	Thesis Organization	4
2	Background Information	5
2.1	Introduction to FPGAs	5
2.2	Logic Block Architecture	8
2.2.1	Logic Block Functionality	8
2.2.2	Lookup Tables	10
2.2.3	Arithmetic Logic	11
2.2.4	FPGA Memory	12
2.3	Routing Architecture	13
2.3.1	Definition of Terms	13
2.3.2	Routing Organization	14
2.3.3	Pin-to-Channel Connections	16
2.3.4	Channel-to-Channel Connections	17
2.3.5	Segmentation	17
2.3.6	Dedicated Routing Resources	18
2.4	Application Specific FPGAs	19
2.5	Commercial Devices	20
2.5.1	Altera FLEX 8000	20
2.5.2	Xilinx XC4000	22
2.5.3	AT&T ORCA	25

3 The Datapath-FPGA Architecture	28
3.1 DP-FPGA Organization	28
3.2 Programming Bit Sharing	30
3.3 Applying Programming Bit Sharing to the DP-FPGA	34
3.3.1 Selecting N	34
3.3.2 Logic Block and Routing Granularities	36
3.4 Logic Block	37
3.4.1 Lookup Table	37
3.4.2 Carry Chain	39
3.4.3 Flip-Flops	42
3.4.4 Summary	43
3.5 Routing Architecture	43
3.5.1 Data Routing	44
3.5.2 Control Routing	47
3.5.3 Shift Block	48
 4 Investigation of Data Path Logic Blocks	 51
4.1 Logic Block Options	52
4.2 Experimental Procedure	56
4.2.1 Benchmark Circuits	56
4.2.2 Mapping Software	57
4.2.3 Area Model	63
4.3 Experimental Results	65
4.3.1 Flip-Flop Enables and Direct Inputs	66
4.3.2 Multiplexers and 3-State Buffers	68
4.3.3 Output Configuration	69
4.3.4 Lookup Table Size	70
4.3.5 Area Parameter Variation	71
4.4 Comparison with Commercial Architectures	71
4.5 Experimental Analysis of Bit Sharing	73

5	Conclusions and Future Work	76
5.1	Conclusions	76
5.2	Future Work	78
A	Datapath Netlist Format	79
	References	82

List of Figures

Figure 1.1	DP-FPGA Organization	3
Figure 2.1	Programmable Connection	6
Figure 2.2	Static RAM Programming Technology	7
Figure 2.3	Lookup Table Constructed Using a Multiplexer	11
Figure 2.4	Multi-Output Lookup Table	11
Figure 2.5	Symmetric Routing Architecture	14
Figure 2.6	Row-Based Routing Architecture	15
Figure 2.7	Pin-to-Channel Connections	16
Figure 2.8	Altera FLEX Logic Element (Normal Mode)	20
Figure 2.9	Altera FLEX Logic Element (Arithmetic Mode)	21
Figure 2.10	Altera FLEX Routing Architecture	22
Figure 2.11	Xilinx XC4000 Configurable Logic Block	23
Figure 2.12	Xilinx XC4000 Arithmetic Logic	24
Figure 2.13	Xilinx XC4000 Routing Architecture	25
Figure 2.14	AT&T ORCA Programmable Function Unit	26
Figure 2.15	AT&T ORCA Routing Architecture	27
Figure 3.1	DP-FPGA Organization	29
Figure 3.2	Section of Datapath	31
Figure 3.3	Mapping to an FPGA without programming bit sharing	32
Figure 3.4	Mapping to an FPGA with programming bit sharing	33
Figure 3.5	Carry Chain	42
Figure 3.6	Computation of P and G	43
Figure 3.7	Basic Logic Block Architecture	44
Figure 3.8	DP-FPGA General Routing Architecture	45
Figure 3.9	Shift Block	49

Figure 4.1	Dedicated multiplexer configuration	52
Figure 4.2	Potential output signals	54
Figure 4.3	Example of a logic block candidate (identified as d1e1k4m1o7)	55
Figure 4.4	Multiplexer example circuit	60
Figure 4.5	Multiplexer example implementation	61
Figure 4.6	Area vs. K when using dedicated enable logic and direct flip-flop inputs ..	67
Figure 4.7	Area vs. K using dedicated multiplexers and 3-state buffers	68
Figure 4.8	Area vs. number of output pins	70

List of Tables

Table 3.1	Area per bit for $\alpha = 2$ and $\alpha = 3$	36
Table 3.2	Pin-to-channel connection resources	46
Table 4.1	Output Formats	54
Table 4.2	Summary of logic block options	55
Table 4.3	Benchmark Circuits	56
Table 4.4	Summary of logic block area parameters	64
Table 4.5	Summary of Results	66
Table 4.6	The Effect of Bit Sharing on the Number of Logic Blocks	74

Chapter 1

Introduction

The market for Field-Programmable Gate Arrays (FPGAs) has experienced tremendous growth since their introduction in [1]. These highly integrated devices contain logic and routing structures that can be configured in the field, in contrast to the configuration of Mask-Programmed Gate Arrays (MPGAs) which requires metallization steps at a fabrication facility. The removal of the fabrication stage and its long turnaround time of several weeks can significantly reduce design cycle times. An FPGA can be programmed in minutes. In addition, the high NRE cost of creating metal masks makes conventional gate arrays more expensive for prototypes and low volume production. These factors make using an FPGA a lower risk proposition.

1.1 Thesis Motivation

FPGAs compete directly with MPGAs for ASIC designs of up to 20 000 gates. There are many advantages to using an FPGA, including faster time-to-market, lower cost on small volumes, increased flexibility and lower risk. However, for designs requiring large volumes and/or high gate counts, MPGAs remain the best solution due to their superior logic density. This work is devoted to reducing the area differences between FPGAs and MPGAs.

The logic density of an FPGA is approximately one-tenth that of an MPGA [2]. This is a direct consequence of the programmability of an FPGA. Presently, FPGAs use Static RAM cells, anti-fuses, EPROM transistors or EEPROM transistors to store configuration information. A pass-transistor controlled by one of these storage devices is typically used to join metal segments. By turning on the appropriate sequence of pass transistors, a complete routing connection can be made. In an MPGA, routing connections are made using only metal wire. Thus, the FPGA's programmable connections will be much larger, regardless of the programming technology used.

Although the density of an FPGA will fundamentally be lower than that of an MPGA, the gap can be reduced by improving the design of the FPGA's logic blocks and routing network, collectively referred to as its architecture. The selection of an architecture is dependent on the FPGA's intended applications. Commercially available FPGAs, such as those described in [3][4][5][6] are general-purpose and were developed for applications containing varying amounts of datapath and control logic. As such, they are flexible and enjoy widespread usage. However, their bit-level granularity in both the logic and routing structures can be inefficient for implementing wide datapaths. In particular, they are unable to take advantage of the regular bit-slice nature of datapaths which allows common resources to be shared among slices. As well, no distinction is made between data and control resources, even though their characteristics are quite different.

In full custom integrated circuits, the density of regular structures is roughly three times that of random logic. The goal of this work is to develop a datapath-oriented FPGA architecture, called the Datapath-FPGA (DP-FPGA), which similarly exploits the regularity of datapaths to gain area advantages over a general-purpose FPGA. Improving the density of designs with large amounts of datapath would make the FPGA a feasible implementation method for a larger range of digital circuits.

1.2 The Datapath-FPGA

The DP-FPGA is intended for a wide range of datapath-intensive circuits, such as those found in the areas of digital signal processing, communications, circuit emulation and special-purpose processors. The targeted systems may contain several datapaths of various widths, and may process a small number of bits in a different manner than the others. Thus, the DP-FPGA architecture must contain some programmability at the bit level, yet still take advantage of the high degree of regularity that exists in these circuits.

The DP-FPGA is organized as shown in Figure 1.1 with separate logic and routing resources to construct the three basic components of a system: control logic, datapath and memory. Area and performance advantages can be achieved by optimizing each section for its specific function. However, the flexibility of each is reduced which could lead to poor utilization of resources, increasing area and delay. This work concentrates on the datapath section which, due to its regularity, provides the potential for significant density improvements that overcome the loss of flexibility. The control section will likely resemble general-purpose architectures which are good at implementing random logic. High density blocks of RAM may be useful in large datapath designs. Since the target circuits are datapath intensive designs, the datapath section will occupy the majority of the total area. Thus, the ability to optimize the datapath will yield an indication of the feasibility of using specialized resources.

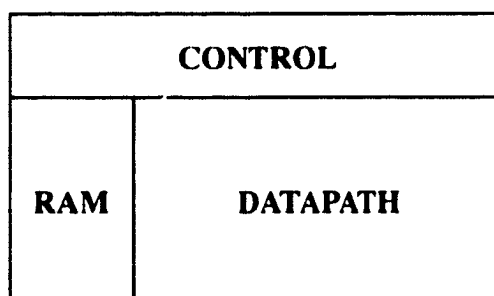


Figure 1.1 DP-FPGA Organization.

1.3 Research Approach

The number of possible choices for an FPGA architecture is unlimited. In this work, the problem is simplified by breaking it into two steps. First, by examining a number of target circuits, the way in which data is manipulated is informally characterized. These observations are used to propose a class of logic blocks and a routing structure meeting these requirements. The class of logic blocks is parameterized so that each potential block is defined by its parameter values. The routing architecture is left as a generalized network and is not investigated further in this work. The second step is an experimental evaluation of all of the logic block parameters to determine good values for each. In this investigation, a set of benchmark circuits is mapped into each potential architecture. The results of the mapping process are fed to a model estimating the area of each block, yielding the total implementation area of each circuit in each block which is used as a basis for comparison.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides background information on relevant issues relating to FPGA architectures, as well as a brief summary of selected commercial architectures. Chapter 3 introduces the idea of programming bit sharing and develops the basic DP-FPGA logic block and routing architectures. In chapter 4, an experimental analysis of the logic block is presented and the resulting architecture is compared with some commercial architectures. Finally, conclusions drawn from this work are summarized in chapter 5.

Chapter 2

Background Information

This chapter is intended to provide general background information concerning FPGAs and to introduce terminology and concepts used in later chapters. Section 2.1 provides a brief introduction to FPGAs and is followed by more detailed discussions of FPGA logic block and routing architectures in sections 2.2 and 2.3. Application specific FPGAs are discussed in section 2.4. In section 2.5, summaries of three commercial architectures, the Xilinx XC4000, the Altera FLEX and the AT&T ORCA, are given.

2.1 Introduction to FPGAs

An FPGA can be considered as the combination of an MPGA and a Programmable Logic Device (PLD). Similar to an MPGA, it consists of a regular array of logic blocks interconnected using a general routing network. Similar to a PLD, it is user-programmable. Its final digital functionality is determined by the user after the integrated circuit has been fabricated and packaged. Many FPGAs can be reconfigured repeatedly and can be reprogrammed in circuit, allowing complete systems to be field-programmable.

The method in which an FPGA is customized is referred to as its *programming technology*. The choice of programming technology has a direct impact on the density, performance and power consumption of the device. There are several different programming technologies

which are presently being used in commercial FPGAs, including those based on static RAM cells, anti-fuses, EPROM transistors, and EEPROM transistors.

Figure 2.1 illustrates a common technique used in customizing an FPGA, where a programmable connection element is used to join two metal wire segments. The programming element acts as a switch to provide low or high resistance between the two wire segments. Regardless of which programming technology is used, the area required is significantly larger than that of the hard-wired metal connections of an MPGA. The switch also introduces larger parallel capacitance and series resistance (in its *on* state). These factors make FPGAs slower than MPGAs in all programming technologies, although the values of R_{on} , C_{on} and C_{off} vary among them.

Since SRAM programming technology is discussed extensively in this thesis, its usage is considered in more detail. Further information concerning the other programming technologies mentioned can be found in [2]. An FPGA can be customized by loading bits of configuration data from off-chip permanent storage such as a PROM or disk into SRAM cells each time the chip is powered up. SRAM-based FPGAs can be reconfigured repeatedly and

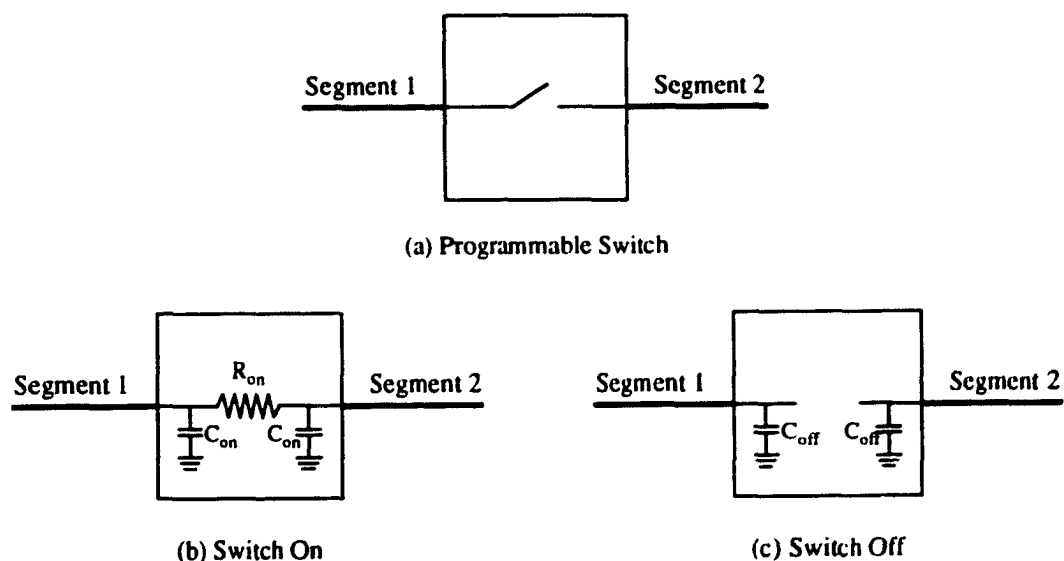


Figure 2.1 Programmable Connection.

can be reprogrammed in-circuit, making them suitable for applications in which the hardware is changed “on-the-fly”. Configuration bits stored in SRAM cells are typically used to control pass transistors, transmission gates and multiplexers or as storage for lookup tables. Examples of how SRAM cells are used to programmably connect wire segments are shown in Figure 2.2. They are used in lookup tables to store the truth table of a logic function, as described in section 2.2.2.

The configurability of an FPGA imposes fundamental reasons why its density and performance cannot match that of a conventional gate array. However, FPGA designers attempt to minimize these deficiencies through the choice of the FPGA’s architecture. An FPGA is generally comprised of three types of programmable resources, *I/O blocks*, *logic blocks*, and an *interconnection network*. The I/O blocks form the interface between the package pins and the internal logic. Normally, each I/O block is associated with a package pin and is configured to determine the pin’s directionality. As well, it may provide such features as signal registering and slew rate control. The logic blocks implement the digital functionality of the circuit and may be based on structures such as lookup tables, multiplexer trees, programmable logic arrays (PLAs) or NAND gates. The interconnection network defines the method of routing signals among logic block and I/O block pins. It typically consists of metal segments which can be joined using programmable switches. The architectures of the logic

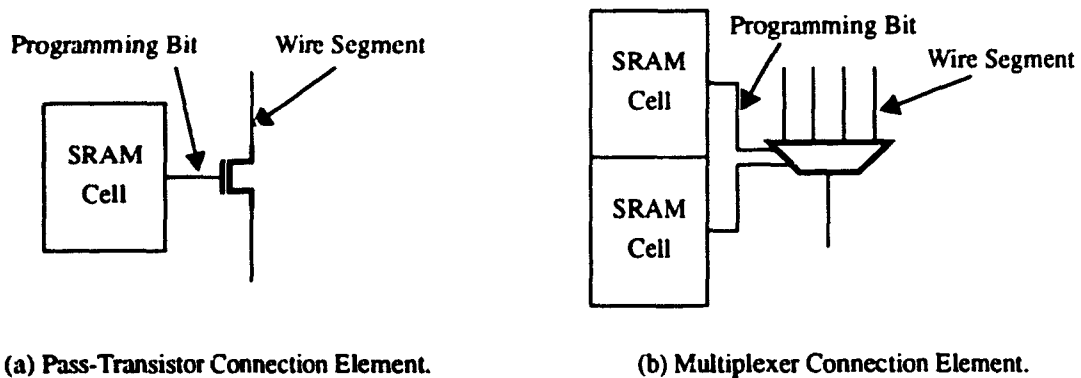


Figure 2.2 Static RAM Programming Technology.

block and interconnection network are the keys to determining both the density and performance of an FPGA and are the focus of this work. The I/O block will not be discussed further.

2.2 Logic Block Architecture

The architecture of the logic block has a significant influence on both density and performance. This section provides a brief description of the functions and characteristics of FPGA logic blocks. First, a general description is given, followed by a more detailed discussion of selected topics which are highly relevant to this work.

2.2.1 Logic Block Functionality

FPGAs contain an array of logic blocks which are normally identical (a homogeneous array), although there are exceptions [3][7]. The digital logic of a circuit is partitioned into individual blocks. The logic block itself may or may not be programmable. An important characteristic of a logic block is its level of functionality, referred to as its granularity. Logic block architectures range from small blocks which consist of only a few gates to large blocks capable of implementing thousands of different functions. There is a trade-off involved in the choice of granularity. Fine-grained blocks can be well utilized, but a high number of programmable connections are required, which is expensive in terms of chip area. Although increasing a logic block's functionality reduces the total number of blocks required, this can be more than offset by an increase in routing area due to a larger number of pin connections per block [8]. As well, it becomes more difficult to get good utilization of a block as its granularity increases, reducing the overall density.

Fine-grain blocks have two or three inputs and contain only a few transistors [9]. Their functionality is usually fixed. Examples include blocks using individual CMOS transistors and small logic elements [10], two-input NAND gates [11][12], or small combinations of multiplexers, NAND and XOR gates [13][14]. These blocks can be highly utilized since the

logic can be partitioned at such low granularity. However, a large number of them are needed to implement a circuit. This requires a significant amount of programmable routing to connect them, which is expensive in terms of both area and speed.

Medium-grain blocks can have as many as twenty inputs and can implement many more functions [9]. There are two broad categories of medium-grained logic blocks, differentiated by the main structure used to implement logic, namely multiplexers or lookup tables. Examples of multiplexer-based logic blocks include [3][7][20]. Lookup table based FPGAs, such as [4][5][6][15], store the truth table of a function which is addressed by its input signals. Medium-grained blocks have high functionality per input pin, which results in good area efficiency [8].

Large-grain blocks are often described as complex PLDs, using AND-OR arrays with large fan-in [9]. In addition, they may include some other gates to increase functionality. Examples of large-grain blocks include [17][18][19]. The large granularity means that fewer are required. However, the block usage tends to be low, reducing the overall density. As well, the high number of pins requires a large amount of area to make connections to the routing network. While not efficient in terms of area, large-grained blocks can achieve good performance since fewer levels of logic are required, avoiding much of the slow programmable interconnect [9].

FPGAs must have some means of implementing sequential logic. Most commercial FPGAs now explicitly include one or more flip-flops in the logic block, while some of the earliest FPGAs used feedback around combinational logic [11][13][20]. In their area efficiency study of lookup table based logic blocks, Rose et al. concluded that it is better to include a D flip-flop when implementing circuits containing sequential logic [8]. A flip-flop constructed using the same resources as combinational logic uses several logic blocks, increasing the total number of blocks as well as the amount of routing resources required. Their results showed that this more than offset the larger block size due to the embedded flip-flop.

In devices that explicitly contain a flip-flop, it is common to include programmable features which govern its use. Some allow the type of flip-flop operation (D, T, JK,SR) to be altered, while most have set/reset capabilities whose synchronicity may be programmable. The set/reset and clock signals may be local or global. An enable signal can be included to control the loading of the flip-flop. In some FPGAs the input to the flip-flop must come from the combinational logic output, while in others a direct input to the logic block may be selected instead.

2.2.2 Lookup Tables

A lookup table can be used to implement a combinational logic function by storing its truth table and using the function inputs to index into the table. The truth table of a K-input function requires 2^K memory cells. By loading different truth tables, any function of K inputs may be implemented. The number of functions that can be implemented by a K-input lookup table (commonly referred to as a K-LUT) is given by:

$$\text{Number of Possible Functions} = 2^{(2^K)}$$

A 4-LUT is capable of implementing 65 536 functions, while there are over four billion ways to program a 5-LUT. Thus, lookup tables provide high functionality per input pin.

Truth table data is stored in lookup tables during configuration. A value can be read from a K-LUT using a 2^K -to-1 multiplexer controlled by the address inputs, as shown in Figure 2.3. An N-to-1 multiplexer can be constructed as a tree of 2-to-1 multiplexers. Using this structure, a large lookup table can be divided into a number of smaller tables which have shared inputs. Figure 2.4 illustrates how a block of eight memory cells can be use to implement one 3-LUT or two 2-LUTs with common inputs. The restriction on sharing the inputs in the smaller LUTs can be removed by altering the multiplexer control inputs. Multi-output LUTs are used in logic blocks from AT&T [6] and Xilinx [15]. Lookup tables can also be cascaded to form a hierarchy with hard-wired links between them to improve both speed and density [5][21].

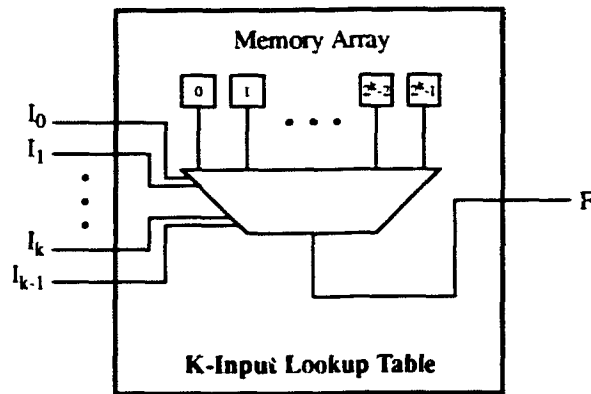


Figure 2.3 Lookup Table Constructed Using a Multiplexer.

2.2.3 Arithmetic Logic

Early FPGAs had logic densities of less than two thousand gates and were typically used for small controller applications. As densities reach tens of thousands of gates, much larger applications involving datapaths are being targeted for FPGAs. Arithmetic manipulation of data is typically poorly implemented (in terms of both speed and area) in general-purpose logic elements such as lookup tables. Carry propagation across the width of the datapath

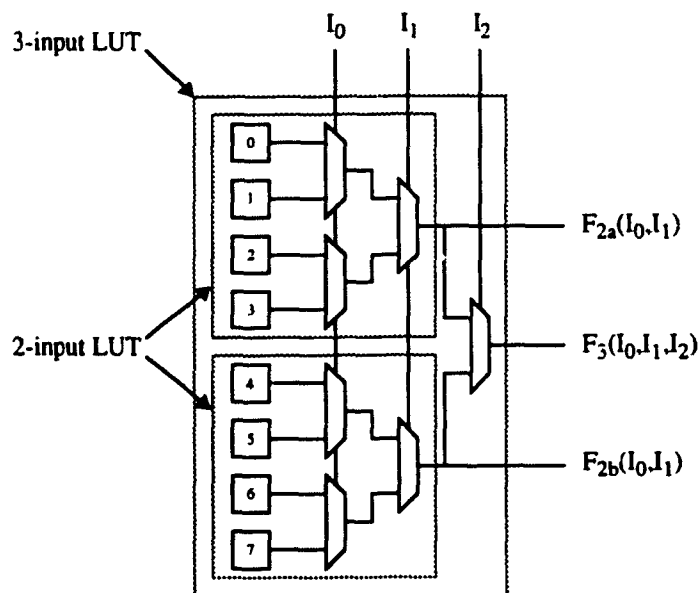


Figure 2.4 Multi-Output Lookup Table.

through programmable logic and routing will limit the speed of a design. A ripple carry adder requires two single-output lookup tables per bit, which is inefficient. Lookahead techniques can improve speed at the expense of up to double the number of logic blocks.

A technique used to improve both the speed and area of an FPGA is to replace programmable logic with dedicated resources. Many commercial FPGAs now include dedicated arithmetic logic in the logic block [4][5][6]. Typically, dedicated carry circuitry is used to create a fast path from carry-in to carry-out. The non-critical sum function is often generated using the normal programmable function generators. In addition, dedicated routing segments are supplied to connect the carry-out of one block to the carry-in of the next. This significantly reduces the amount of programmable switches on the path over the use of general-purpose interconnect. Three commercial examples of this feature are described in section 2.5.

2.2.4 FPGA Memory

Memory is an important component of a digital system. The only memory contained in most FPGAs is that of the flip-flops located within logic blocks. These flip-flops are suitable for tasks with small storage requirements such as controller states, pipeline registers and counters. Implementing larger structures such as register arrays, FIFOs or blocks of RAM is impractical in such low density, highly distributed storage devices. These structures must be implemented using off-chip memory resources. Since chip-to-chip data transfers are significantly slower than on-chip transfers, system speeds are reduced. As well, off-chip solutions are I/O intensive, requiring memory data and address buses to be sent off-chip.

Some commercial FPGAs, such as the Xilinx XC4000 [5] and AT&T ORCA [6] families, utilize higher density memory components. They allow the SRAM in 4-LUTs to be used as a 16-bit memory array. Multiple arrays can be combined to build larger memories. These techniques are flexible and allow larger memories to be constructed on-chip, but densities are still too low for many applications.

Practical on-chip memory solutions must be based upon high density memory arrays. They must also be highly flexible, yielding numerous configurations in order to achieve widespread usage in FPGA applications. An example of such a *Field-Reconfigurable Memory* is the FiRM chip [22] developed at the University of Toronto which consists of four 1 Kbit high-density RAM blocks. This chip provides 203 memory configurations. As with all specialized resources, the utilization of high density RAM blocks must be sufficiently high to justify the costs. This is especially true with large RAM arrays which consume significant area. The large variations in size and organization of memories restricts the range of applications for which a given architecture will be cost-effective.

2.3 Routing Architecture

The area taken up by the programmable interconnection network can take up 50-90% of an FPGA's total area [8]. As well, much of the performance limitations faced by FPGAs can be attributed to slow programmable connections. In this section, a brief survey of some of the main issues involved in the choice of routing architecture is given.

2.3.1 Definition of Terms

In order to facilitate the discussion of routing architectures in this thesis, some terminology is given, as defined in [2]:

- *pin* - a logic block input or output.
- *connection* - a pair of logic block pins that are to be electrically connected.
- *net* - a set of pins that are to be electrically connected. A net can be divided into one or more connections.
- *wire segment* - a straight section of wire that is used to form part of a connection.
- *routing switch* - a programmable switch used to electrically connect two wire segments.
- *track* - a straight section of wire that spans the entire width or length of a routing channel. A track can be composed of a number of wire segments.
- *channel* - the rectangular area that lies between two rows or two columns of logic blocks. A routing channel contains of number of tracks.

2.3.2 Routing Organization

In most FPGAs the routing resources are organized as vertical and horizontal channels. Exceptions are FPGAs which use a sea-of-gates style routing in which all routing resources overlay the logic blocks [11][13][14]. A characteristic differentiating architectures which use routing channels concerns the symmetry of the horizontal and vertical channels. Two common organizations, fully symmetric and row-based, are discussed below.

An architecture with symmetric horizontal and vertical channels is depicted in Figure 2.5. Examples of FPGAs which use this style of routing include [5][15][17]. Channels of both orientations contain approximately the same number of tracks. Pin-to-channel connections

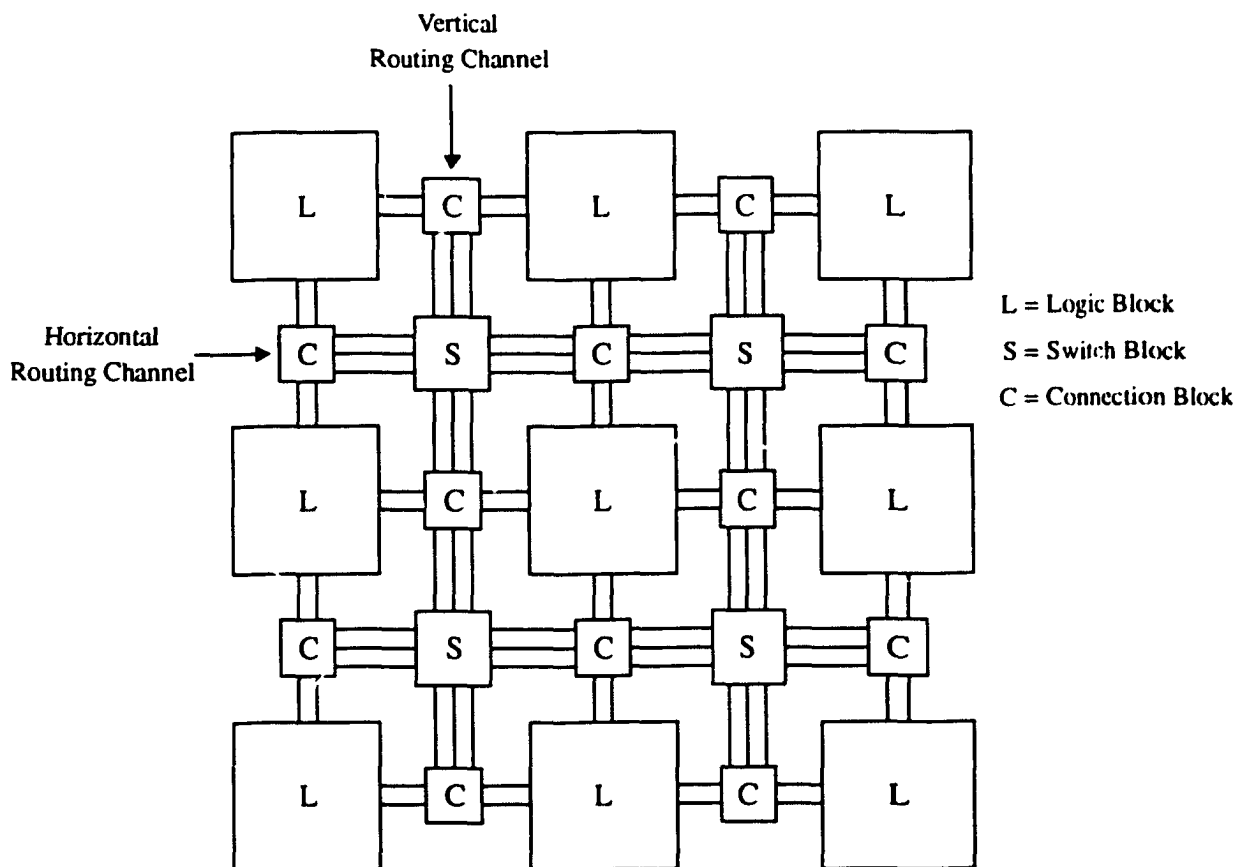


Figure 2.5 Symmetric Routing Architecture.

can be made between a logic block and all four neighbouring routing channels. This has an effect on the placement of logic blocks as there is no directional bias [9].

Row-based routing architectures have a structure similar to that used in conventional gate arrays. The chip is organized as alternating rows of logic blocks and routing channels, as shown in Figure 2.6. Examples of FPGAs with this structure include [3][20]. There are many more horizontal tracks than vertical tracks and logic block pins connect only to tracks in horizontal channels. As a result, placement tools should be biased to create a higher percentage of horizontal connections.

The interconnection network may also have a hierarchical organization with different features at each level. For example, logic blocks may be grouped and interconnected with local routing. These groups can then be connected at a higher level using global routing, forming a

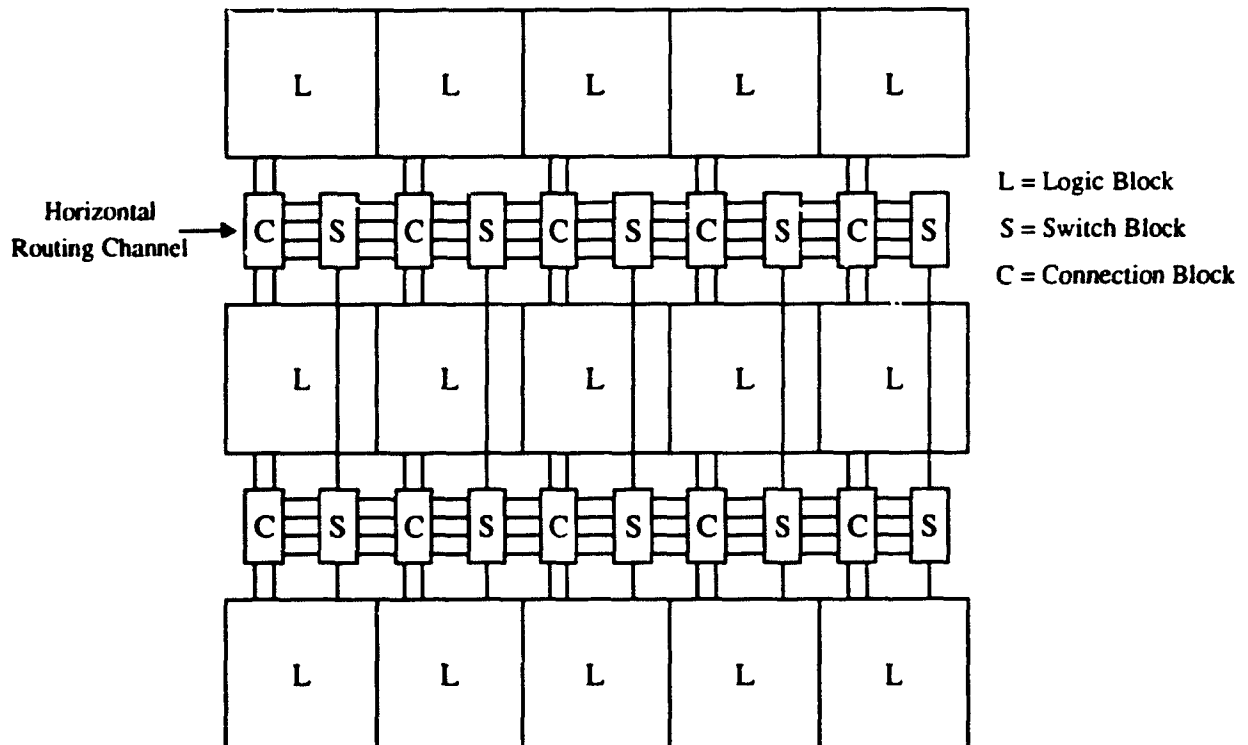


Figure 2.6 Row-Based Routing Architecture.

two-level hierarchy. The Altera FLEX FPGA contains this type of routing organization [4]. More generalized hierarchical routing structures are described in [23].

2.3.3 Pin-to-Channel Connections

The set of resources used to connect logic block pins to a routing channel is called a *connection block* [24]. Each pin may be capable of connecting to one or more tracks in a channel, referred to as the *flexibility* of the connection block. Increasing the population of a connection increases routability, but also increases the connection block area. Experimental results indicate that high flexibility is essential for the connection block [24]. In the Xilinx XC4000 series FPGA, input pin connections are highly populated, while output pins are connected to only a few tracks [5].

When connecting several tracks to a pin, there are numerous topologies. A single switch can be placed between the pin and each track (fully de-multiplexed) or a multiplexing scheme can be used. These two cases are illustrated in Figure 2.7. The delay incurred using the direct switch method is shorter since there is only one switch along the path, whereas the multiplexer

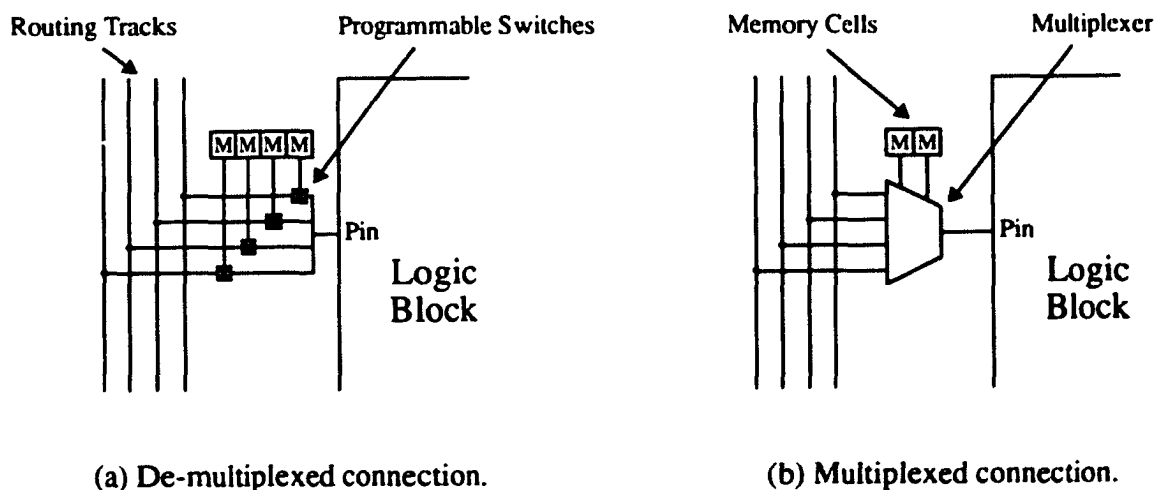


Figure 2.7 Pin-to-Channel Connections.

technique has several. Other advantages of the direct method include the abilities to connect a pin to several tracks and to make connections between tracks, both of which add flexibility to the routing structure. However, multiplexing techniques require fewer configuration bits. This is important when using a large programming technology such as SRAM. The choice of connection method is also influenced by the directionality of the pin. Output pins are usually connected to the routing channel using direct switches to allow fanout of the signal to several tracks. Since an input to a block is normally connected to only one track, both methods are suitable.

2.3.4 Channel-to-Channel Connections

The set of resources used to make connections between two routing channels is called a *switch block* [24]. The structure of the switch block affects the routability, density and performance of a design. The issues considered in the examination of pin-to-channel connections apply to channel-to-channel connections as well.

In switch blocks, direct switches between horizontal and vertical channels are normally used in all but the fine-grain architectures, where multiplexers are common. Rose and Brown showed that good routability can be achieved with low flexibility of channel-to-channel connections, provided that there are sufficient tracks per channel [24].

2.3.5 Segmentation

Routing tracks can be made up of a number of wire segments. The *length* of a segment is the number of logic blocks that it spans. The shortest possible segment has length one, while the longest spans the entire length or width of the chip and is often referred to as a *long line*. The number of each type of segment is called the *segment distribution* [9].

The segmentation of an architecture affects both density and performance. Longer segments improve performance as there are fewer switches along a path (less series resistance). As well, since fewer switches are needed, the area taken up by tracks with long segments is smaller

than those with shorter segments. However, the utilization of long segments is poorer than for short segments since a path can be partitioned into finer grained elements in the latter case. This results in more tracks being needed to achieve the same level of routability. Thus, there is an area trade-off that must be considered.

2.3.6 Dedicated Routing Resources

The density and performance of an FPGA can be improved through careful use of dedicated resources. A dedicated resource is one that is included for a specific purpose (e.g. carry chain), as opposed to general use (e.g. lookup table). By optimizing a resource for a specific purpose, the number of programmable elements needed is reduced, yielding speed and area improvements. However, dedicated resources are less flexible by definition and often remain unused, reducing the overall density. Dedicated routing resources can be divided into two categories, global or local. These cases will be discussed in sequence.

Dedicated global resources are hard-wired metal networks which cover the entire chip, allowing access to all logic blocks. Clock and reset nets are often given global dedicated resources. A critical issue related to clock nets is that of timing skew. The high capacitance associated with programmable routing can cause the amount of skew on large nets to be great. Since dedicated global resources contain no programmable channel-to-channel connections and few (if any) pin-to-channel connections, the amount of timing skew is reduced significantly. As well, the delay on these critical nets is much shorter than if general interconnect is used. Dedicated global lines also improve density if their utilization is high since the area required for a global network is much smaller than the large amount of programmable interconnect that would otherwise be required.

Dedicated wiring can also be used effectively for short, local connections. One example, the use of dedicated lines to route carries between blocks, was explained in section 2.2.3. Another takes advantage of the fact that there are a large number of connections from the output of a

block to the inputs of neighbouring blocks or even to its own inputs (feedback of a registered value). These connections will be referred to as *nearest neighbour* connections. To efficiently support such connections, some devices provide a direct line to one or more neighbouring blocks [15]. If these lines are highly utilized, the amount of area-expensive general-purpose interconnect required will be reduced, improving density. As well, the delay on direct lines is shorter, improving performance.

2.4 Application Specific FPGAs

Presently available commercial FPGAs can be classified as general-purpose, not having been designed for any particular application. They are flexible and can be used in a wide variety of applications. However, the characteristics of digital circuits can vary significantly from circuit to circuit or even between modules of a single circuit, making no single architecture best for all applications. For example, the best FPGA architecture for a bus controller would likely be much different from that for a digital signal processor (DSP). FPGAs which are targeted towards a certain class of digital circuits are called *application specific FPGAs*. By narrowing the range of target circuits, additional optimization is possible. The penalty is a smaller target audience.

The application specific paradigm has been applied to the field of DSPs. In [25], Agarwala and Balsara introduced a DSP logic module which is particularly efficient at implementing multiplexers and adders which are a DSP's main building blocks. By optimizing the logic block for the dominant functions at the expense of other functions, they were able to achieve better performance than a comparable general-purpose architecture, without sacrificing area.

In contrast to the bit-level granularity of the previous DSP architecture, Chen and Rabaey [26] have developed a field-programmable multiprocessor targeted for high performance DSP applications. It operates at a word level and uses a cluster of execution units interconnected by a configurable crossbar switch.

2.5 Commercial Devices

In this section, summaries of three currently available commercial devices, the Altera FLEX 8000, the Xilinx XC4000 and the AT&T ORCA, are given. These FPGAs provide examples of many of the architectural features that are discussed in subsequent chapters. As well, these three FPGAs are compared with the DP-FPGA in chapter 4.

2.5.1 Altera FLEX 8000

The FLEX 8000 series is Altera Corporation's highest density family of programmable devices with gate counts ranging from 2500 to 24000 usable gates [4]. Altera FLEX FPGAs utilize CMOS SRAM programming technology with lookup table based logic blocks and a hierarchical routing structure. The FLEX architecture is organized as a matrix of Logic Array Blocks (LABs). Eight logic blocks, called Logic Elements (LEs), are contained within each LAB. With this hierarchical structure, the density advantages of a medium-grained architecture at the lower level are combined with high speed and predictable routing delays of a coarse-grained architecture at the higher level.

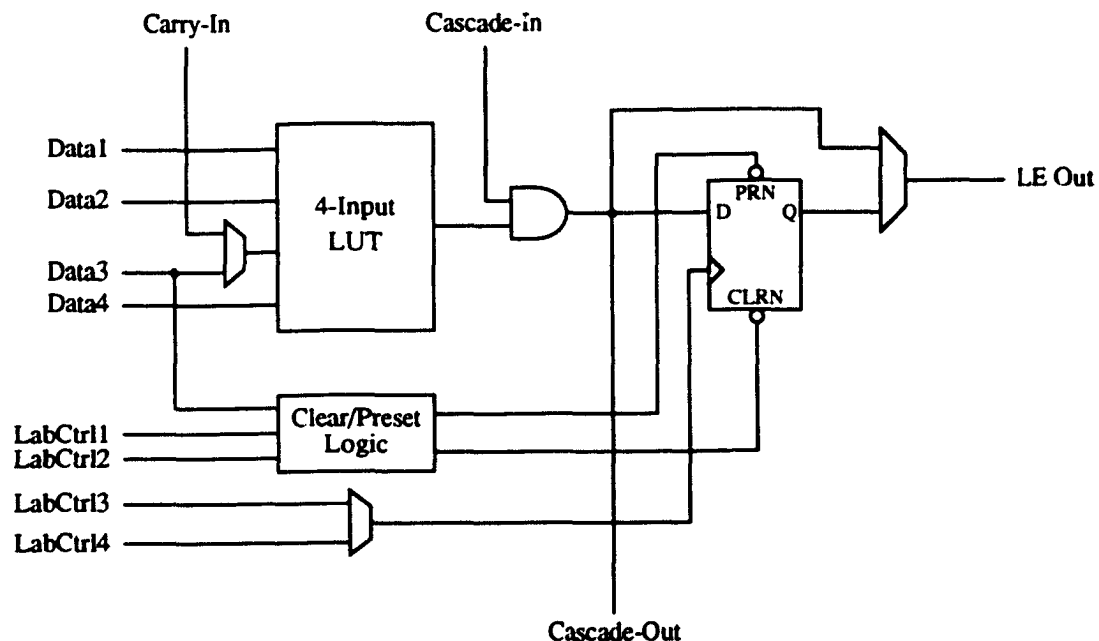


Figure 2.8 Altera FLEX Logic Element (Normal Mode).

Each LE contains a four-input lookup table, a programmable flip-flop, a carry chain and a cascade chain. An LE operating in its normal mode configuration is illustrated in Figure 2.8. There is a single output which may come from either combinational logic or a flip-flop. For arithmetic logic, the lookup table can be programmed to produce both sum and carry outputs, as shown in Figure 2.9. Dedicated routing is used to transfer the carry-out of one LE to the carry-in of the next, allowing a ripple-carry adder of arbitrary size to be created. The cascade chain is used for functions which have wide fan-in. The cascade output of one LE is connected directly to the cascade input of an adjacent LE.

The Altera FLEX FPGA uses a two-level hierarchical routing structure, as shown in Figure 2.10. The lower level consists of the LAB local interconnect and the upper level is comprised of the FastTrack global interconnect. Within each LAB, the local network routes the LAB input and local feedback signals among the 8 LEs. Four control signals (for 2 clocks, preset and clear) are common to all 8 LEs and may come from the local interconnect or low-skew dedicated global lines. The FastTrack interconnect is organized as rows and columns for inter-LAB routing and connections to the I/O blocks. All LAB inputs come from the above row.

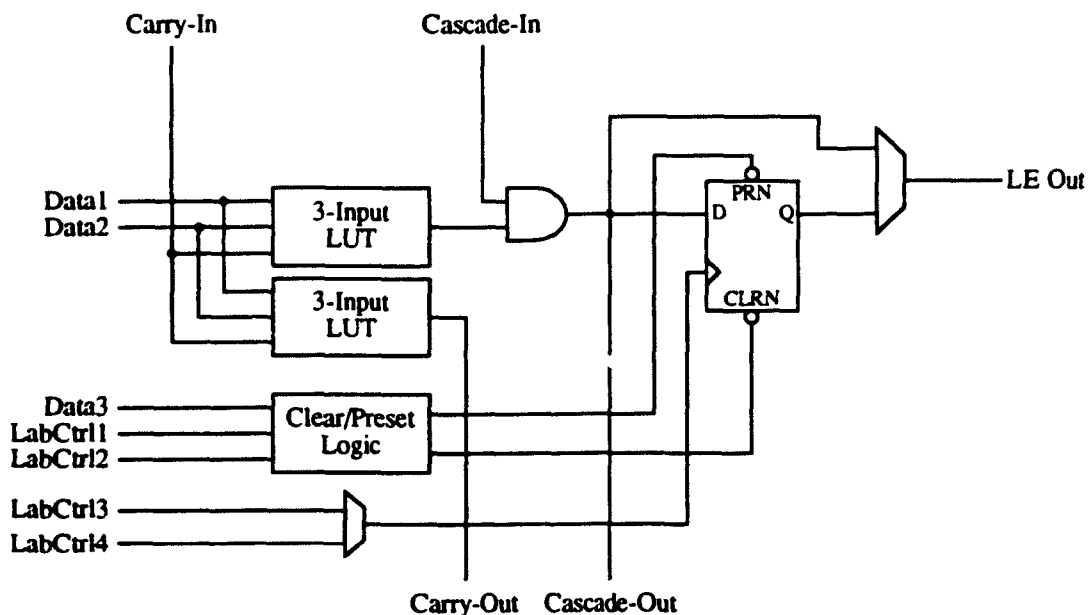


Figure 2.9 Altera FLEX Logic Element (Arithmetic Mode).

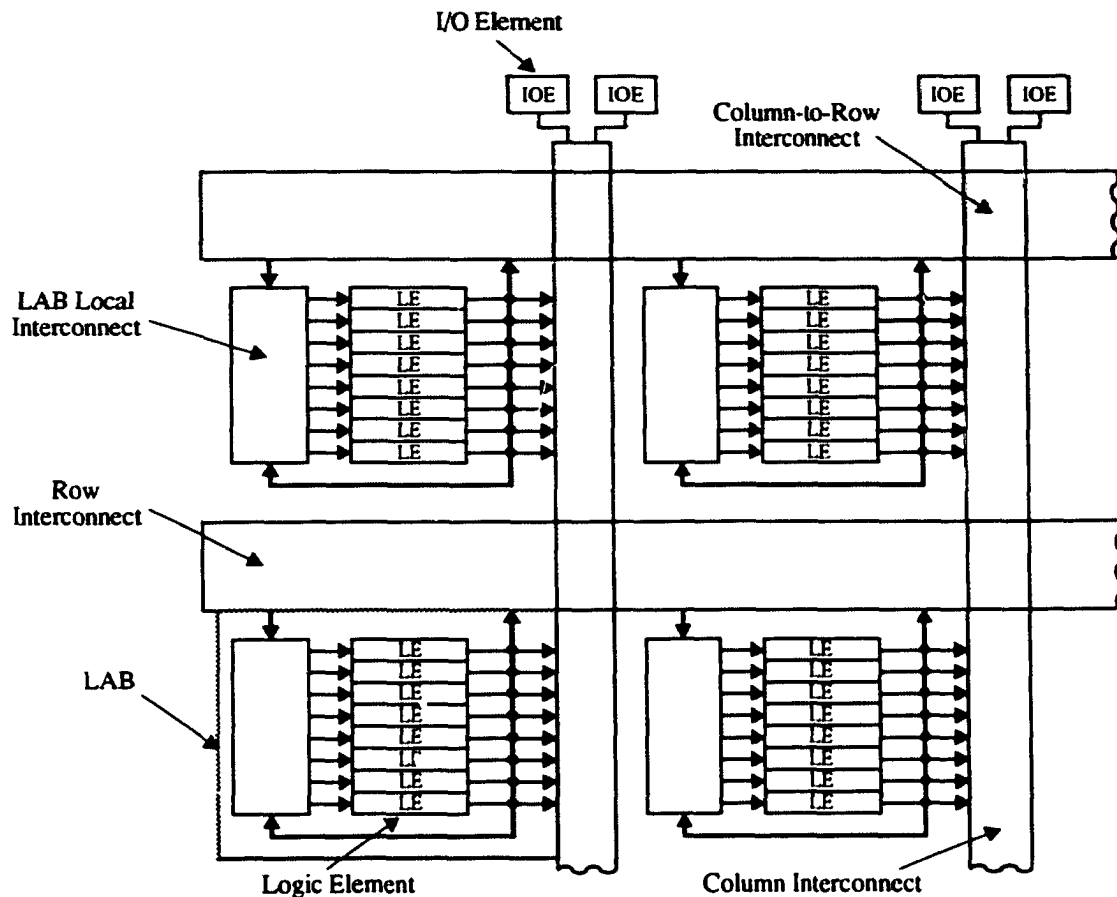


Figure 2.10 Altera FLEX Routing Architecture.

LAB outputs may be driven on both the above row and right side column. Column-to-row switching takes place at the intersections. There are 16 routing tracks per column and 168-216 tracks per row (depending on the device). Unlike segmented routing architectures, the device wide interconnection lines of the FastTrack architecture allows predictable performance.

2.5.2 Xilinx XC4000

The XC4000 family represents the third-generation of FPGAs from Xilinx. Its members range in size from 2000-25000 gates [16]. The XC4000 FPGA uses CMOS SRAM programming technology with a two-dimensional array of lookup-table based logic blocks. The programmable interconnect is organized as symmetrical horizontal and vertical routing channels.

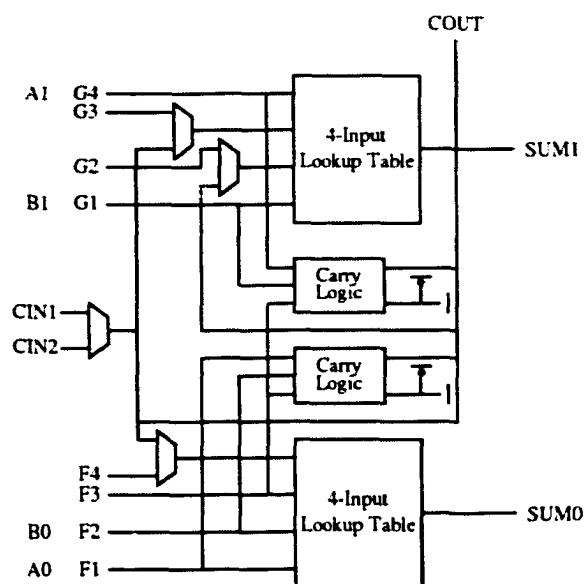


Figure 2.12 Xilinx XC4000 Arithmetic Logic.

Figure 2.12. A sum bit is calculated in each 4-LUT, while dedicated logic produces the ripple carries. Dedicated routing is used to pass carries between blocks for fast propagation across wide datapaths. Another special feature of the XC4000 is the ability to use the storage cells of the two 4-LUTs as RAM. The LUT inputs become the address inputs and the CLB control inputs are used as the data inputs and the write enable. Each CLB can be configured as a block of 16x2 or 32x1 bits of RAM. The 3-LUT and flip-flops can still be used in their normal manner while in RAM mode.

Connections between CLBs are made using the programmable interconnect which is arranged in a grid of rows and columns, as shown in Figure 2.13. A switch matrix is located at each intersection of the routing channels. Signals entering a switch matrix can be passed directly through or switched left or right onto the crossing row or column. The routing tracks are segmented with lengths of one or two. There are also long lines in each channel, which may optionally be driven using 3-state drivers.

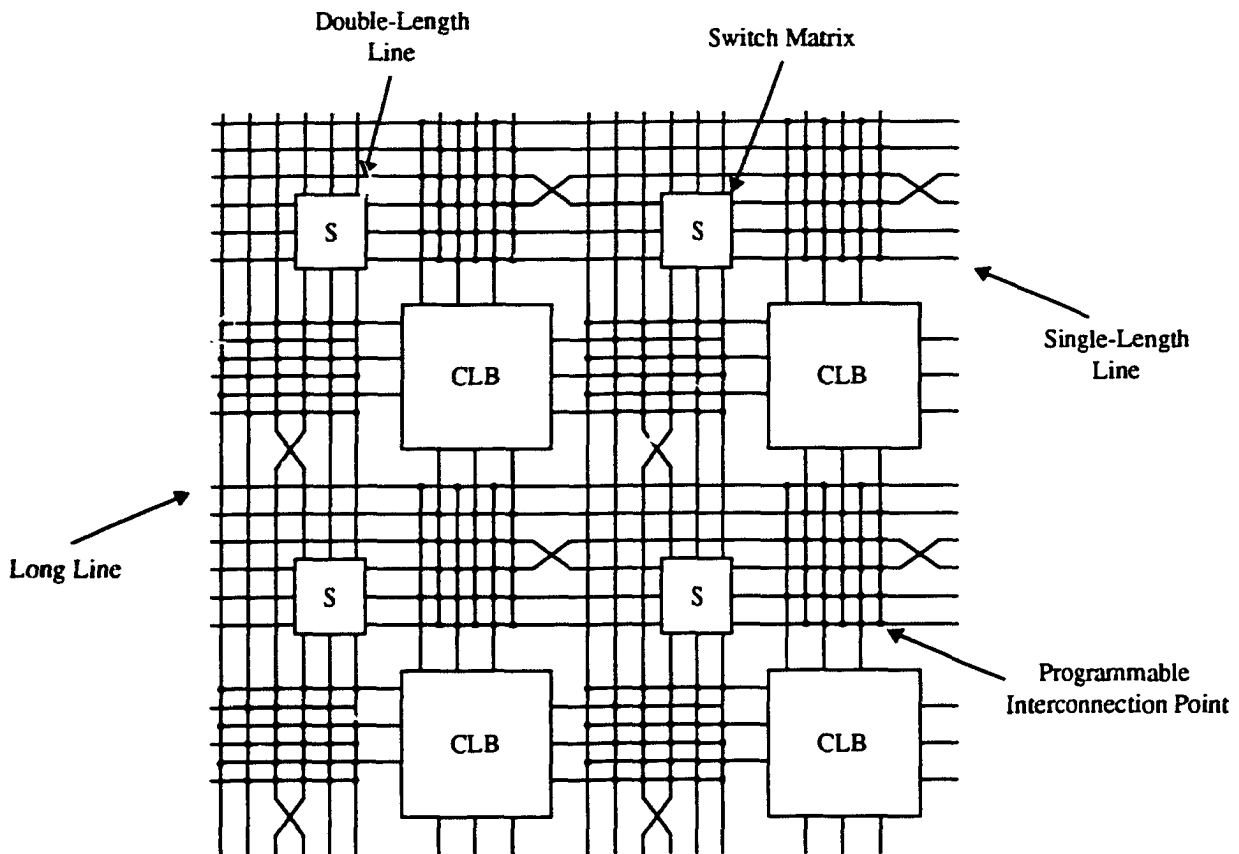


Figure 2.13 Xilinx XC4000 Routing Architecture.

2.5.3 AT&T ORCA

AT&T's ORCA family of high density FPGAs were designed to support applications with datapaths and on-chip RAM without degrading support for random logic [6]. They utilize SRAM programming technology with lookup-table based logic blocks. The ORCA architecture is both homogeneous and symmetrical.

The ORCA logic block, referred to as a Programmable Function Unit or PFU (Figure 2.14), can be used for three types of logic: random, datapath and RAM. For random logic, the ORCA PFU provides the largest lookup table of any commercially available FPGA, with 64 storage cells. The lookup table is highly configurable and can be used to implement four 4-input functions, two 5-input functions, one 6-input function or selected functions of up to 11 inputs.

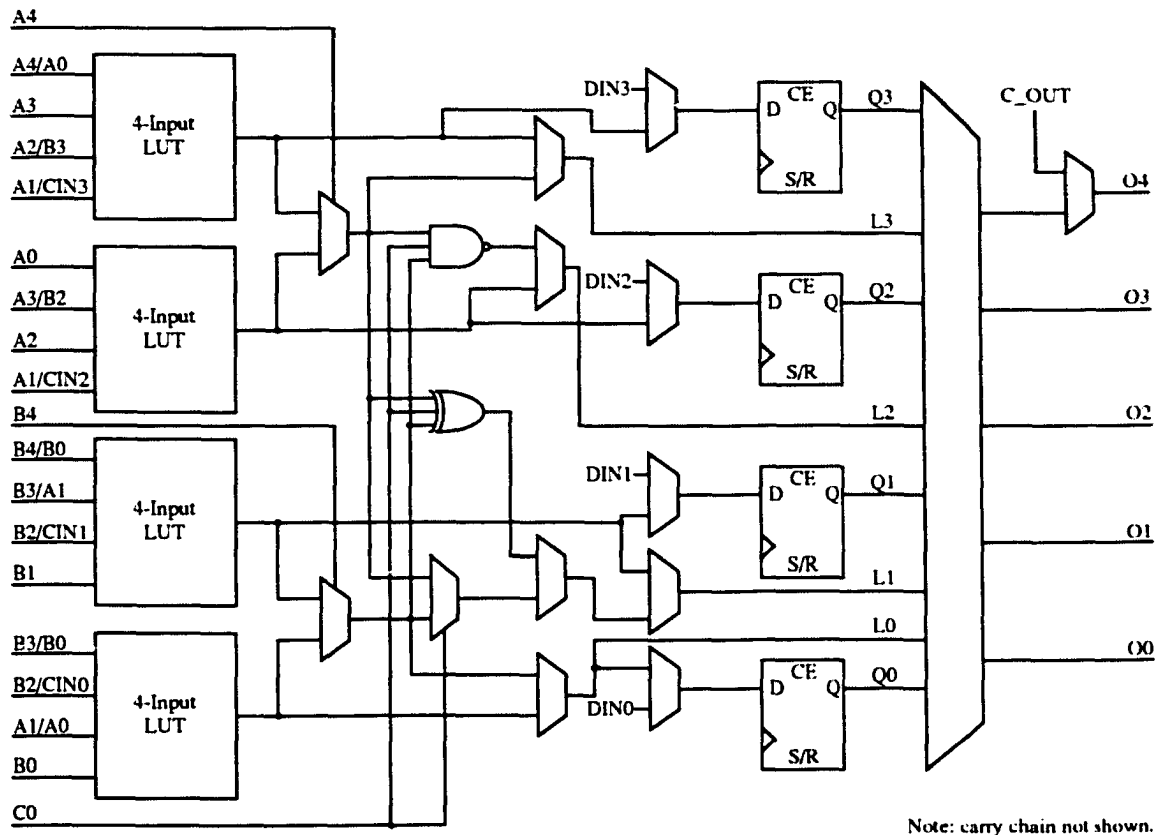


Figure 2.14 AT&T ORCA Programmable Function Unit.

For datapaths, the PFU can be used to perform bit operations on two 4-bit data buses. A fast carry circuit is used to improve arithmetic performance. Carries can be rippled to adjacent PFUs using dedicated routing. When used as memory, the lookup table is divided into two halves, each implementing a 16x1 RAM. To support deeper memories, four 3-state buffers are provided to allow 16-bit banks to be selectively enabled. Four flip-flops are provided in each PFU. Each flip-flop can be configured using an enable, a global set/reset and a local set/reset. The inputs may come from the lookup table outputs or dedicated logic block input pins.

The interconnection network uses a topology which AT&T refers to as Multi-Level Hierarchical Routing with a Distributed Switch Matrix. This architecture is illustrated in Figure 2.15. PFU pins are called Access R-nodes and are connected to metal segments

referred to as Switching R-nodes through programmable switches. In turn, Switching R-nodes can be configurably connected to Inter-Block R-nodes. Inter-Block R-nodes are segmented with lengths of 1, 4 or the length of a row or column (long lines).

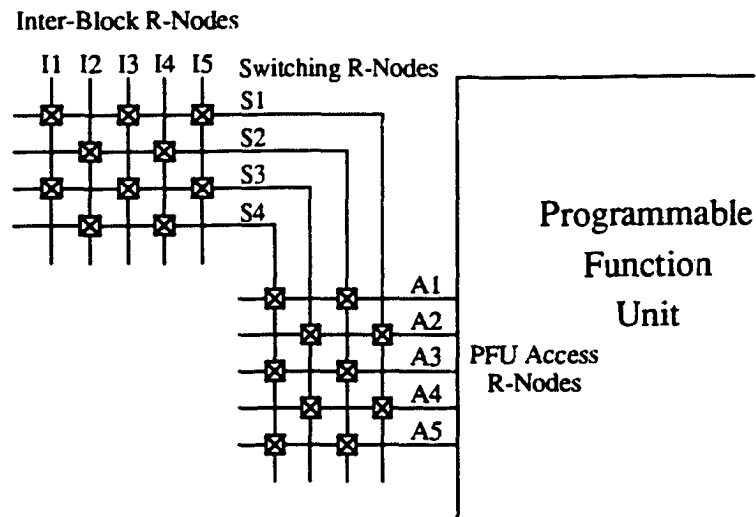


Figure 2.15 AT&T ORCA Routing Architecture.

Chapter 3

The Datapath-FPGA Architecture

This chapter presents the general architecture of the Datapath-FPGA. Its top-level organization is described, dividing the device into three major sub-components: datapath, control and memory. The remainder of the chapter focuses on the architecture of the datapath. The area of the datapath section is reduced by exploiting the unique characteristics of datapaths. The primary technique used, known as programming bit sharing, is explained in detail. This leads to a discussion of the granularity of the logic blocks and the use of data buses. Finally, the two main components of the architecture, the logic block and the interconnection network, are discussed and it is shown how the extensive use of programming bit sharing and other datapath specific features can improve the density of datapaths in FPGAs.

3.1 DP-FPGA Organization

The DP-FPGA architecture attempts to reduce the density gap between conventional gate arrays and FPGAs when implementing datapath oriented designs. This application specific approach allows the regularity of datapaths to be exploited to a greater extent than is possible with general-purpose FPGAs. The targeted systems may contain several datapaths of various widths and have irregularities in some bit-slices.

The datapath-style layout used in full custom designs has been directly applied to the organization of the DP-FPGA [27]. Figure 3.1 illustrates the DP-FPGA floorplan. Circuits are divided into their three basic constituents: datapath, control and memory. Each is implemented in its respective section of the DP-FPGA where the resources are different and are optimized for their specific functions. This work aims to show that use of function-specific resources is feasible due to the dominance of the datapath in the target circuits and the significant density improvements that can be achieved by optimizing an FPGA for datapaths. The control and memory sections are discussed briefly below, while the remainder of the thesis deals with the architecture of the datapath.

The control section is used to implement the logic required to generate control signals, as well as any other logic that does not fit well in the datapath portion. A complete analysis of the control section is not given for two main reasons. First, general-purpose FPGAs handle control logic well and only marginal gains are expected through function specific optimization. Second, the target circuits contain only small amounts of control logic by definition, meaning slight improvements in the control portion will have minimal effect on the final circuit area. A suitable control architecture may be similar to the high density FPGAs currently available, although streamlined to remove the unnecessary hooks for datapath and memory functions. Some optimizations might be made to facilitate the construction of finite

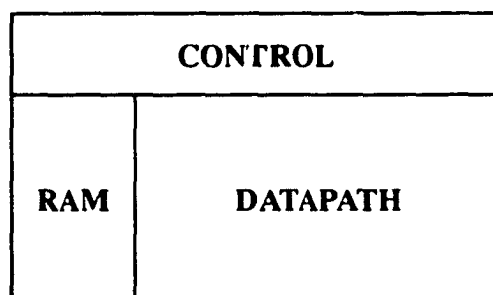


Figure 3.1 DP-FPGA Organization.

state machines, which often require complex functions with wide fanin and feedback from state registers.

Many datapath oriented designs require large blocks of memory which would not be efficiently implemented using flip-flops. There are many advantages of including this memory on-chip, providing a high level of system integration (section 2.2). The memory section also yields great potential for optimization as the density of memory in general-purpose FPGAs is poor. The specification of a high density RAM block is beyond the scope of this work and is left for future investigation. Although it could be implemented off-chip as is done with general-purpose FPGAs, it is included in the DP-FPGA specification as a necessary component for bringing highly integrated, large-scale ASICs into the realm of FPGAs. It should be clear that the memory envisioned here would be configurable blocks of high density full-custom RAM. Since the amount of area allocated for on-chip memory is fixed, there is the potential for large amounts to go unused. This problem could be overcome by producing a family of DP-FPGAs, with members having different amounts of RAM. The user could then choose a part with an amount of RAM compatible with the requirements of a given application.

3.2 Programming Bit Sharing

A primary reason why FPGAs have lower density than MPGAs is the large number of configuration bits required. This is especially true in the case of SRAM programming technology since the storage cell is larger than in other technologies such as anti-fuse, EPROM or EEPROM. The DP-FPGA uses SRAM programming technology for its benefits of being in-circuit reprogrammable and using standard CMOS process technology. The disadvantage of requiring large storage cells is reduced by sharing each programming bit across a number of identically programmed datapath slices. This organization is possible in the DP-FPGA due to the regularity of datapath logic, as illustrated in the following example.

A typical section of datapath is shown in Figure 3.2. An implementation of this section using an FPGA whose logic block contains a 3-LUT and a flip-flop with enable appears in Figure 3.3. The programming bits for the lookup tables and selected pin-to-channel routing connections are also shown, which require $17N$ SRAM cells (using a fully de-multiplexed connection block topology).

In this example, the routing has been done such that each data bus uses the same track in each horizontal routing channel. All bits of bus A are routed on track 1, while B is routed on track 2 and Z is on track 3. Further, the logic block pin connections have been chosen so that the *Select* input is connected to pin L1 of each logic block, A_i to pin L2 and B_i to pin L3. With this arrangement, the programming bits for routing connections are identical across the N bit-slices and each lookup table stores the same logic function. This motivates the idea of sharing programming bits across datapath slices.

Figure 3.4 shows the same section of datapath implemented using programming bit sharing. A single set of 17 SRAM cells is shared by all N bit-slices, reducing the total number of SRAM cells by a factor of N . The N distinct lookup tables have been replaced by a single table with N read ports. Each read port is constructed as a multiplexer tree, selecting one table value.

While programming bit sharing always reduces the number of SRAM cells by a factor of N , the number of connection elements required is different for data and control signals. Each bit of a data bus is independent and requires its own connection element controlled by a common

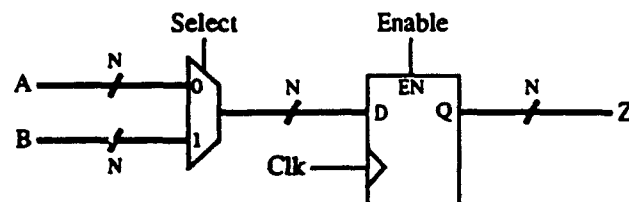


Figure 3.2 Section of Datapath.

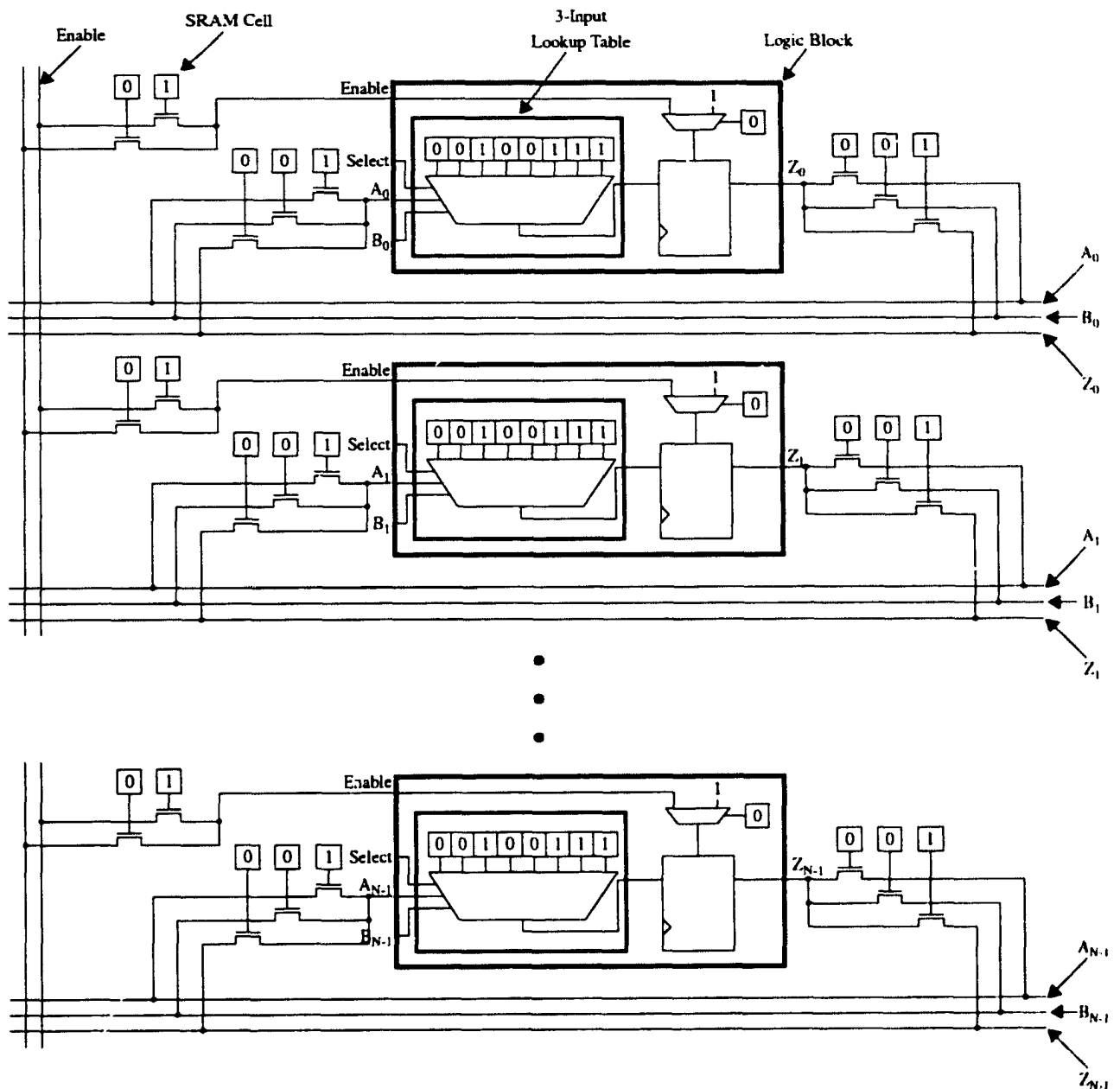


Figure 3.3 Mapping to an FPGA without programming bit sharing.

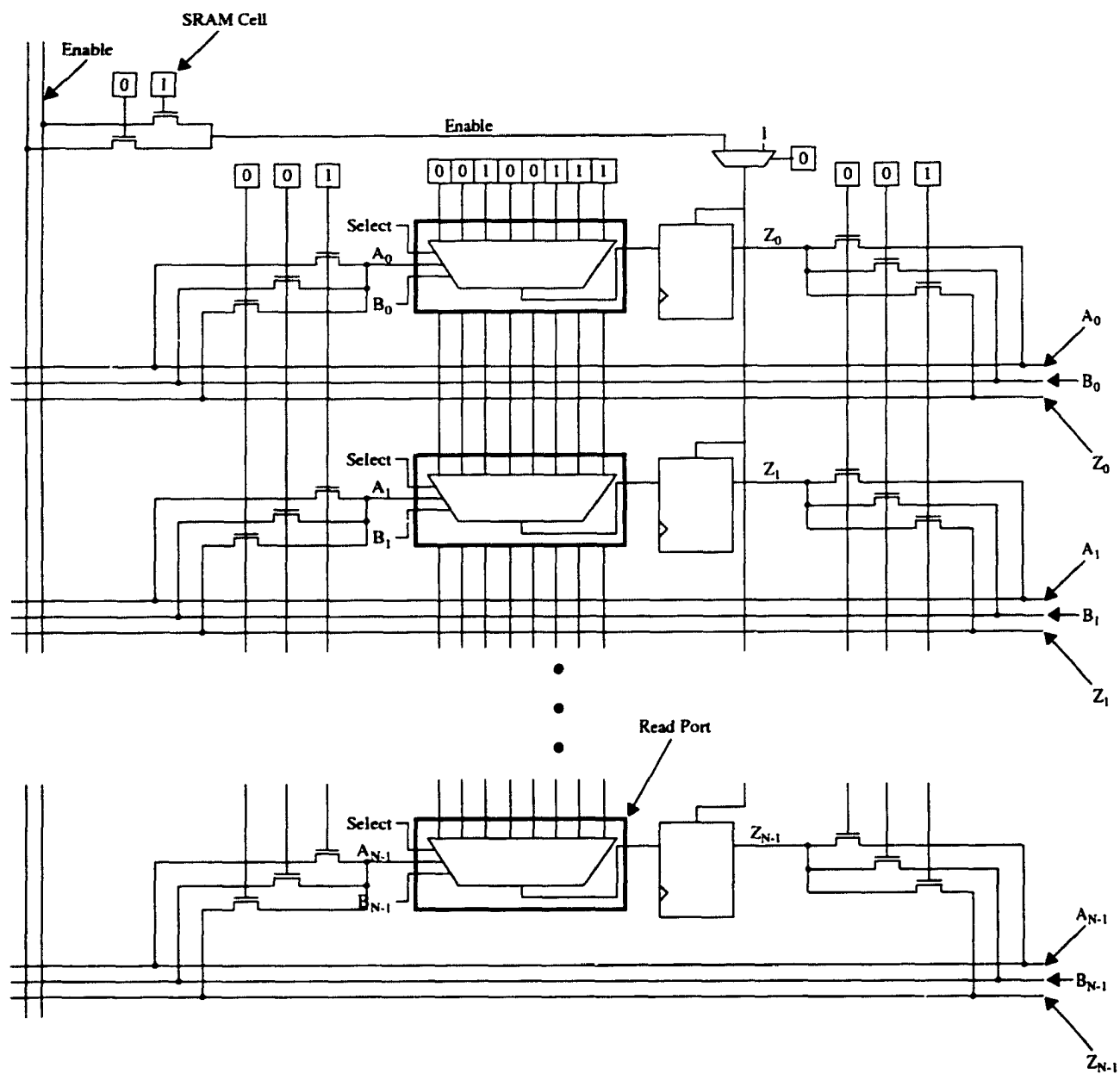


Figure 3.4 Mapping to an FPGA with programming bit sharing.

programming bit. Since an SRAM cell is several times larger than a pass transistor, a significant reduction in the total area to connect the data bus is expected. A control signal, such as the *Enable* signal in the example, is common to all bit-slices. Thus, the number of connection elements can also be reduced by N when dealing with control signals, resulting in notable area savings.

3.3 Applying Programming Bit Sharing to the DP-FPGA

The concept of sharing programming bits across many slices of a datapath has been introduced. This section will discuss some of the issues involved in its application to the DP-FPGA architecture. An analysis of the number of data bits over which a programming bit should be shared is given, resulting in the selection of this value to be used in the DP-FPGA. A discussion of the granularity of the logic blocks and routing resources follows.

3.3.1 Selecting N

In the following discussion, the variable W is defined to be the width of a datapath, while N is used to represent the number of data bits controlled by each shared programming bit. The architecture of an FPGA is fixed at fabrication and its physical resources cannot be modified by the user for each application. In particular, the value of N must be selected and cannot be changed. The examples used to introduce bit sharing assumed that each programming bit is shared across the entire width of the datapath. However, datapath widths will differ from N in many applications.

There are two area considerations which influence the selection of N . The first is the area penalty that occurs when W is not an integer multiple of N . The second is the area required per programmable connection as a function of N . These two issues will be discussed in sequence, followed by the selection of N for the DP-FPGA.

For a datapath of width W , a total of $\lceil \frac{W}{N} \rceil$ shared N -bit units are needed across the width of a connection. Normally, a datapath width can easily be extended to a multiple of N bits, which may add precision to the design. The DP-FPGA architecture contains resources to allow the proper implementation of designs where this is not possible. Examples of this situation will be given later. In either case, $W \bmod N$ bits are wasted as they were not required in the original design. For example, a 12 bit wide datapath has no wastage if $N = 4$, but wastes 4 bits if $N = 8$. Although average wastage with respect to N depends on the probability distribution of W (which is unknown), it is expected to have local minima at powers of two and a tendency to increase with N .

The second issue affecting the choice of N concerns the amount of area required by each connection. A simple model is used to illustrate some of the trade-offs involved. A fixed cost (MC) which is independent of N is assigned to the memory cell and an incremental cost (CE) is attributed to each connection element. The area per connection (A) is expressed by the following equation:

$$A(N) = \frac{MC + N \times CE}{N} = \frac{MC}{N} + CE$$

This is a decreasing function of N . However, the incremental benefit decreases rapidly as $\frac{MC}{N}$ becomes smaller than CE . The relative sizes of MC and CE depend on both the layout and the type of connection being made. For example, CE will differ depending on whether the connection element is a pass-transistor or multiplexer. A special case is the connection of control signals. As shown by the enable signal in Figure 3.4, it is the control signal itself which is shared, so only one connection element is required regardless of N . Let $MC = \alpha CE$. Based preliminary layout studies, α is estimated to be approximately 2-3. Table 3.1 shows $\frac{A(N)}{A(1)}$ for $\alpha = 2$ and $\alpha = 3$. There is little additional savings to be gained by increasing N beyond 8 bits. It should be noted that as N increases, the area required to route programming bits exceeds the area of the logic, causing this simple model to be inaccurate.

N	$\frac{A(N)}{A(1)}$ for $\alpha = 2$	$\frac{A(N)}{A(1)}$ for $\alpha = 3$
1	1.00	1.00
2	0.67	0.63
4	0.50	0.44
8	0.42	0.34
12	0.39	0.31
16	0.38	0.30
∞	0.33	0.25

Table 3.1 Area per bit for $\alpha = 2$ and $\alpha = 3$.

There is a trade-off with wastage favouring small N and the area per connection favouring large N . The optimization goal is to minimize the mean area per connection, given by $\overline{Area} = (\overline{W} + \overline{wastage}) A(N)$ over all connection types and all circuits. In this work, a value of $N = 4$ has been chosen. This value should take good advantage of bit sharing while keeping the amount of wastage low and maintaining ample flexibility for logic partitioning, placement and routing. For this initial study, using $N = 4$ is sufficient to show that bit sharing has the potential to yield significant density improvements.

3.3.2 Logic Block and Routing Granularities

The effectiveness of bit sharing depends on the circuit layout. The number of SRAM cells decreases, but the area per connection may not if the cost of routing the bit to all four programming elements is large. This suggests that the four programming elements should be placed as close to the SRAM cell as possible. This idea of locality motivates the choice of logic block and interconnect granularities in the DP-FPGA. Since 4-bit slices are forced to use identical logic, the resources required for them can be considered as single unit. For example, all of the logic blocks shown in Figure 3.4 could be combined into one large block. This allows all programming elements which are controlled by a shared SRAM cell to be contained

within a single logic block, rather than spanning several.

General-purpose FPGAs route all signals independently, eliminating the possibility of sharing programming bits among a number of signals. In the DP-FPGA, a higher view of the data routing is taken. All data signals belong to a 4-bit bus. All bits on each bus are routed identically. Only one programming bit is required to make connections for the entire bus. As mentioned previously, control signals must be treated differently than data signals. They are routed independently and are connected to logic blocks individually. Within a logic block, a control signal is common to all four bit-slices. Datapath logic blocks have two types of pin connections, control (1-bit signal) and data (4-bit bus). For the remainder of this thesis, the differentiation between data and control signals will be determined by the type of routing resources each uses.

3.4 Logic Block

The number of possible logic block choices is unlimited. The problem of finding a good block is simplified by breaking the process into two steps. First, by examining a number of target circuits, the way that data is manipulated is characterized and a parametrized set of logic blocks meeting these requirements is proposed. In the second step, an experimental evaluation of all possible values of these parameters is carried out and good values for each are determined. In this section, the basic logic block architecture is presented as part of the first phase. A description of the architectural parameters and a discussion of the experimental results is left for chapter 4.

3.4.1 Lookup Table

Digital logic can be implemented in an FPGA in many ways, including arrays of individual transistors, NAND gates, multiplexers, lookup tables and PLAs. Within each category, there are numerous variations. The scope of possible logic block configurations is infinite and must

be constrained for practical investigations. There are several issues that must be considered when narrowing the range of possible logic blocks. First, a thorough investigation must cover the complete range of possibilities within a target class. For example, a study of logic block granularity must consider the entire scope from fine to coarse grained blocks. A second consideration is the ease of enumeration of all of the blocks to be investigated, which affects the manageability of the study. Choosing blocks with random gate configurations would be unwise. Instead, it is preferable to have a set of logic blocks that can be characterized by a small number of parameters. A third important factor is the availability of CAD tools, which are used to carry out the experiments. If the proper tools are not available to evaluate a given block, then they must be developed, which is a time-consuming process. As well, if the tools cannot provide sufficiently accurate results for a given block, then comparisons with others would be unfair or invalid. The following discussion outlines the considerations made in selecting a range of logic blocks for the DP-FPGA under the guidelines just presented.

Fine-grain blocks such as individual transistors or NAND gates require a large amount of expensive interconnect which reduces their density and performance [9]. Thus, they are not considered suitable candidates for the DP-FPGA. As will be shown shortly, there are other fine-grain blocks which fit more easily into an evaluation framework.

Multiplexer-based FPGAs have been shown to be capable of providing high densities with good performance [25][31][34] and may be a good choice for the DP-FPGA. However, they are not as suitable as other possibilities for an initial investigation such as the one carried out in this work. It is difficult to enumerate a restricted set of possible multiplexer configurations that cover the entire range of granularities using a small number of parameters.

The granularity of both lookup table and PLA based logic blocks can easily be controlled using a small set of parameters [8][21][28][29]. Single-output lookup tables are the simplest of these structures. Their functionality can be scaled using a single parameter, the number of inputs to the table. Multi-output and decomposable lookup tables require additional

parameters to specify the number of outputs and shared inputs. With a PLA-based block, the number of inputs, outputs and product terms can be varied.

Single-output lookup tables fit all of the criteria outlined for the selection of a suitable set of logic blocks for this work. Their simplicity makes them an attractive choice for use in combination with the exploration of other features. They cover a sufficient range for this initial investigation into datapath FPGAs. Further, CAD tools to partition circuits into single-output lookup tables were readily available, which was not true for the more complex block structures. In the initial characterization of datapaths in this research, it was found that lookup tables provide an excellent implementation medium for datapath structures. Finally, several commercial FPGAs which use lookup tables have shown them to be capable of yielding high densities [4][5][6]. These existing devices may be used for comparison purposes.

The DP-FPGA logic block operates on 4-bit data buses. The use of programming bit sharing means that only 2^K SRAM cells are required to implement the functionality of four identically programmed K-LUTs. Note that the restriction requiring LUT input pin connections be the same in each bit-slice is necessarily met since all data bits are routed as a bus. Each bit of a bus connects to the same input pin in the respective read ports. Since all four read ports must be programmed identically, it is not apparent how irregularities in some bit-slices can be handled. This topic will be discussed in section 3.5.3. In its normal operation, the lookup table outputs a 4-bit data bus. In arithmetic mode, which is discussed in the following section, two 4-bit buses are produced as outputs.

3.4.2 Carry Chain

As part of the initial investigation of datapath intensive circuits, eight benchmark circuits (see section 4.2.1) were mapped into the Xilinx XC4000 series FPGA. The dedicated carry circuitry was used in an average of 36% of the CLBs. This high utilization implies that the resources available to support arithmetic logic will have a significant impact on both density and performance. For this reason, dedicated carry logic is included in each block. This section

discusses how such circuitry can be efficiently incorporated into the DP-FPGA architecture to yield performance benefits over currently available general-purpose FPGAs.

Commercial general-purpose FPGAs with support for arithmetic logic typically contain dedicated ripple carry logic and dedicated wire segments to route carries between logic blocks. This can yield substantial improvements over the use of the normal logic and routing resources. However, for wide datapaths, the ripple carry propagation can limit system speeds. Fast carry techniques can yield a significant improvement. A 4-bit block lookahead scheme fits well with the 4-bit wide nature of the DP-FPGA logic block. With dedicated routing to propagate carries between 4-bit blocks to create larger chains, faster arithmetic structures can be created.

In a ripple carry scheme, the direction in which the carry travels is not fixed in the carry generation function. A ripple carry chain can be easily programmed to travel up or down by appropriately configuring the inter-block carry routing. The larger granularity of the carry function in a 4-bit block lookahead scheme fixes the direction of the carry.

Fixing the orientation of the carry chain has several consequences. First, the bit order of the data buses would also be fixed. It is sometimes desirable to have the control logic near either the most significant or least significant bits to minimize the length of a control wire. The loss of flexibility of fixing the data bus orientation may increase routing delays and require additional routing resources. Second, there are situations where it is desirable to have the carry travel from the most significant bit towards the least significant bit, as is the case with priority encoders. A fixed direction carry chain would not be capable of handling such functions. Instead, the general-purpose resources would have to be used, reducing performance. A third consideration results from the fixed placement of blocks in an FPGA. If the datapath is too wide and will not fit in one column of logic blocks, it must be folded to use more than one column. In this case, the carry travels down one column and up the next. This requires that the carry chain go in both directions.

The restrictions caused by having a fixed direction carry chain would reduce the flexibility of the device. The DP-FPGA avoids these difficulties by including two chains per block, travelling in opposite directions. Since the Propagate and Generate functions of a block lookahead scheme are independent of the direction of the carry, they are common to both chains. One programming bit is used to control multiplexers which select which chain to use. An example of such a dual carry chain circuit is given in Figure 3.5, where the carry-in to carry-out delay is reduced to that of a single pass transistor plus buffers. Since the chains are constructed using pass-gates, the cost of adding a second chain is not large relative to the size of the logic block.

The Propagate ($P_{3:0}$) and Generate ($G_{3:0}$) signals are efficiently computed by decomposing the K-input lookup table into two (K-1)-input lookup tables with shared inputs, as illustrated in Figure 3.6. This method is particularly area efficient as the full capabilities of the lookup table can be used. For example, an array multiplier cell can be implemented in a single datapath logic block if $K \geq 4$. This is not possible in a Xilinx CLB (where $K = 4$) since the set of carry functions is restricted. Two CLBs are needed (one for the partial product and one for the sum), doubling the size of the multiplier compared to the DP-FPGA implementation. Further, for some arithmetic structures, such as an incrementer, G is 0000_2 . By including this capability, the entire table can be used to compute P .

There are four possible sources for C_{in} . If the block contains the least significant bit, C_{in} may be tied low (e.g. addition) or high (e.g. subtraction) or be driven by a control signal. Otherwise, C_{in} will be driven by C_{out} of the previous block in the chain. A 4-to-1 multiplexer can be used to select among these choices, requiring two SRAM cells. However, the delay due to the connection of C_{out} to C_{in} could be reduced to that of a single pass transistor if four SRAM cells are used rather than just two, trading area for speed.

This dedicated carry implementation is expected to give the DP-FPGA excellent performance, good area efficiency and a high degree of flexibility when implementing arithmetic logic.

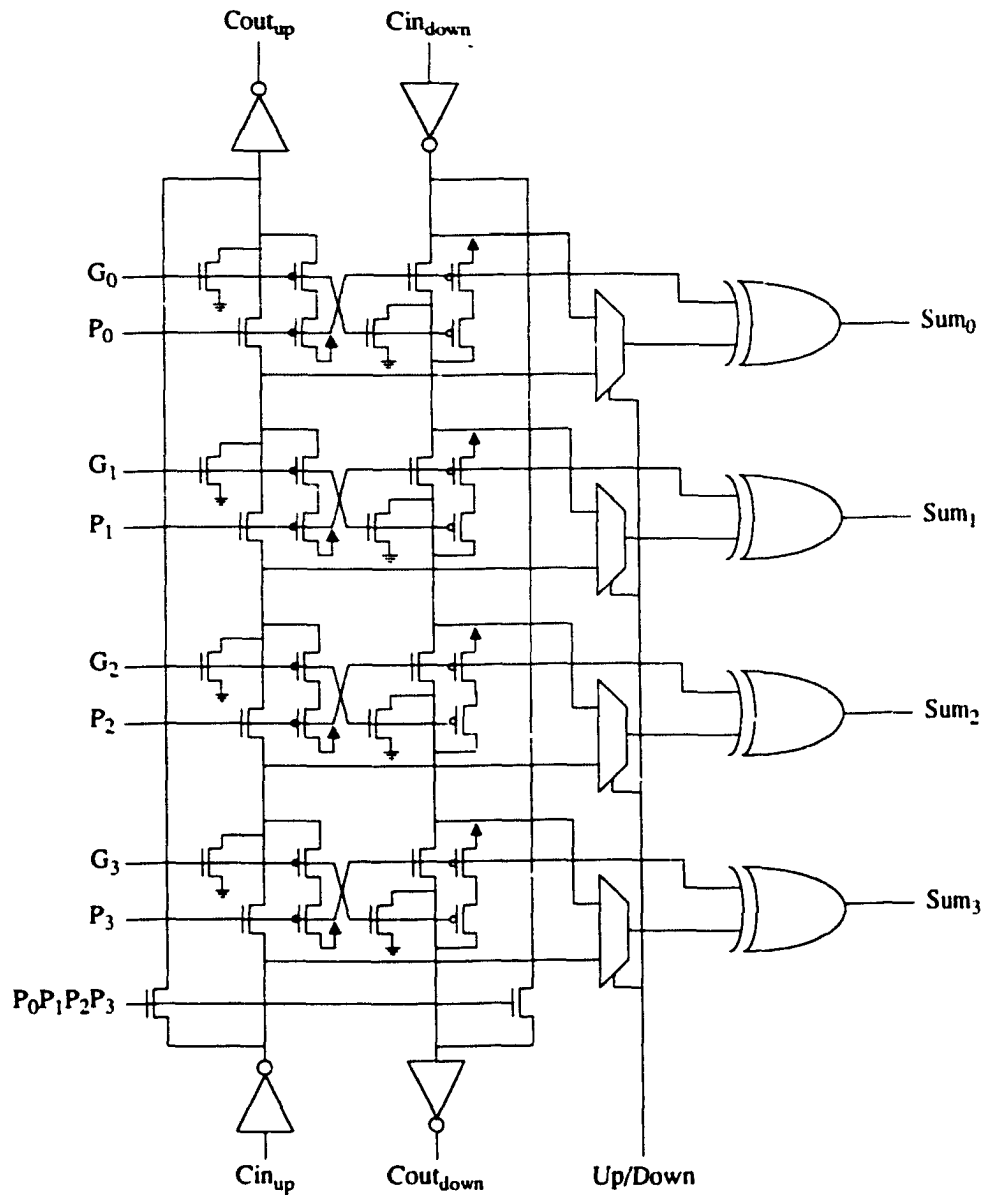


Figure 3.5 Carry Chain.

3.4.3 Flip-Flops

When benchmark circuits were implemented in the Xilinx XC4000 architecture, the flip-flops in the logic block were highly utilized. This is especially true of pipelined designs, where registers are required to buffer the data between stages. Thus, it is important that the DP-FPGA provide efficient register capabilities. Rose et al. [8] showed that it is better to include a

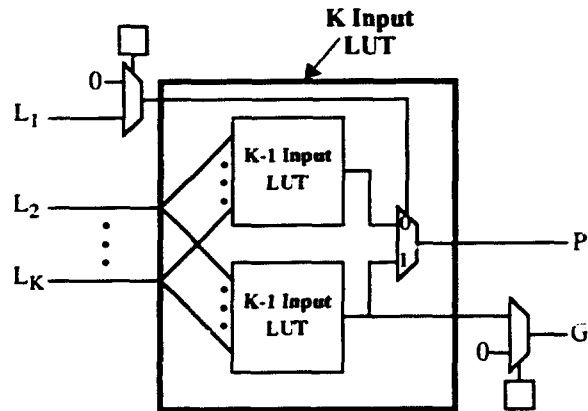


Figure 3.6 Computation of P and G.

D flip-flop when implementing circuits containing sequential logic. For these reasons, the DP-FPGA includes four D flip-flops in each logic block, with common clock and control signals. Some additional features related to the use of the flip-flops, such as the inclusion of enable signals and the ability to bypass the lookup table, are examined in Chapter 4.

3.4.4 Summary

The fundamental components of the logic block have been described. In chapter 4, an experimental evaluation of several other features will be presented. The general logic block model, which is used as a basis in the detailed analysis, is shown in Figure 3.7. It contains a 4-bit K-input lookup table, a fast carry chain and a 4-bit register. This represents the minimum set of features that will be included in the logic block. All others will be added onto this basic structure.

3.5 Routing Architecture

Routing connections in a datapath have a considerably more regular structure than is found with control logic. In contrast to conventional FPGAs, the DP-FPGA provides separate routing for data and control signals. A third resource, called the shift block, is unique to the

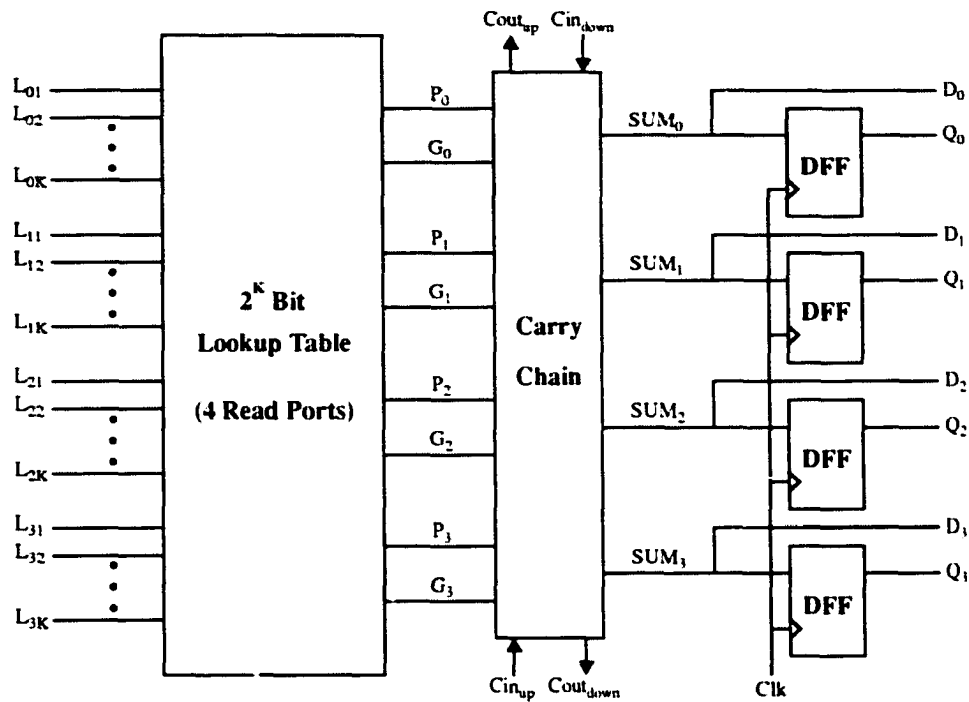


Figure 3.7 Basic Logic Block Architecture.

DP-FPGA and is included to eliminate some of the restrictions associated with the use of bit sharing. In this section, some ideas on how these three resources can be used to exploit the regularity of datapaths are discussed. A general framework for the routing architecture is given, leaving many of the fine details, which are best determined through extensive experimentation, for future work. The DP-FPGA routing architecture is illustrated in Figure 3.8.

3.5.1 Data Routing

The application of programming bit sharing leads to a higher level of data routing granularity in the DP-FPGA, where all data signals are routed as 4-bit buses. A single configuration bit controls each set of connection elements for the bus, providing denser routing resources.

A key feature of the data interconnection network is the presence of a strong horizontal bias due to the fact that nearly all data bus connections are made between blocks in the same row. In the DP-FPGA, routing a bus from one horizontal channel to the one directly above or below corresponds to a shift of four bits. Ideally, shifts of multiples of four bits would be the only need for vertical data routing. However, it may not be possible or efficient to place the entire

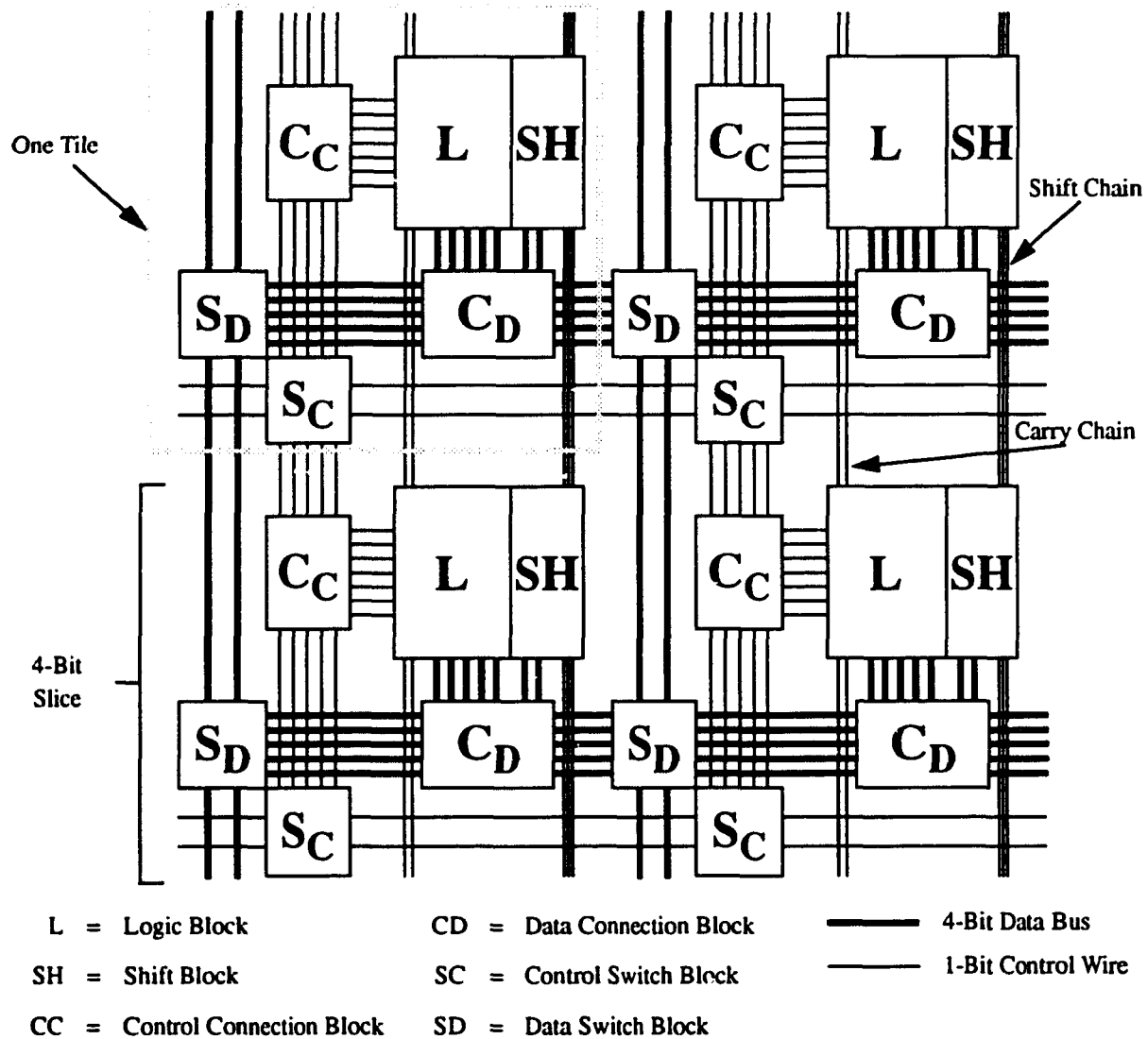


Figure 3.8 DP-FPGA General Routing Architecture.

bit-slice in the same row since the array of logic blocks is fixed. The connection of data signals to IO pins will also require vertical routing. It should be evident that placement issues will significantly affect the amount of vertical routing required.

A consequence of the horizontal bias is that logic block pins need only be connected to horizontal routing channels. As well, a high percentage of connections will require only horizontal routing. Tseng showed that reducing the number of bends in routing connections will make a less flexible channel-to-channel switch block more feasible [32]. Since the routing connections in a datapath have inherently fewer bends than in more random logic, it may be possible to use switch blocks with low flexibility to reduce the routing area. As well, paths with fewer bends will tend to pass through fewer switches, meaning that this routing architecture should provide increased performance as well as density. This can be enhanced further with a good choice of segment length distribution.

Since bit sharing is used, the connection cost per bit is reduced. This makes the idea of using a direct switch topology in the connection block more attractive than in other SRAM based FPGAs. To illustrate this point, consider the connection of a logic block pin (4 bits wide) to one of sixteen routing tracks. The resources required, in terms of number of SRAM cells and pass gates, are listed in Table 3.2 for four cases: with or without bit sharing and using direct switches or multiplexer trees. Bit sharing reduces the number of SRAM cells required by a factor of four. The use of fully multiplexed inputs also reduces the number of SRAM cells by a factor of four in this case. Without sharing, the direct method is less dense than using

Connection Description	SRAM Cells	Pass Gates
direct, no sharing	64	64
mux, no sharing	16	120
direct, sharing	16	64
mux, sharing	4	120

Table 3.2 Pin-to-channel connection resources.

multiplexers since the SRAM cells dominate the total area. However, when bit sharing is used, the difference is less pronounced. The number of pass gates per SRAM cell is much higher, mitigating the effect of reducing SRAM cells. Thus, without sharing, the cost of sacrificing area (multiplexer method) for performance (direct method) is greater than with sharing. An interesting comparison is between the case using multiplexers without sharing and the case using direct switches with bit sharing. Table 3.2 shows that the resources required are similar. However, the performance in the latter case will be superior since there is only one pass gate in the path as opposed to four. This illustrates that bit sharing can be used to gain speed without sacrificing area.

3.5.2 Control Routing

In contrast to data signals, control signals must be routed individually. Rather than explicitly sharing control routing programming bits across four bit-slices, it is each control signal which is shared. This leads to substantial area savings since only one pin is required to connect a control signal to the 4-bit slice.

In a datapath, a control signal is normally used by every bit-slice. This can be accomplished by running control signals perpendicular to the direction of data flow, as would be done in a custom VLSI implementation. This gives the control routing structure a strong vertical bias. Logic block control pins need only connect to vertical routing channels.

Ideally, control segments should span the entire datapath for improved density and performance. Since all control signals are generated above or below the datapath, the use of long lines is essential. In order to improve routability when more than one datapath is stacked vertically, shorter segments may also be needed.

The amount of horizontal control routing needed in the DP-FPGA is minimal. Input control signals are generated in the control logic located above or below the datapath. If there is sufficient horizontal resources in the control section or at the control/datapath interface, then

control signals could be routed down the appropriate column without the need for any horizontal routing. In a datapath oriented design, control signals are often on the critical path due to their high fanout. Long lines in general-purpose FPGAs are intended to minimize delay in these cases. However, the driving gate is a logic block output, just as with low fanout nets. In many cases the drive is insufficient and the signal must be buffered using logic blocks as flow-thrus. In the DP- FPGA, higher strength drivers could be placed at the control-datapath interface for these high fanout control nets. This would give the DP-FPGA a distinct performance advantage.

3.5.3 Shift Block

The use of programming bit sharing causes two major difficulties. First, since data signals are routed as 4-bit buses, it is not possible to implement shifts of non-multiples of four. Second, irregularities in bit-slices cannot be handled. To solve these problems, an additional resource, referred to as the shift block, is included in the DP-FPGA.

The shift block is illustrated in Figure 3.9. A bidirectional CMOS shifter forms the core of this unit, allowing data buses to be shifted up or down by zero to three bits. Each block can be connected to those directly above and below to perform shifts on wide buses. Each input or output is selected from one of the following four sources on an individual bit basis: a data signal, a common control signal, power or ground. While many programming bits are required, the high degree of flexibility in the choice of signal sources has several purposes. It allows the bits being shifted in to be programmed as zero, one or a control signal and sign extension to be implemented. It provides the ability to place constants on data and control tracks which is useful in handling irregularities, as explained below. Finally, since the connection is bidirectional, control signals can be inserted into data buses and conversely, individual data signals can be tapped off using control resources. The need for a block of such generality was found when implementing circuits in an architecture which uses bit sharing.

In many datapaths, the logic in some bit-slices is slightly different from the others. Such irregularities can be handled using constant data buses since an independent value can be placed in each bit position. Consider an example where three 4-bit input data buses $a[3:0]$, $b[3:0]$ and $c[3:0]$, are used to generate a 4-bit output data bus $z[3:0]$, according to the following equations:

$$\begin{aligned} z[3] &= F(a[3], b[3], c[3]) \\ z[i] &= G(a[i], b[i], c[i]) \quad \text{for } i = 0, 1, 2 \end{aligned}$$

A four-input lookup table can be used to implement this logic by calculating F and G for each slice and selecting one result using a multiplexer controlled by the fourth input s . Setting s

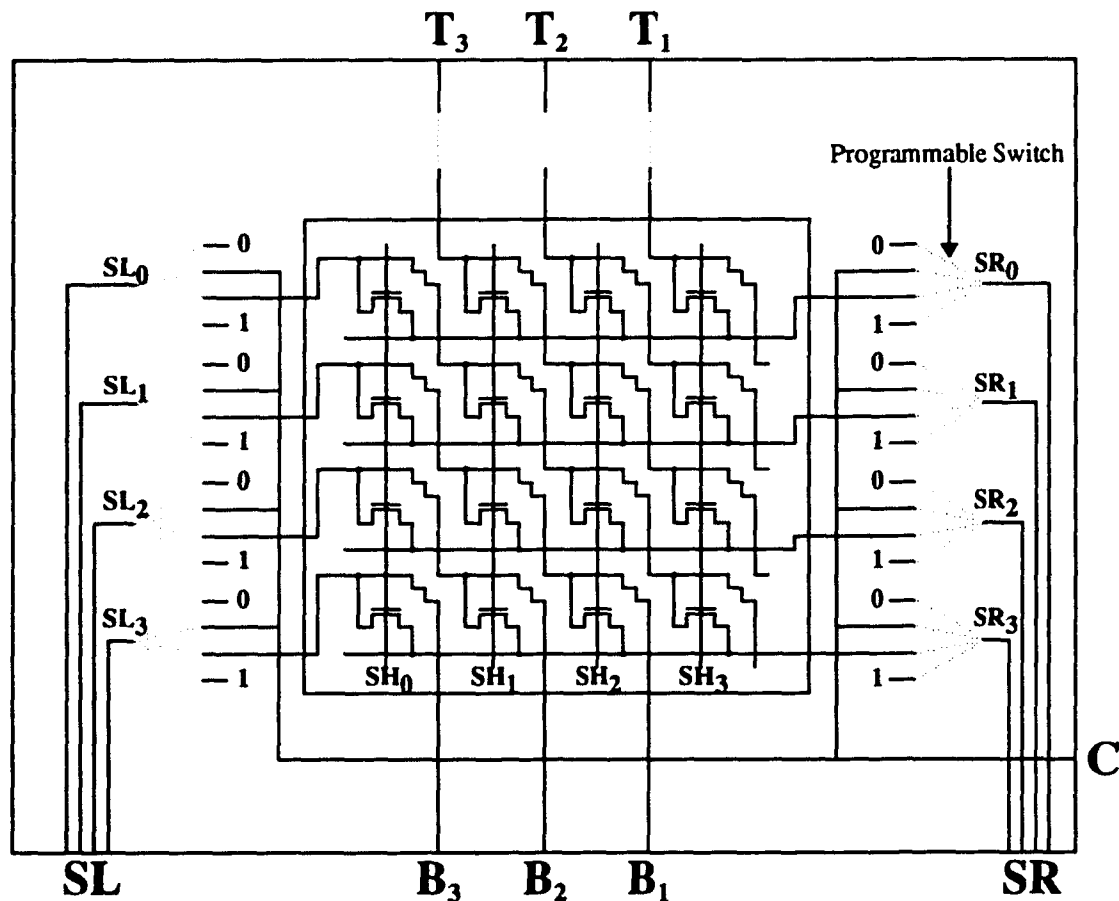


Figure 3.9 Shift Block.

equal to the constant 1000_2 , will produce the required functionality. The failure to include a mechanism to handle such irregularities would significantly restrict the range of circuits that could be implemented using the DP-FPGA.

Chapter 4

Investigation of Data Path Logic Blocks

The basic DP-FPGA architecture was introduced in Chapter 3, with examinations of the logic block and routing architectures conducted at a general level. In this chapter, a detailed experimental evaluation of logic block architectures for the DP-FPGA is presented. The optimization goal is to minimize the area required to implement circuits using the DP-FPGA.

Previous architectural studies, such as those in [8][28][29], have investigated the effects of logic block functionality on FPGA area efficiency using benchmarks containing a mixture of datapath and control logic. This study is conducted in a similar manner, but the benchmark circuits contain only datapaths. In addition to an analysis of the lookup table granularity, several other issues such as the inclusion of dedicated resources for common datapath functions and the availability of signals at logic block outputs are also examined.

This chapter first outlines the architectural features that will be investigated. A brief description of the benchmark circuits used in the analysis is given to provide an indication of the types of designs targeted by this application-specific FPGA. This is followed by an explanation of the experimental approach used in the evaluation, including a description of the software created to facilitate this process. The experimental results are presented and discussed. A comparison of the resulting architecture with three commercial architectures is given. Finally, an experimental analysis of programming bit sharing is summarized.

4.1 Logic Block Options

In order to simplify the description of the logic block, only the components for a single bit-slice are specified. This slice is replicated four times to form the complete block, with control and programming signals being common to all four slices. For further ease of description, the shared memory lookup table is described as if there are four separate tables that are programmed identically, although it is really implemented as a single table with four read ports.

The trade-off between area efficiency and logic block granularity is examined by varying the size of the lookup table as was done in [8]. In addition, the possibility of using dedicated resources to increase the density of an FPGA is investigated. A dedicated resource will improve the overall density if the reduction in the total number of blocks as a result of the added functionality outweighs the increase in tile area.

An analysis of datapaths showed that the most common functional units are adders, multiplexers and flip-flops. As outlined in chapter 3, dedicated carry support for adders and flip-flops are included in the basic logic block feature set. The benefits of including dedicated multiplexers is not clear. Although they would be highly utilized, they can also be efficiently realized using lookup tables, unlike adders and flip-flops. For this reason, the inclusion a dedicated multiplexer in addition to the lookup table is studied. The configuration chosen is shown in Figure 4.1. One data input comes from the lookup table output, while the other is a

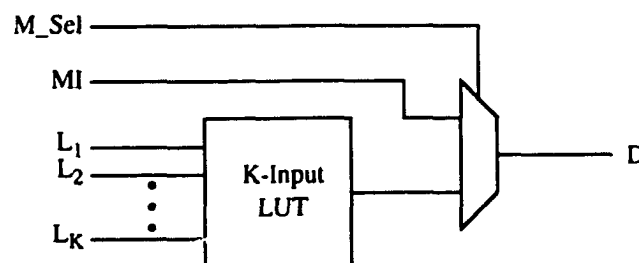


Figure 4.1 Dedicated multiplexer configuration.

direct block input, as is the selector control signal. The presence or absence of the multiplexer in a given block is specified using the binary parameter M, which is 1 if it is included and 0 if it is not.

Optional features related to the flip-flops are the use of dedicated enable signals and the inclusion of a direct data input to the flip-flop allowing the lookup table to be bypassed. These two features are used as in a Xilinx XC4000 CLB [5], with the exception that each flip-flop in the DP-FPGA has a separate direct input signal (from a data bus), whereas the direct input signal in an XC4000 CLB is common to both flip-flops. The binary parameters D and E are used to signify the presence of direct flip-flop inputs and enables in an logic block candidate.

The final optional dedicated resource to be studied is the presence of one or more 3-state buffers at the output of the block. These drivers can be used to build distributed multiplexers. In contrast to FPGAs produced by Xilinx, which have 3-state buffers outside the logic block and are restricted to selected long lines, this study places them at the outputs of logic blocks as is done in [8]. This configuration is more flexible and does not place any constraints on the routing resources to be used.

The inclusion of numerous dedicated resources in the logic block raises the question of which signals should be available as block outputs. Each output pin requires an expensive connection to the routing resources. The number of output pins can be reduced by multiplexing the possible output signals onto a smaller number of pins. However, this may reduce the utilization of the internal resources. Thus, there is a trade-off that must be considered. In this study, four signals are considered as potential outputs, as shown in Figure 4.2. The number of possible combinations is large. Sixteen output combinations are considered. These cases are enumerated using the integer parameter O and are given in Table 4.1. Multiple signals assigned to a given pin indicate that it is programmed to select one of the named signals.

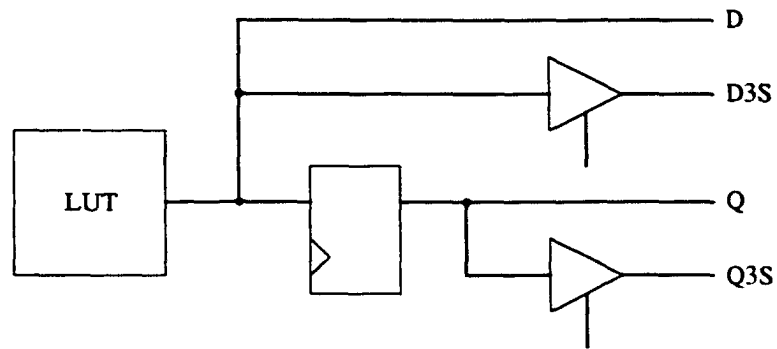


Figure 4.2 Potential output signals.

O	Pin 1	Pin 2	Pin 3	Pin 4
1	D D3S Q Q3S			
2	D D3S Q Q3S	D D3S Q Q3S		
3	D Q	D D3S	Q Q3S	
4	D	Q	D3S	Q3S
5	D Q			
6	D	Q		
7	D Q	D D3S Q Q3S		
8	D	Q	D3S Q3S	
9	D Q3S	D3S Q		
10	D	D D3S Q Q3S		
11	D Q	D3S Q3S		
12	D D3S	Q Q3S		
13	D Q3S	D D3S Q Q3S		
14	D D3S	D D3S Q Q3S		
15	D	Q Q3S	D3S Q3S	
16	Q	D D3S	D3S Q3S	

Table 4.1 Output Formats.

Using their assigned parameters, specific logic blocks are easily identified. These parameters are summarized in Table 4.2. The format used to name blocks is D#E#K#M#O#, where # is the value of each parameter. For example, the block shown in Figure 4.3 is identified as D1E0K3M1O8. Since each parameter is independent, a total of 640 different logic block architectures are investigated.

Parameter	Possible Values	Description
D	0, 1	direct input to flip-flop
E	0, 1	dedicated flip-flop enable circuitry
K	2 - 6	number of lookup table inputs
M	0, 1	dedicated multiplexer
O	1 - 16	output configuration

Table 4.2 Summary of logic block options.

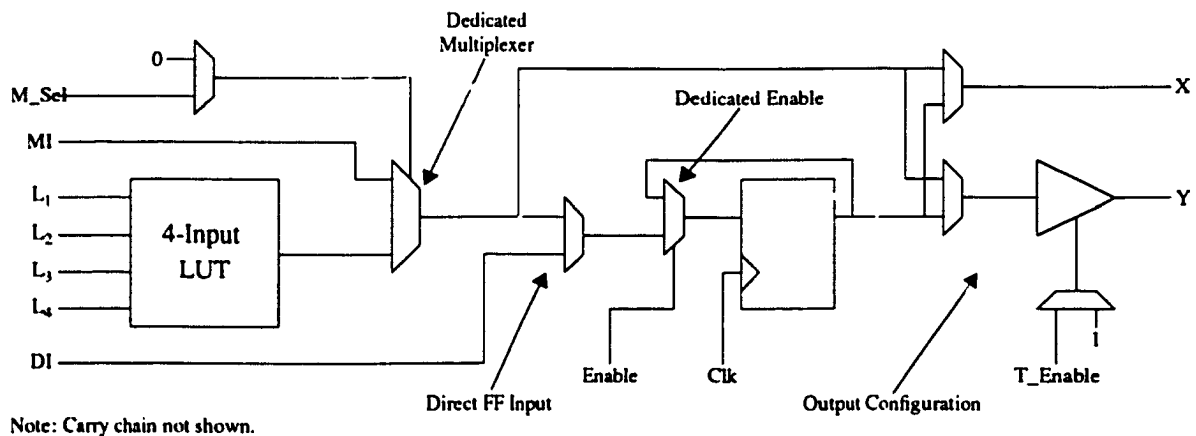


Figure 4.3 Example of a logic block candidate (identified as d1e1k4m1o7).

4.2 Experimental Procedure

This section outlines the experimental procedure used to compare logic block architectures. Descriptions of the steps taken and the algorithms used are given. Comparisons between architectures are based on the circuit area required by each to implement a set of benchmark circuits.

4.2.1 Benchmark Circuits

This section provides a brief overview of the benchmark circuits used in this empirical study. When using benchmark circuits to make architectural decisions, it is essential that they adequately reflect the designs for which the architecture is intended. Eight circuits were used in this study, all of which were developed at the University of Toronto. The benchmark circuits were taken from the following fields: digital signal processing, communications, arithmetic units and special purpose hardware. A list of the circuits used is given in Table 4.3.

Name	Description
awsim	hardware accelerator
coder	arithmetic coder/decoder
csamult	14x14 CSA multiplier
filter	digital filter
fpadder	32-bit floating pt. adder
hubnet	network controller
mult	8-bit array multiplier
viterbi	Viterbi decoder

Table 4.3 Benchmark Circuits.

An important objective in the development of the DP-FPGA was to encompass a wide range of datapath intensive circuits and not be limited to a specific application, datapath width or degree of regularity. Thus, in addition to being used in the comparison of logic block

architectures, the benchmarks were used to guide the development of the architecture as a whole and to identify the features required to support an FPGA with bit sharing. For example, the Shift Block was developed through attempts to map benchmarks into the DP-FPGA framework. The benchmark circuits reflect this objective. For example, MULT is an 8x8 array multiplier which contains only one datapath and is perfectly regular. In contrast, CODER is an arithmetic encoder/decoder for data compression and contains numerous datapath widths and irregularities.

4.2.2 Mapping Software

A collection of software utilities were developed to enable the eight benchmark circuits to be mapped into all 640 logic block candidates within reasonable time. This section outlines the mapping procedure and describes the features of the automated tools. A summary of the mapping process is given below:

For each benchmark circuit

1. Separate the datapath(s) from the rest of the circuit and extend the width(s) to a multiple of four if necessary. Add logic required to handle irregularities.

For each logic block architecture

2. Remove logic that will be implemented using dedicated resources, such as carry chains, flip-flops, flip-flop enable logic and 3-state buffers.
3. Perform technology mapping to map remaining combinational logic into K-input lookup tables using chortle-crf [30].
4. Search graph of lookup tables for multiplexer functions. Remove multiplexer logic when beneficial to do so and replace with a dedicated multiplexer or two 3-state buffers. If a multiplexer is removed (only one removed per iteration), return to step 3, else continue to step 5.
5. Pack all resources (lookup tables, carry chains, multiplexers, flip-flops, 3-state buffers) into logic blocks under the constraints imposed by the output configuration. When packing flip-flops, utilize direct inputs to minimize the number of blocks.
6. Output the total number of logic blocks required.

End for

End for

Since step 1 of the procedure is done only once for each circuit, it is done manually. It is used to separate the datapath from the rest of the circuit, extend the datapath width to a multiple of four and handle irregularities. Once this is done, the technology mapping problem becomes

equivalent to that of general purpose FPGAs which have bit-level granularity. The inner loop of the procedure, which is executed 640 times per circuit, is fully automated. The CAD tools developed are based around the Chortle lookup table mapper [30]. In step 3, an iterative approach to utilizing the dedicated multiplexers and 3-state buffers is chosen to yield excellent results while sacrificing compute-time efficiency. A pre-processor (step 2) and a post-processor (step 4) are used to separate out the logic which must be mapped and then to repack all of the resources to form complete logic blocks. The individual steps are now discussed in more detail.

Step 1: Circuit Specification

The benchmarks used in this investigation are all circuits taken from work being carried out at the University of Toronto. Each was initially specified by its designer using hand-drawn schematics, state transition diagrams and equations. No common schematic capture or high level language description was available. Therefore, these designs must be converted into a common netlist format usable by CAD software for automated mapping. Rather than using an existing netlist format, a new format called the Datapath Netlist Format (DNF) was developed. A description of this netlist format is given in Appendix A. This format is suitable for datapaths, as it makes extensive use of bus structures and allows circuits to be specified using a set of high level building blocks such as adders, multiplexers, flip-flops and shifters. This format also allows features such as adders or flip-flops, which require dedicated resources to be easily identified.

The process of creating a DNF version from the original design representation is generally straightforward, as most of the building blocks used by the designers exist as components in DNF. Several issues merit further explanation. Since only the datapath of the circuit is mapped in this study, it must be separated from the rest of the circuit. If the width of the datapath was not originally a multiple of four, it is extended. As well, if the circuit contains irregularities in

some bit-slices, additional logic is added (as explained in section 3.5.3). Fixed shifts of non-multiples of four which are normally handled through proper routing in a bit-level FPGA, must use the shift block in the DP-FPGA. These shifts are treated as a functional block in the DNF specification. Once the DNF file for each benchmark has been created, the remaining steps in the mapping process are computer automated.

Step 2: Netlist Conversion

The objective of this phase is to convert the DNF netlist into a format which can be used as input to the Chortle technology mapper. A netlist translation program called DNF2EQN was developed for this purpose. The required input to Chortle is a netlist in EQN format, which is a low level equation-based format with bit-level granularity, not suitable for the original specification. In addition to parsing the DNF file and translating the format, some netlist modifications are required.

Chortle operates only on combinational logic. Therefore, all flip-flops are removed and stored in an associated database. Inputs to flip-flops are treated as circuit outputs, while flip-flop outputs become circuit inputs. If the logic block does not contain dedicated flip-flop enable resources, the necessary logic is added. Similarly, logic which is identified as being capable of benefitting from the dedicated carry chain is removed and stored for later reinsertion.

Chortle operates on a bit-level representation as opposed to the 4-bit granularity of the DP-FPGA, requiring some means of conversion. Since all lookup tables in a four-bit slice must be programmed identically, the four-bit slice needs to be mapped in Chortle only once. For example, consider the AND of two 16-bit busses. The original DNF form would be:

and z[15:0] a[15:0] b[15:0];

It would be translated into the following EQN statements:

$$\begin{aligned} z[0] &= a[0] \& b[0]; \\ z[4] &= a[4] \& b[4]; \\ z[8] &= a[8] \& b[8]; \\ z[12] &= a[12] \& b[12]; \end{aligned}$$

Recall that the widths of all buses in the DNF files are multiples of four. As well, any irregularities present in the design were handled by adding logic at the outset. Fixed shifts of non-multiples of four, to be implemented in the shift blocks, are also removed at this stage.

Step 3: Lookup Table Mapping

In this stage, the remaining combinational logic in EQN netlist format is mapped into K-LUTs. The Chortle technology mapping program is used for this purpose. One remaining difficulty with the use of Chortle for DP-FPGA logic blocks is the presence of dedicated multiplexers and 3-state buffers, of which Chortle has no knowledge. Unlike other dedicated logic such as flip-flops or carry chains, which should always be used instead of the LUT, it is sometimes more efficient to use the LUT, as illustrated by the example below.

Consider the circuit given in Figure 4.4, to be implemented in an architecture whose logic block contains a 3-LUT and a dedicated multiplexer. The most efficient implementation constructs one multiplexer in a LUT and the other using the dedicated mux (Figure 4.5). In general, although muxes may be identified prior to LUT mapping, it is difficult to predict in

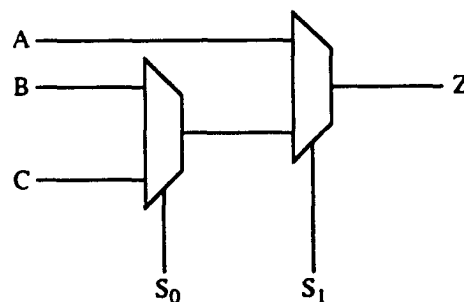


Figure 4.4 Multiplexer example circuit.

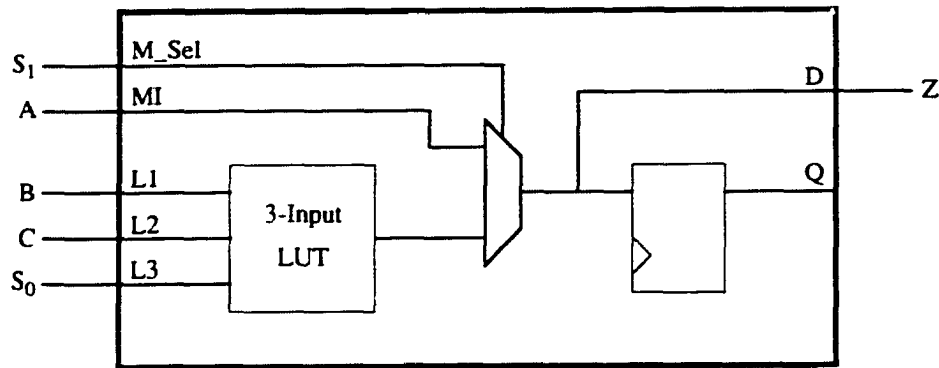


Figure 4.5 Multiplexer example implementation.

advance which ones should be implemented using dedicated logic. Thus, the technique of removing logic prior to LUT mapping (as with flip-flops and adders) cannot be used.

One solution to this problem would be to write a new LUT mapper for this type of logic block. Another would be to modify the Chortle program. Since the objective of this work is not to write a technology mapper, neither of these solutions were selected. Instead, an iterative approach was chosen, which allowed Chortle to be used without any changes. The algorithm used is described below.

```

INPUT: combinational netlist in EQN format
loop
    map into K-LUTs using Chortle
    loop
        search netlist from inputs to outputs for a mux
        if (no mux found) then
            exit
        else
            determine if beneficial to use dedicated mux
            if (beneficial) then
                remove gates comprising mux
                add mux to MuxFile
                add searched LUTs to LutFile
                generate new netlist of remaining logic
                exit
            end if
        end if
    end loop
    exit when no mux removed
end loop
OUTPUT: list of LUTs & list of dedicated muxes

```

The “search for a mux” is a pattern search for multiplexing functions (recall that EQN format uses AND, OR and NOT gates). When a multiplexer is found, it is only removed if it can legally be implemented using the dedicated mux (at least one data input must come from the output of a LUT) and if it is deemed beneficial to do so. It will be beneficial if it reduces the number of inputs to a LUT that will be remapped and therefore allows for the possibility of fewer LUTs being required when remapped. If it is not beneficial, the search continues. Otherwise, the gates comprising the mux are removed and the mux is placed in a list of muxes. The circuit must be remapped in order to attempt to reduce the total number of LUTs. Only one mux can be removed per iteration since remapping will repartition the logic into LUTs and may change the decision as to whether a given mux should be removed. Due to the Chortle algorithm, the LUTs already searched will not get packed more efficiently when remapped [33]. They are also removed from the circuit and placed in a list of LUTs. Only the unsearched logic is remapped, increasing the efficiency of the program.

This process repeats until a pass is made in which no muxes are removed. All remaining LUTs are added to the list of LUTs, completing the LUT mapping procedure.

Step 4: Logic Block Packing

At this point in the mapping procedure, the number of LUTs required is known. However, this value is not necessarily equal to the number of logic blocks required. Additional blocks may be required when an input to a flip-flop does not come from the output of a LUT or when the output configuration does not allow a logic block to be packed as tightly as the amount of logic might allow. A final packing program is used to pack the LUTs and dedicated logic into logic blocks.

The packing program first packs all logic blocks as tightly as possible. It then checks each block to determine if it contains a legal packing with respect to availability of signals as block outputs. If all required outputs cannot be connected, additional logic blocks are added. In

some cases, logic is replicated to yield efficient solutions. For example, if the logic block has only one output pin, but both signals Q and D3S (see Figure 4.2) are required as outputs, it is best to replicate any logic implemented in the lookup table so that only two logic blocks are used. Logic removed during earlier stages is re-inserted into the proper blocks, yielding a final netlist of logic blocks. The total number of logic blocks is used in comparing logic block architectures, while the netlist itself is useful for analyzing the architecture (e.g. utilization of dedicated logic) and for verifying the correctness of the mapping tools.

Summary

The primary objective of the mapping procedure is to achieve as high quality results as possible and not to bias the mapping towards any particular architecture. The quality of the mapping in this study is essentially determined by the quality of the Chortle program, which has been shown to yield good results [30]. The other main steps which influence the results are either done by hand or done in an exhaustive manner. While no claim of optimality can be made regarding the CAD tools developed, comparisons were made with hand-mapped circuits for numerous examples with excellent results. A limitation in the use of dedicated multiplexers was that no attempt was made to use them to implement other functions such as AND, OR or XOR gates. The intention here is to determine if replacing multiplexers with hardwired logic is beneficial.

4.2.3 Area Model

Once the mapping process is complete, the circuit implementation area for each benchmark in each architecture is known in terms of the number of logic blocks required. However, the areas of the logic blocks differ between architectures as will the amount of routing area required. In order to compare architectures in terms of implementation area, the area of each tile must be estimated using an area model. In this model, logic block area and routing area are considered separately.

The model of logic block area is based upon the one developed by Rose et al. in [8], but is extended to incorporate the use of programming bit sharing and to account for the various optional resources. An area cost is assigned to all of the main components of the block using a number of parameters. A nominal value (assuming a 1.2 μm CMOS process) is assigned for each parameter, representing an estimate of the cost. Each parameter can be varied to determine its effect on the total area. This is necessary since the model parameters are estimated and will depend strongly on the physical layout. Table 4.4 summarizes the parameters used. Since bit sharing is used, some components will be shared across four slices, while others will be replicated four times.

Parameter	Description	Nominal Value (μm^2)
BA	SRAM cell	500
FA	Fixed area	12000
MA	Multiplexer	150
TA	3-state driver	360

Table 4.4 Summary of logic block area parameters.

The fixed area (FA) accounts for components that are common to all architectures, such as the carry chain and flip-flops. Although the shift block is considered a routing resource, its area is modeled in the same manner as the logic block since its structure is similar.

Placement and routing steps are not performed in this work. Instead, the routing cost is estimated in the same manner as Hill in [29]. In this routing model, a cost is assigned to each logic block pin, assuming that each requires a fixed amount of interconnection area. This is a reasonable approximation since routing area correlates well with the number of pins on a block [2]. In this case, however, it is necessary to differentiate between data and control pins as their costs will not be equal. Control lines have less flexibility, consisting mostly of vertical

long lines. They will have less routing overhead associated with them. Data routing resources will have more tracks, shorter segments and more channel-to-channel switching circuitry. A reasonable value for the control pin cost is eight memory cells, whereas that for data pins is fourteen SRAM cells per bit if bit sharing is not used. These costs are doubled for pins with bit sharing to account for the larger number of transistors, raising the cost of a 4-bit data pin to twenty-eight memory cells. These routing parameters correspond to initial layout studies of the DP-FPGA [35].

4.3 Experimental Results

Using the procedure outlined in the previous section, the numbers of logic blocks required to implement each of the eight benchmark circuits in all 640 logic block architectures were determined. These values were translated into silicon area estimates using the area model. In this section, the results are presented and discussed.

The different logic block architectures are compared using normalized averages to give each circuit equal weighting. If the implementation area of circuit i using logic block x is represented as A^i_x , then the corresponding normalized area N^i_x , as defined in [8] is given by

$$N^i_x = \frac{A^i_x}{\min_{\text{all } y} \{A^i_y\}}$$

A summary of results using the nominal values for all area parameters is provided in Table 4.5. The architectures listed were chosen such that the best block for each value of each parameter, except the output configuration, is included. These results show that no block is clearly superior, with several in the range of experimental variation. Thus, rather than treating these figures as absolute, they are used as a guide to study the interrelationships between the parameters and to determine why some features might be more useful than others.

Architecture Tag	Normalized Area Averaged over all Benchmarks	Increase in Average Area over Best Block
d1e1k4m0o7	1.202	0.0%
d1e1k3m0o7	1.228	2.2%
d1e1k4m1o6	1.228	2.2%
d0e0k3m0o5	1.256	4.5%
d1e1k5m0o6	1.281	6.6%
d1e1k6m0o6	1.642	36.6%
d0e0k2m1o5	1.939	61.3%

Table 4.5 Summary of Results.

4.3.1 Flip-Flop Enables and Direct Inputs

The use of dedicated flip-flop enable logic and direct flip-flop inputs are analyzed together due to their strong inter-dependencies. The use of these two features both individually and in combination is examined. Figure 4.6 plots the area results for the architecture with the lowest average area for each of the four cases versus K. The use of enable and no direct input does not appear to be a good idea. For small values of K, the increased block cost due to the additional control pin exceeds the benefits, while for larger tables the usefulness of the extra circuitry is reduced due to the increased lookup table functionality.

Direct flip-flop inputs allow the flip-flop to be used independently of the lookup table. The output configuration must allow access to both output signals simultaneously. This feature saves a logic block each time it is used, as otherwise a lookup table would have to be used as a buffer. The main cost is an additional data input pin. It is not intuitive that this option would be frequently used as normally the flip-flop input is generated from combinational logic (lookup table). Without enable circuitry, it was used in the benchmark circuits in two situations: (i) the flip-flop input comes directly from a flip-flop output (common in pipelined designs); (ii) the output configuration does not allow all needed signals to be accessed, forcing

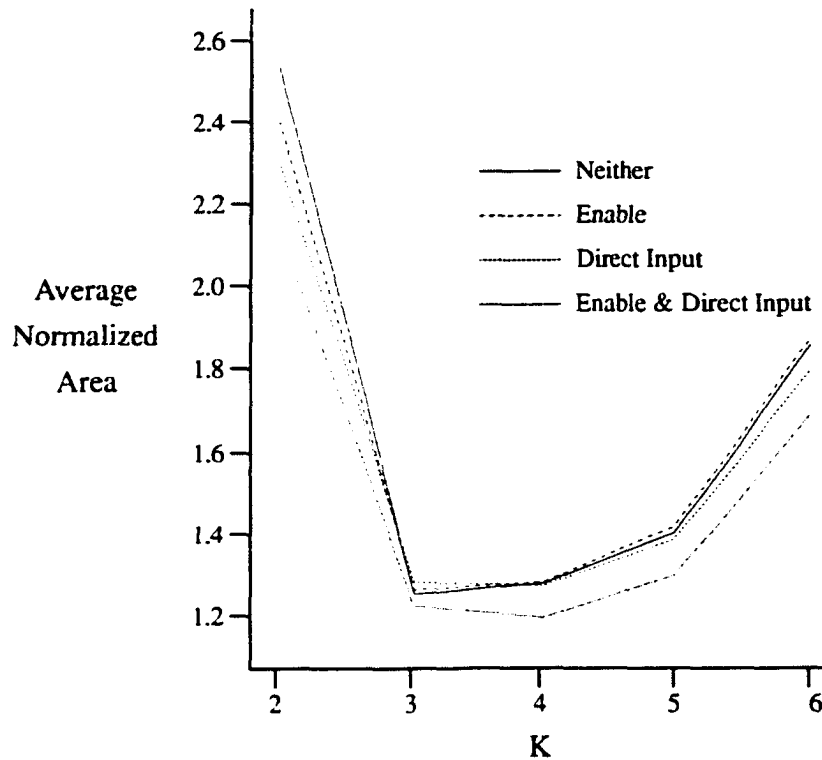


Figure 4.6 Area vs. K when using dedicated enable logic and direct flip-flop inputs.

the registered value to be generated elsewhere. Figure 4.6 shows that direct input without enable logic is not beneficial for small K (pin cost is too high relative to total block size), but is marginally useful for K greater than three.

While these two features are not beneficial individually, the combination offers an advantage. The reason for this is that the dedicated enable logic creates the following additional situations for using direct inputs: (iii) case (i) above where the flip-flops have enables (e.g. pipelines which can be stalled); (iv) registered inputs which use load signals; (v) signals fanning out to two registers with different enables. With the exception of situation (ii), the use of direct inputs is independent of K. Since the relative cost of this feature decreases with increasing K, it becomes increasingly beneficial. It is important to keep in mind that for datapaths if a feature is used in one slice, it will likely be used in all slices. Thus, each situation in which the direct input is used saves a logic block for every four bits, which becomes increasingly significant for wider datapaths.

4.3.2 Multiplexers and 3-State Buffers

Dedicated multiplexers and 3-state buffers are considered at the same time since they perform the similar multiplexing functions. Again, their usefulness is studied when they are used both individually and in combination. Figure 4.7 plots the average normalized area for the best blocks with these options against lookup table size. It shows that the dedicated multiplexer is not beneficial. The main costs associated with the dedicated multiplexer are one data pin and one control pin. Rose et al. concluded that the most area efficient blocks are those with the largest amount of functionality per connected pin [8]. When implementing datapaths, the two additional input pins associated with the inclusion of a dedicated multiplexer are too expensive to justify the amount of additional functionality they provide when used strictly as a multiplexer.

Interestingly, the results show that the inclusion of a dedicated 3-state buffer, which is used to construct distributed multiplexers, may be beneficial. Since the data input is hardwired, only

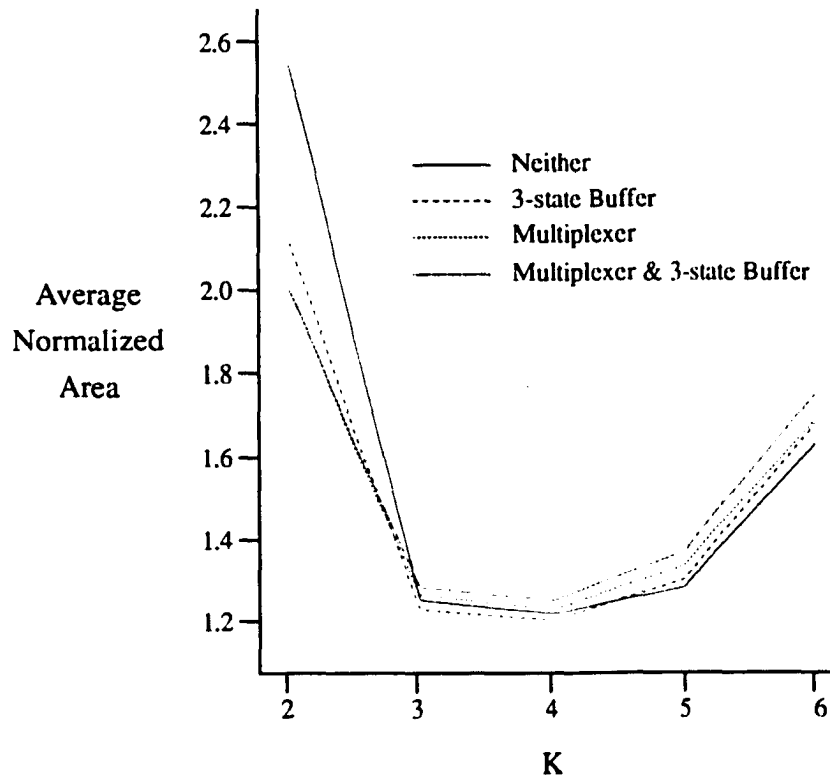


Figure 4.7 Area vs. K using dedicated multiplexers and 3-state buffers.

an additional control pin is needed. The mapping scheme uses 3-state buffers to build two-to-one multiplexers whenever it is beneficial to do so. As well, 3-state buffers can be used to build wide multiplexers efficiently (this transformation was done by hand). For example, using four 3-state buffers to build a four-to-one multiplexer instead of three three-input lookup tables will save three logic blocks. Of course, more control logic is required to generate the additional two control signals (which is not accounted for in our experiments), but this cost is expected to be small compared to the savings gained across a wide datapath.

4.3.3 Output Configuration

There are several considerations in determining the configuration of logic block output signals. A decision must be made as to which signals should be available as possible outputs. As well, the number of logic block output pins must be determined. This involves a trade-off. Having a large number of outputs improves the utilization of the block, but requires more data pins. If the number of output pins is less than the number of output candidates, a multiplexing scheme must be used.

Figure 4.8 plots average normalized area against number of output pins for some interesting blocks. In cases where there are several configurations having the same number of pins, the best one is chosen for each architecture. This plot shows that having two output pins is the best choice when the architecture contains both direct input capabilities and dedicated logic for flip-flop enables. This is because the majority of blocks require only one or two outputs. Using three outputs is a close second choice, as the extra output allows the 3-state buffers to be more heavily utilized.

Figure 4.8 shows that if the logic block does not have direct input and enable options, then only one output should be used. These results suggest that extra output pins are only needed to allow the lookup table and flip-flop to be used independently.

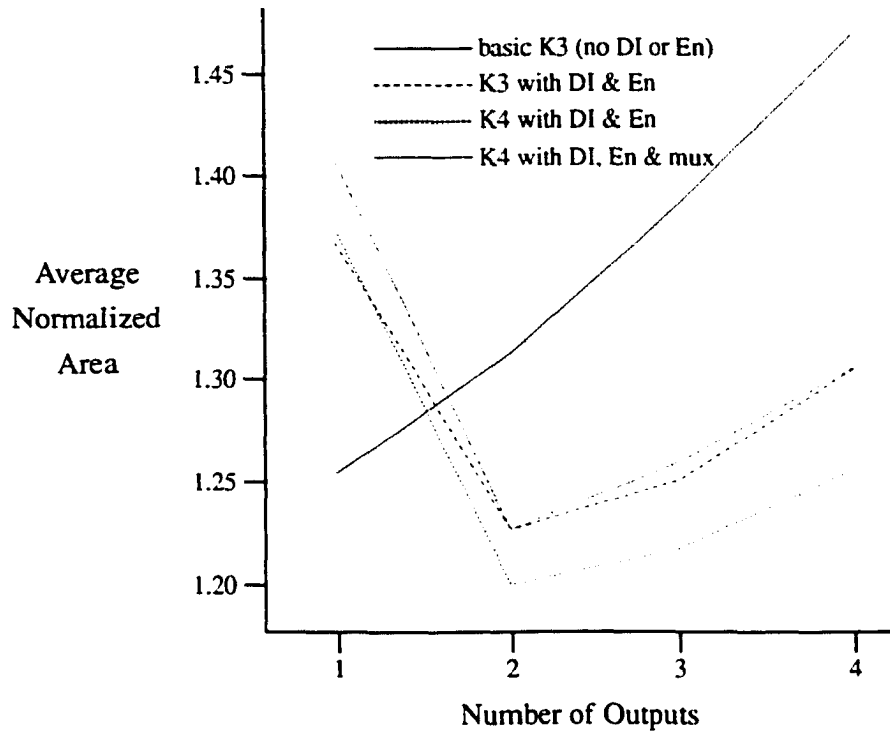


Figure 4.8 Area vs. number of output pins.

If no 3-state buffer is included in the block, then only the lookup table and flip-flop outputs are needed. If 3-state buffers are used, then multiplexing logic is needed to get at the four possible signals. Since the estimated cost of the connection interface is larger than that of the multiplexing logic, it is best to make the output configuration highly flexible. However, full flexibility would require two 3-state buffers. Since they would rarely be used simultaneously, the extra cost is not justified. A better solution is to provide the most flexibility possible with only one 3-state buffer, as shown in Figure 4.3.

4.3.4 Lookup Table Size

Most of the results have been presented as a function of K. All of the curves display the same general shape, with the minimum areas occurring at values of either three or four, with the results slightly favouring four. This shape exemplifies the trade-off involved in determining the optimum granularity of a logic block.

At low granularity ($K=2$), the blocks are highly flexible and the available functionality is well utilized. However, the amount of routing area to connect the many small blocks is too costly. As the granularity increases above $K=4$, it becomes difficult to utilize the high amount of logic per block, decreasing logic density. Lookup tables with 3 or 4 inputs provide a compromise, yielding the smallest implementation areas.

The granularity of a logic block also has a significant influence on the speed of an FPGA. Based on the findings presented in [31], a DP-FPGA using 4-LUTs is expected to have better performance than one which uses 3-LUTs. Since the results favor 4-LUTs as well, $K=4$ appears to be the better choice.

4.3.5 Area Parameter Variation

The area model makes assumptions about the area of circuit components which may not correspond exactly to an actual implementation. To study the effect of inaccuracies in the estimates, the parameter values were varied around their nominal values. It was found that reasonable sized variations do not change the results significantly. In particular, the influence of the pin cost estimates was investigated, as these parameters are the least well known. Varying the data pin cost by $\pm 50\%$ did not change the best block. As the control pin cost is raised above 25% of its nominal value, the use of the 3-state buffer (which adds a control pin) becomes inferior. The block that produced the smallest average area over a wide range of parameter variations is the one shown in Figure 4.3, except no dedicated multiplexer is included (D1E1K4M007).

4.4 Comparison with Commercial Architectures

Having described the DP-FPGA architecture and discussed the experimental results, this section presents a qualitative comparison with three existing commercial devices, namely the Altera FLEX [4], Xilinx XC4000 [5], and AT&T ORCA [6] series FPGAs. These three were chosen due to their similarities with the DP-FPGA (each uses SRAM technology and lookup

tables) and due to their added support for datapaths. All three have included a dedicated carry chain which has widely been found to be essential for datapaths. However, their ripple carry techniques cannot match the four-bit carry skip scheme of the DP-FPGA in terms of performance, which can be critical for wide datapaths.

All three commercial devices include a flip-flop per lookup table output to build data registers. The XC4000 and ORCA architectures also have direct input and enable capabilities which this work has shown experimentally to be good for datapaths. Altera provides only one block output (either D or Q of the flip-flop) which Figure 4.6 suggests is the proper number when no direct input and enable are used. Xilinx allows both D and Q to be available simultaneously, but AT&T does not (for four-bit buses). This means that in the ORCA architecture it is not possible to use the direct input to build a 4-bit data register and independently use the lookup tables for four bit-slices of datapath logic (they can be used for a control function).

While many of the features of existing FPGAs are similar to those of the DP-FPGA, there is a fundamental difference which is the amount resource sharing that is exploited. The FLEX block is one bit wide, allowing no sharing between slices. The XC4000 block can share control pins for flip-flop enables, set/resets and clocks across two slices. The ORCA block can implement four slices and utilizes the highest degree of sharing. Enables, set/resets and clocks are shared by all four flip-flops. In addition, three lookup table inputs are shared between pairs of 4-LUTs. However, this also limits the lookup table functionality for datapaths as the four 4-LUTs cannot implement functions of 2 data and 2 control inputs or 3 data and 1 control input, making them more comparable to the 3-LUT version of the DP-FPGA. Control inputs to lookup tables are shared across only two slices, as compared to four in the DP-FPGA. However, the ORCA structure is exceptional at building wide multiplexers.

The DP-FPGA not only shares control signals across four slices, but programming information as well. For example, the four 4-LUTs used in a 4-bit wide section of datapath logic require 64 SRAM cells in each of the three commercial FPGAs, whereas the DP-FPGA

requires only 16. The same ratio holds true for all logic block features requiring programming cells and most importantly in the data routing fabric, giving the DP-FPGA superior density for datapaths. Unfortunately, there is a penalty for using such a degree of sharing, which is the inability to implement control functions with reasonable cost. Distinct control logic blocks are required.

4.5 Experimental Analysis of Bit Sharing

Without having fully defined the routing architecture at this point, it is not possible to make a fair numerical comparison of the DP-FPGA architecture with presently available commercial architectures. It is desirable, however, to get an indication of the potential of such an FPGA. To gain some insight into the usefulness of bit sharing, the best block architecture determined in section 4.3 is compared with the same architecture without bit sharing (independent 1-bit blocks).

For the purpose of the following discussion, an ideal datapath is defined as one whose width is a multiple of four and which contains no irregularities. An ideal datapath mapped into logic blocks without bit sharing requires exactly four times the number of blocks. If no shifting is allowed, the 4-bit block will provide much better density since approximately one-fourth the number of SRAM cells are required. These savings are reduced for real datapaths due to the following three factors: (i) datapath widths need to be extended to multiples of four; (ii) irregularities require additional logic; (iii) shift blocks are required. The reduction in flexibility due to bit sharing in factors (i) and (ii) may require additional blocks in the bit sharing case, reducing the 1-bit to 4-bit block ratio below four. Factor (iii) is an overhead cost associated with bit sharing, regardless of regularity.

To see how these factors influence the implementation area, the benchmark circuits are mapped into the 1-bit blocks using the procedure described in section 4.2, except all bits are mapped independently and no additional logic is added for datapath extension or

irregularities. A modification of the area model is used to estimate the area of the one-bit block. This involves the straightforward scaling of the logic block area components and the removal of the shift block. The routing area modifications require an estimate of the relative cost of a 4-bit data pin to a 1-bit data pin. While the number of tracks per slice and thus the number of SRAM cells per pin will not change, each SRAM cell in the 4-bit version will control four pass transistors or multiplexers rather than one. Using estimates based on preliminary layout studies [35] the cost of the 1-bit data pin (which does not use bit sharing) is set at half that of the 4-bit pin. The cost of a control pin does not change. Table 4.6 summarizes the results.

Circuit	A # Blocks Without Sharing	B # Blocks With Sharing	$\frac{A}{B}$	Increase over Ideal	$\frac{\text{Area}(A)}{\text{Area}(B)}$
awsim	287	92	3.1	28%	2.0
coder	673	193	3.5	14%	2.2
csamult	516	162	3.2	26%	2.0
filter	396	100	4.0	1%	2.5
fpadder	395	123	3.2	24%	2.0
hubnet	199	53	3.8	6%	2.4
mult	64	16	4.0	0%	2.5
viterbi	310	82	3.8	5%	2.4

Table 4.6 The Effect of Bit Sharing on the Number of Logic Blocks.

If each circuit consisted of only ideal datapaths, the number of blocks in the 4-bit case would be $\left\lceil \frac{A}{4} \right\rceil$. The column titled "Increase over Ideal" represents the percentage increase of the actual number of 4-bit blocks over this ideal value. It is an indication of how far each circuit is from ideal regularity due to factors (i) and (ii). Table 4.6 shows that an average of 13% more blocks are required. Despite these costs and the shift block overhead, the total area is still estimated to be 2 to 2.5 times larger if bit sharing is not used. While this is only a rough

approximation since the actual values depend heavily on the physical layout, it suggests that significant improvement in density can be achieved through the use of bit sharing. The effect of bit sharing on other architectures or using other area models will depend on the cost of the SRAM cells and control circuitry relative to the total tile cost, as the area of these resources is reduced by a factor approaching four.

The preceding discussion considers only the effectiveness of bit sharing. The optimizations of the logic block and routing architectures discussed are expected to also contribute significantly to the DP-FPGA's density advantage.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis has proposed a new FPGA architecture for datapath oriented circuits which allows separate optimization of the datapath, control and memory portions of a design. The datapath architecture presented has the potential to improve FPGA densities, primarily through the use of a technique referred to as programming bit sharing. This involves sharing programming information across four identically programmed bit-slices to reduce the number of configuration storage cells by a factor approaching four. Logic block and routing architectures incorporating programming bit sharing have been developed. A new resource, called the Shift Block, has been introduced to remove the restrictions imposed by programming bit sharing, namely the inability to execute shifts of non-multiples of four bits and to implement irregularities in bit-slices.

A number of target circuits have been mapped into the DP-FPGA architecture. Estimations of implementation area show that the use of programming bit sharing can improve densities by a factor of two over an identical architecture that contains no sharing. Such significant density improvements could make FPGAs a feasible alternative to Mask-Programmed Gate Arrays for implementing large datapath intensive designs.

This work has proposed a logic block architecture which includes several novel features. This block implements identical logic across four bit-slices using a shared lookup table with four read ports. Control inputs are common to all four bit-slices. It was concluded that a dedicated carry chain and a flip-flop per bit-slice are essential components of a datapath logic block. The DP-FPGA carry chain differs from those previously developed for FPGAs, consisting of a bi-directional carry skip circuit which reduces the carry-in to carry-out propagation delay to that of a single pass-transistor plus input and output buffers. The Propagate and Generate inputs to the carry chain are both computed within the lookup table for increased area efficiency and flexibility.

The characteristics of data and control routing have been discussed and it was observed that they have differing requirements. Unlike general-purpose FPGAs which use the same resources for each, the DP-FPGA provides distinct interconnection networks for data and control signals. The data routing network features a 4-bit granularity with a strong horizontal bias and a variety of segment lengths, while the control network consists mainly of independent vertical long lines.

An investigation of lookup table based datapath logic blocks was conducted. In addition to studying the area efficiency of different sized lookup tables, which has been previously examined for mixed datapath and control applications, this work also investigated the inclusion of direct input and enable capabilities for the flip-flops, dedicated multiplexers and 3-state output drivers. It was determined that a lookup table with four inputs per bit-slice would be a good choice of lookup table granularity. It was found that allowing the flip-flops to be used independently of the lookup table through the inclusion of dedicated inputs would be beneficial provided that dedicated enable circuitry is also included. While the use of dedicated 3-state output drivers inside the logic block proved to be marginally useful for implementing distributed multiplexers, the inclusion of a multiplexer with one input hardwired to the table output was not.

5.2 Future Work

This thesis represents an initial proposal of the Datapath-FPGA architecture. Several issues have been left unresolved and require future investigation.

Programming bit sharing was studied in this work assuming that programming bits are shared across four bit-slices ($N=4$). A comparison with other values of N such as 2 or 8 is merited. This would likely involve more detailed circuit layout information than has been gathered to date.

While the datapath logic blocks were studied experimentally to define specific details, the datapath routing resources were left as a generalized network. An experimental study is necessary to more precisely recommend its composition. In particular, issues such as channel widths, segment length distribution, pin-to-channel and channel-to-channel flexibilities and the use of nearest-neighbour connections should be examined. Many area and performance benefits are expected to result from such a study, which will require the development of sophisticated CAD tools for placement and routing.

This work has focussed on improving the density of the datapath section of the DP-FPGA to show the feasibility of such an FPGA architecture. The control and memory sections, as well as the interfaces between the three sections, require investigation.

The density advantages of the DP-FPGA architecture have been presented in this thesis. Optimization for performance is another area for future work. Performance optimizations may be gained through a similar study of logic block and routing architectures with speed as the primary goal. Of particular interest will be the control and data routing segment length distribution and the use of dedicated routing resources.

Appendix A

Datapath Netlist Format

The Datapath Netlist Format (DNF) is a technology independent netlist format for describing digital circuits. It was developed specifically for use in this thesis to allow benchmark circuits to be described in a common format which could be used as input to the CAD tools. It is not intended for general use as a standard format. The DNF format allows datapaths to be specified at a functional block level and supports the use of bus structures. Hierarchy is not supported.

A DNF file consists of a sequence of statements, each describing a functional block. Electrical connections between blocks are specified as named *nets*. A net can be an individual signal or a bus. Net names consist of alpha-numeric characters or underscores. Buses are identified using indexes in brackets, similar to arrays in programming languages. Examples of individual signals are *enable* and *select*. Examples of buses are *bus_1bit[4]* and *bus_12bits[11:0]*. The character '!' immediately preceding a net name signifies inversion of the signal. For example, *!enable* means "NOT *enable*". Lines beginning with the character '#' are treated as comments. There are three constant nets '0', '1' and '-', representing logical 0, 1 or function default value (0, 1 or no connect) respectively.

Functional block statements are quite flexible. The data width of all blocks is variable. The width of a functional block is equal to the widths of the input nets, all of which must be the same, with one exception. The exception is that 1-bit inputs are also allowed and are treated as control signals (fed to all bit-slices in the block). The widths of the output nets are determined by the function of the block. For many blocks, the number of inputs may also be variable. For example, an AND gate will always have one output, but may have any number of inputs.

A brief description of the available functional blocks and their syntax is given below. Block names are shown in **bold**. These are keywords and may not be used as net names. Although the net names used in the syntax descriptions are those of individual signals for simplicity, they may be substituted for buses. The characters '...' are used to signify that an variable number of inputs are allowed. Nets in square brackets are optional.

and in0 in1 ... out ;

AND gate.

add [cin] in0 in1 out [cout] ;

Adder with optional carry-in and carry-out.

counter load count in out ;

Loadable counter.

dff [enable] in out ;

D flip-flop with optional enable.

input in0 in1 ... ;

Primary inputs to the network.

inc in out ;

Incrementer.

inv in out ;

Inverter.

lsh #bits [ShiftIn0 ShiftIn1 ShiftIn2 ShiftIn3] in out ;

Logical shift left. #bits is an integer from 0-3 determining the amount to shift. The ShiftIn inputs specify the values to be shifted in and may be a constant or a 1-bit net. Shifts of 0 bits are used to place a constant, specified by the ShiftIn inputs, on the output.

or in0 in1 ... out ;

OR gate.

output out0 out1 ... ;

Primary outputs of the network.

mux sel0 sel1 ... in0 in1 ... out ;

Multiplexer.

rsh #bits [ShiftIn0 ShiftIn1 ShiftIn2 ShiftIn3] in out ;

Logical shift right. #bits is an integer from 0-3 determining the amount to shift. The ShiftIn inputs specify the values to be shifted in and may be a constant or a net. Shifts of 0 bits are used to place a constant, specified by the ShiftIn inputs, on the output.

tbuf enable in out ;

3-state buffer.

xor in0 in1 ... out ;

XOR gate.

References

- [1] W. Carter et. al, "A User Programmable Reconfigurable Gate Array," *IEEE 1986 Custom Integrated Circuits Conference*, May 1986, pp. 233-235.
- [2] Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.
- [3] Mike Ahrens et al., "An FPGA Family Optimized for High Densities and Reduced Routing Delay," *IEEE 1990 Custom Integrated Circuits Conference*, May 1990, pp. 31.5.1-31.5.4.
- [4] *Altera FLEX 8000 Programmable Logic Device Family Data Sheet*, Altera Corporation, Aug. 1992.
- [5] Hung-Cheng Hsieh et al., "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," *IEEE 1990 Custom Integrated Circuits Conference*, May 1990, pp. 31.2.1-31.2.7.
- [6] Barry K. Britton et al., "Optimized Reconfigurable Cell Array Architecture for High-Performance Field Programmable Gate Arrays", *IEEE 1993 Custom Integrated Circuits Conference*, May 1993, pp. 7.2.1-7.2.5.
- [7] *Act-3 Field Programmable Gate Array Data Sheet*, Actel Corporation, 1994.
- [8] Jonathan Rose, Robert J. Francis, David Lewis and Paul Chow, "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid-State Circuits*, Oct. 1990, pp.1217-1225.
- [9] Jonathan Rose, Abbas El-Gamal and Alberto Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proceedings of the IEEE*, Vol. 81, No. 7, July 1993, pp. 1013-1029.
- [10] *CP20K Field Programmable Gate Arrays Data Book*, Crosspoint Solutions, Inc., 1992.
- [11] ERA 60100 Preliminary Data Sheet, Plessey Semiconductor, 1989.
- [12] H. Muroga et. al, "A Large Scale FPGA with 10K Core Cells with CMOS 0.8um 3-layered Metal Process," *IEEE 1991 Custom Integrated Circuits Conference*, May 1991, pp. 6.4.1-6.4.4.
- [13] *CAL 1024 Datasheet*, Algotronix Ltd, 1989.
- [14] *CFA6006 Field-Programmable Gate Array Data Sheet*, Concurrent Logic, 1991.

- [15] Hung-Cheng Hsieh et al., "A 9000-Gate User-Programmable Gate Array," *IEEE 1988 Custom Integrated Circuits Conference*, May 1988, pp. 15.3.1-15.3.7.
- [16] *The Programmable Logic Data Book*, Xilinx, Inc., 1994.
- [17] *EPM7000-Series Devices Data Sheet*, Altera Corporation, Oct. 1991.
- [18] *Mach Devices High Density EE Programmable Logic Data Book*, AMD, 1990.
- [19] *FPGA2040 Field-Programmable Gate Array Data Sheet*, Plus Logic, 1989.
- [20] A. El Gamal et. al, "An Architecture for Electrically Configurable Gate Arrays," *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 2, April 1989, pp. 394-398.
- [21] Kevin Chung, "Architecture and Synthesis of Field-Programmable Gate Arrays with Hard-Wired Connections," *Ph.D. Thesis*, Department of Electrical and Computer Engineering, University of Toronto, 1994.
- [22] Tony Ngai, "An SRAM-Programmable Field-Reconfigurable Memory," *M.A.Sc. Thesis*, Department of Electrical and Computer Engineering, University of Toronto, 1994.
- [23] A. Aggarwal and D. Lewis, "Routing Architecture for Hierarchical Field Programmable Gate Arrays," *FPGA '94 2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Feb. 1994.
- [24] J. Rose and S. Brown, "Flexibility of Interconnection Structures for Field-Programmable Gate Arrays," *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 3, March 1991, pp. 277-282.
- [25] Manisha Agarwala and Poras T. Balsara, "Application Specific Logic Module Architecture for FPGAs," *IEEE 1992 Custom Integrated Circuits Conference*, 1992, pp. 4.1.1-4.1.4.
- [26] D.C. Chen and J.M. Rabaey, "A Field Programmable Architecture for High Speed Digital Signal Processing Applications," *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Feb. 1992, pp. 117-122.
- [27] Neil H.E. Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design A Systems Perspective*, Addison-Wesley Publishing Company,
- [28] Jack L. Kouloheris and Abbas El Gamal, "FPGA Area versus Cell Granularity - Lookup Tables and PLA Cells," *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Feb. 1992, pp. 9-14.
- [29] Dwight Hill and Nam-Sung Woo, "The Benefits of Flexibility in Look-up Table FPGAs," *FPGAs*, W. Moore and W. Luk Eds., Abingdon 1991, edited from the *Oxford 1991 International Workshop on Field Programmable Logic and Applications*, pp. 127-136.

- [30] R.J. Francis, Jonathan Rose and Zvonko Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proceedings of the 28th ACM/IEEE Design Automation Conference*, June 1991, pp. 227-233.
- [31] Satwant Singh, Jonathan Rose, Paul Chow and David Lewis, "The Effect of Logic Block Architecture on FPGA Performance," *IEEE Journal of Solid-State Circuits*, Vol. 27 No. 3, Mar. 1992, pp. 281-287.
- [32] Benjamin Tseng, Jonathan Rose, Stephen Brown, "Using Architectural and CAD Interactions to Improve FPGA Routing Architectures," *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Feb. 1992, pp. 3-8.
- [33] R.J. Francis, Personal communication, Sept. 1992.
- [34] R. Gopisetty and S.H. Lan, "Exploration of Multiplexer-based FPGA Logic Modules," *More FPGAs*, Edited from the Oxford 1993 International Workshop on Field Programmable Logic and Applications by W.R. Moore and W. Luk, Abingdon EE&CS Books, 1994.
- [35] M. Liu, J. Mokhlesi, C. Wang and Q. Wang, ELE1832S Final Report, Department of Electrical and Computer Engineering, University of Toronto, May 1993.