DIGITAL CIRCUIT DESIGN USING FLOATING GATE TRANSISTORS

A Dissertation

by

MONTHER Y. A. ABUSULTAN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirement for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Sunil P. Khatri
Committee Members,	Peng Li
	Laszlo B. Kish
	Duncan M. Walker
Head of Department,	Miroslav M. Begovic

May 2017

Major Subject: Computer Engineering

ABSTRACT

Floating gate (flash) transistors are used exclusively for memory applications today. These applications include SD cards of various form factors, USB flash drives and SSDs. In this thesis, we explore the use of flash transistors to implement digital logic circuits. Since the threshold voltage of flash transistors can be modified at a fine granularity during programming, several advantages are obtained by our flash-based digital circuit design approach. For one, speed binning at the factory can be controlled with precision. Secondly, an IC can be re-programmed in the field, to negate effects such as aging, which has been a significant problem in recent times, particularly for mission-critical applications. Thirdly, unlike a regular MOSFET, which has one threshold voltage level, a flash transistor can have multiple threshold voltage levels. The benefit of having multiple threshold voltage levels in a flash transistor is that it allows the ability to encode more symbols in each device, unlike a regular MOSFET. This allows us to implement multi-valued logic functions natively. In this thesis, we evaluate different flash-based digital circuit design approaches and compare their performance with a traditional CMOS standard cell-based design approach. We begin by evaluating our design approach at the cell level to optimize the design's delay, power energy and physical area characteristics. The flash-based approach is demonstrated to be better than the CMOS standard cell approach, for these performance metrics. Afterwards, we present the performance of our design approach at the block level. We describe a synthesis flow to decompose a circuit block into a network of interconnected flash-based circuit cells. We also describe techniques to optimize the resulting network of

flash-based circuit cells using don't cares. Our optimization approach distinguishes itself from other optimization techniques that use don't cares, since it a) targets a flash-based design flow, b) optimizes clusters of logic nodes at once instead of one node at a time, c) attempts to reduce the number of cubes instead of reducing the number of literals in each cube and d) performs optimization on the post-technology mapped netlist which results in a direct improvement in result quality, as compared to pre-technology mapping logic optimization that is typically done in the literature. The resulting network characteristics (delay, power, energy and physical area) are presented. These results are compared with a standard cell-based realization of the same block (obtained using commercial tools) and we demonstrate significant improvements in all the design metrics. We also study flash-based FPGA designs (both static and dynamic), and present the tradeoff of delay, power dissipation and energy consumption of the various designs. Our work differs from previously proposed flash-based FPGAs, since we embed the flash transistors (which store the configuration bits) directly within the logic and interconnect fabrics. We also present a detailed description of how the programming of the configuration bits is accomplished, for all the proposed designs.

To my parents.

ACKNOWLEDGMENTS

My sincere thanks to my committee chair, Dr. Sunil P. Khatri. I thank him for his guidance, support and encouragement throughout my Ph.D. program.

I would also like to thank my committee members: Dr. Peng Li, Dr. Laszlo B. Kish, Dr. Duncan M. Walker and Dr. Gianfranco Gerosa for their valuable suggestions and feedback. Dr. Gerosa deserves special thanks for his insightful feedback and comments that helped me strengthen this work further. His industrial background bridged the gap between my research and current VLSI industry challenges and needs.

Finally and most importantly, I would like to thank my dear mother, father and siblings for their endless encouragement and support. They have always stood behind me in every step towards my success. I would also like to thank my loving wife for her support and for standing next to me during the stressful times I have been through. I thank her also for taking care of our baby, especially during the late and long hours I have spent in the office working on my research.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professors Sunil P. Khatri (advisor), Peng Li and Laszlo B. Kish of the Department of Electrical and Computer Engineering, Professor Duncan M. Walker of the Department of Computer Science and Engineering and Dr. Gian Gerosa from Intel Corporation.

All work conducted for the dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by a research assistantship from Texas A&M University.

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGMENTS v

CONTRIBUTORS AND FUNDING SOURCES vi

TABLE OF CONTENTS vii

LIST OF FIGURES xii

LIST OF TABLES xvi

CHAPTER I INTRODUCTION 1

I.1 Digital Circuit Design 1
I.2 Design Performance Metrics 3
I.2.1 Delay 3
I.2.2 Power Dissipation 5
I.2.3 Energy Consumption 6
I.2.4 Area 7
I.3 Digital Design Challenges 8
I.3.1 Process Variations 8
I.3.2 Design Regularity 8
I.4 Digital Design Approaches 9
I.4.1 Full-custom Designs 9
I.4.2 Fully Automated Designs 10
I.5 Static CMOS Versus Dynamic Logic Designs 10
I.6 Application Specific Integrated Circuits 14
I.7 Programmable Logic Arrays 16
I.8 Field Programmable Gate Arrays 18
I.9 Floating Gate (Flash) Transistors 20
I.9.1 Device Structure 20
I.9.2 Programming the Flash Transistor 21
I.9.3 Flash Transistors in Memory 22
I.9.4 Read and Write Disturb 24
I.9.5 Write Endurance 25

CHAPTER II THESIS OUTLINE 26

II.1 Implementation of Flash-based Cells 28
II.1.1 Ternary-valued, Flash-based Digital Logic Cell Implementation 28
II.1.2 Binary-valued, Flash-based Digital Logic Cell Implementation . 28
II.1.3 PLA-like, Flash-based Digital Logic Cell Implementation 29
II.1.4 Multi-valued, Flash-based Digital Logic Cell Implementation . . 29
II.2 CAD Flow 30
II.2.1 Block-level Implementation of Binary-valued, Flash-based Digital Designs 30
II.2.2 SAT-based Optimization for Flash-based Digital Designs 31
II.3 Field Programmable Gate Array (FPGA) 31
II.3.1 Flash-based Field Programmable Gate Array (FPGA) 32

CHAPTER III TERNARY-VALUED, FLASH-BASED DIGITAL LOGIC CELL IMPLEMENTATION 33

III.1 Background 34
III.2 Previous Work 35
III.3 Approach 36
III.3.1 Overview 36
III.3.2 Ternary Logic Flash-based Design Conversion 36
III.3.3 Ternary Logic Cluster (TLC) Circuit Design 39
III.3.4 Programming the Flash Ternary Logic Cluster 47
III.3.5 Logic Minimization of the TLC 48
III.4 Experiments 49
III.4.1 Simulation Environment 49
III.4.2 Flash Model Card Regression 50
III.4.3 Ternary-valued, Flash-based Implementation Details 51
III.4.4 Results and Analysis 52
III.5 Chapter Summary 54

CHAPTER IV BINARY-VALUED, FLASH-BASED DIGITAL LOGIC CELL IMPLEMENTATION 56

IV.1 Background 56
IV.2 Previous Work 58
IV.3 Approach 58
IV.3.1 Flash-based Design Conversion 59
IV.3.2 Flash Cluster Circuit Design 61
IV.3.3 Programming the Flash Cluster 68
IV.3.4 Read and Write Disturb 70

IV.3.5 Mapping a CMOS-based Design into a Flash-based Design . . . 71
IV.4 Experiments . . . 71
IV.4.1 Simulation Environment . . . 72
IV.4.2 Flash-based Implementation Details . . . 73
IV.4.3 Results and Analysis . . . 73
IV.5 Chapter Summary . . . 80

CHAPTER V BLOCK-LEVEL IMPLEMENTATION OF BINARY-VALUED, FLASH-BASED DIGITAL DESIGNS . . . 83

V.1 Background . . . 83
V.2 Previous Work . . . 85
V.3 Approach . . . 85
V.3.1 Overview . . . 85
V.3.2 Flash-based Design Conversion . . . 86
V.3.3 FC-based CAD Flow Overview . . . 87
V.4 Experiments . . . 91
V.4.1 Simulation Environment . . . 91
V.4.2 Flash-based Analysis Details . . . 92
V.4.3 Results and Analysis . . . 97
V.4.4 FC Statistics . . . 100
V.4.5 Shifting the Threshold Voltage . . . 102
V.5 Chapter Summary . . . 104

CHAPTER VI SAT-BASED OPTIMIZATION FOR FLASH-BASED DIGITAL DESIGNS . . . 106

VI.1 Background . . . 106
VI.2 Previous Work . . . 109
VI.3 Approach . . . 111
VI.3.1 Overview . . . 111
VI.3.2 Flash Cluster Circuit Design . . . 113
VI.3.3 FC-based CAD Flow . . . 115
VI.4 Experiments . . . 121
VI.4.1 Simulation Environment . . . 121
VI.4.2 Results and Analysis . . . 123
VI.5 Chapter Summary . . . 128

CHAPTER VII PLA-LIKE, FLASH-BASED DIGITAL LOGIC CELL IMPLEMENTATION . . . 130

VII.1 Background . . . 130
VII.2 Previous Work . . . 132
VII.3 Approach . . . 134

VII.3.1 Overview . . . 134
VII.3.2 PLA-like Flash Cluster (PFC) Circuit Design . . . 134
VII.3.3 Programming the PFC . . . 143
VII.4 Experiments . . . 143
VII.4.1 Simulation Environment . . . 144
VII.4.2 Flash-based Implementation Details . . . 145
VII.4.3 Results and Analysis . . . 145
VII.5 Chapter Summary . . . 149

CHAPTER VIII MULTI-VALUED, FLASH-BASED DIGITAL LOGIC CELL IMPLEMENTATION . . . 151

VIII.1 Background . . . 151
VIII.2 Previous Work . . . 152
VIII.3 Approach . . . 154
VIII.3.1 Overview . . . 154
VIII.3.2 Flash-based Design Conversion . . . 154
VIII.3.3 Multi-valued Logic Flash-based Design Conversion . . . 155
VIII.3.4 Multi-valued Flash Cluster Circuit Design . . . 157
VIII.3.5 Multi-valued Logic Array . . . 159
VIII.3.6 Multi-valued Logic Bundles . . . 162
VIII.3.7 MVLB Configuration . . . 164
VIII.3.8 Programming the Multi-valued Flash Cluster . . . 166
VIII.3.9 Logic Minimization of the MVFC . . . 167
VIII.4 Experiments . . . 168
VIII.4.1 Simulation Environment . . . 168
VIII.4.2 Results and Analysis . . . 170
VIII.5 Chapter Summary . . . 173

CHAPTER IX FLASH-BASED FIELD PROGRAMMABLE GATE ARRAY (FPGA) . . . 175

IX.1 Background . . . 175
IX.2 Previous Work . . . 181
IX.3 Approach . . . 183
IX.3.1 Overview . . . 183
IX.3.2 Conventional SRAM-based LUT Structure . . . 184
IX.3.3 Conventional Flash-based LUT Structure . . . 186
IX.3.4 Proposed Static Flash-based LUT (SF-LUT) . . . 190
IX.3.5 Proposed Dynamic Flash-based LUT (DF-LUT) . . . 194
IX.3.6 Configuration Time . . . 198
IX.3.7 Conventional SRAM-based Programmable Switch Structure . . . 199
IX.3.8 Conventional Flash-based Programmable Switch Structure . . . 199

IX.3.9 Proposed Flash-in-path (FIP) Programmable Switch 200
IX.4 Experiments 202
IX.4.1 Simulation Environment 202
IX.4.2 LUT Implementation Details 202
IX.4.3 Results and Analysis 203
IX.5 Chapter Summary 207

CHAPTER X THESIS SUMMARY AND CONCLUSIONS 208

X.1 Choosing the Right Flash-based Design Approach 209

CHAPTER XI FUTURE WORK 212

XI.1 Flash Technology Scaling 212
XI.2 3D NAND Technology 214
XI.3 Static Flash-based Implementation Using P-type Flash 215
XI.4 Preset V_T Levels to Meet Application Needs 216
XI.5 Replacing Always-on Flash Transistors with Metal Wires 217
XI.6 Customized Espresso-MV for Multi-valued Flash-based Design 219
XI.7 Delay Driven Optimization 220

REFERENCES 224

LIST OF FIGURES

FIGURE		Page
1.1	Moore's Law Trend from 1971 to 2011	1
1.2	Transition and Propagation Delays	4
1.3	Static CMOS NAND Gate	11
1.4	Dynamic Logic NAND Gate	12
1.5	Premature Discharge in Dynamic Logic	13
1.6	Domino Logic Circuit	14
1.7	ASIC Design Flow	15
1.8	Generic PLA Structure	17
1.9	Example PLA Structure That Implements the Function $G_{3,2}$	19
1.10	NMOS and Flash Device Structures	20
2.1	Thesis Outline Diagram	26
3.1	Types of Ternary Logic Nodes in Our Implementation.	37
3.2	Converting a Logic Netlist into a Flash-based Design	38
3.3	Ternary Logic Cluster (TLC)	40
3.4	i^{th} Ternary Logic Array (TLA_i) Structure.	42
3.5	k^{th} Ternary Logic Bundle in TLA_i ($\text{TLB}_{i,k}$)	43
3.6	Flash Transistor Threshold Voltages	44
3.7	Flash Output Generation Circuit	47
3.8	Layout View of a TLC (des00)	53
4.1	Converting a Logic Netlist into a Flash-based Design	59
4.2	Flash Cluster	60

4.3 Flash Logic Array i (FLA_i) Structure 62

4.4 Flash Logic Bundle i, k ($FLB_{i,k}$) 64

4.5 Flash Transistor Threshold Voltages Used in an FC 65

4.6 Flash Output Generation Circuit 68

4.7 Layout View of an FC (des00) 73

4.8 Histograms of CMOS and Flash-based Designs (Before and After Aging Compensation) 77

4.9 Delay, Power and Energy of the Flash-based Designs as V_T is Shifted. 78

4.10 Delay of Flash-based Designs at Different Process Corners and V_T Levels. 80

4.11 Power of Flash-based Designs at Different Process Corners and V_T Levels. 82

4.12 Energy of Flash-based Designs at Different Process Corners and V_T Levels. 82

5.1 Converting a Logic Netlist into a Flash-based Design 85

5.2 Layout View of an FC 99

5.3 Histograms of FCs for the b17 Benchmark 101

5.4 Delay, Power and Energy Characteristics of Flash-based Blocks as V_T is Shifted 103

6.1 Converting a Logic Netlist into a Flash-based Design 111

6.2 Cluster Cut Based (CCB) Optimization 113

6.3 Layout View of Possible FLB Structures 116

6.4 Example Miter Circuit 118

6.5 Optimization Opportunities Histogram for Benchmark ”b17” 127

6.6 Optimization Opportunities Histogram for Benchmark ”b14” 128

7.1 Example PLA Structure That Implements the Function $G_{3,2}$ 133
7.2 PLA-like Flash Cluster 135
7.3 PLA-like Flash Output Group i (PFOG$_i$) Structure 138
7.4 PLA-like Flash Logic Bundle i, k (PFLB$_{i,k}$) 140
7.5 Flash Transistor Threshold Voltages Used in a PFC 141
7.6 Flash Output Generation Circuit 142
7.7 Example Layout View of a PFC (des00) 145
7.8 Delay, Power and Energy of the Flash-based Designs as V_T is Shifted. 148
8.1 MVFC Types in Our Implementation. 157
8.2 Multi-valued Flash Cluster (MVFC) 158
8.3 i^{th} Multi-valued Logic Array (MVLA$_i$) Structure. 160
8.4 k^{th} Multi-valued Logic Bundle in MVLA$_i$ (MVLB$_{i,k}$) 161
8.5 Threshold Voltages Used in Multi-valued Flash-based Designs 164
8.6 Layout View of an MVFC Implementing Benchmark ”des00” 169
8.7 Delay, Power and Energy of the MVFC-based Designs as the V_T is Shifted. 173
9.1 SRAM-based FPGA with Off-chip PROM 177
9.2 SRAM-based FPGA with On-chip PROM 178
9.3 Flash-based FPGA 179
9.4 FPGA Logic and Interconnect Fabric Elements 184
9.5 Conventional 3-input LUT Structure 185
9.6 5T SRAM Cells 186
9.7 Flash Memory Cells (FMCs) 187
9.8 Threshold Voltages Levels Used in Flash-based Designs 188

9.9 Programming the Configuration Flash Memory Cells 190

9.10 MUX Tree Implementation in SF-LUT 192

9.11 Proposed Static Flash-based LUT Structure (SF-LUT) 194

9.12 LUT Structure Used in DF-LUT 195

9.13 Array of Flash-in-path (FIP) Switches 201

9.14 Normalized Delay, Dynamic Power, Dynamic Energy and Static Power of SF-LUT as the V_T is Shifted. 206

11.1 NAND Flash-like Pulldown Stack Used in Our Flash-based Designs . 218

11.2 Proposed NAND Flash-like Pulldown Stack 220

LIST OF TABLES

TABLE		Page
1.1	Espresso-MV Minimization Output of the Function $G_{3,2}$	18
3.1	The 'Flip' Function.	46
3.2	Programming States of Each Transistor Pair F_x and F_{x_f} for Each Literal.	46
3.3	Delay, Power, Energy and Area Ratios of Ternary-valued Logic Circuits Relative to CMOS Standard Cell Based Circuits	52
4.1	Example of Input Minterm and Cube Distribution of an m-input and n-output Logic Function	71
4.2	Delay, Power, Energy and Cell Area Ratios of Flash-based Digital Circuits Relative to Their CMOS Standard-cell Based Counterparts	74
4.3	Process Corners	81
5.1	Example of Minterm Distribution of an n-output Logic Function with m Inputs	91
5.2	Delay, Power, Energy and Cell Area Ratios of Flash-based Digital Circuits Relative to Their CMOS Standard-cell Based Counterparts	96
6.1	Delay, Power and Cell Area Ratios of Flash-based Digital Circuits (with and without Optimization) Relative to Their CMOS Counterparts ($K = 2$)	122
6.2	Delay, Power and Cell Area Ratios of Flash-based Digital Circuits (with and without Optimization) Relative to Their CMOS Counterparts ($K = 3$)	126
7.1	Espresso-MV Minimization Output of the Function $G_{3,2}$	131
7.2	Association of $PFOG_i$ to Output State and Effect on $F_{m,3}$ Output When Minimizing Against Off-sets	136
7.3	Delay, Power, Energy and Cell Area Ratios of PFC-based Digital Circuits Relative to Their CMOS Standard Cell-based Counterparts	146
8.1	Flash Transistor Configuration F_x for Each Minterm/Cube Literal.	166

8.2 Delay, Power, Energy and Area Ratios of Multi-valued (Ternary) Logic Circuits Relative to CMOS Standard Cell-based Circuits 170

9.1 SF-LUT Programming V_T's Configuration 193

9.2 Delay, Dynamic Power (P_{Dyn}), Dynamic Energy (E_{Dyn}) and Static Power (P_{Static}) Ratios of the LUTs 203

9.3 Delay, Dynamic Power (P_{Dyn}), Static Power (P_{Static}) and Total Power (P_{Total}) Ratios of the Programmable Switches 204

10.1 A Comparison Between Flash-based Design Approaches to Implement Digital Circuits 209

CHAPTER I

INTRODUCTION

I.1 Digital Circuit Design

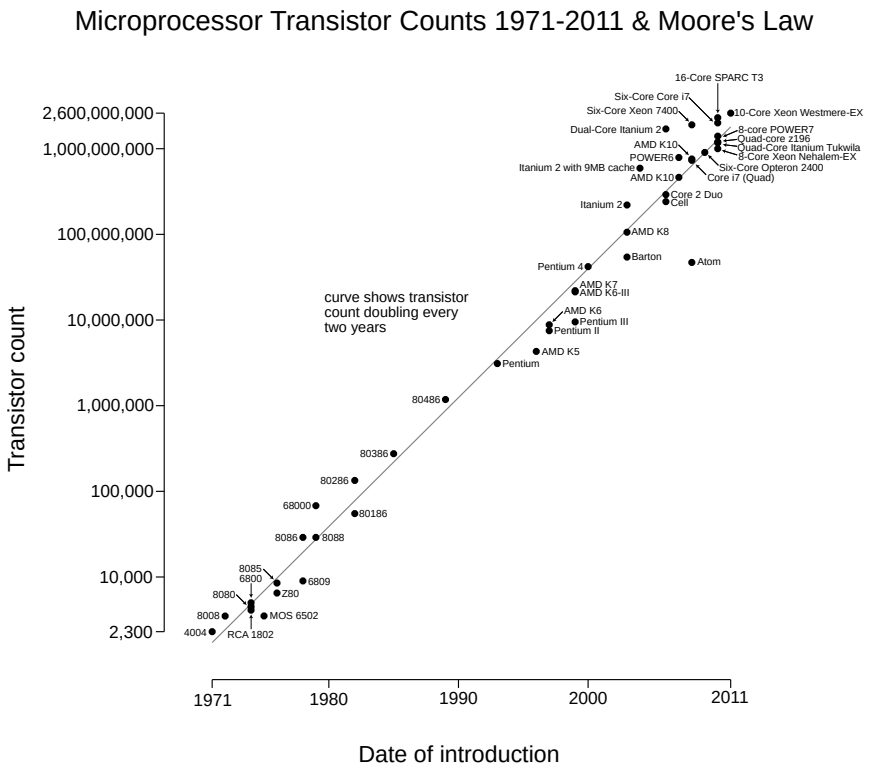


Figure 1.1: Moore's Law Trend from 1971 to 2011 (Figure Courtesy of W. G. Simon, Wikimedia Commons [1])

Complementary metal oxide semiconductor (CMOS) *very large scale integration* (VLSI) circuit design is one of the technologies that have vastly improved our daily lives. CMOS-based circuits use two types of *metal oxide semiconductor* (MOS) tran-

sistors. These are *N-channel metal oxide semiconductor* (NMOS) and *P-channel metal oxide semiconductor* (PMOS) transistors. In 1965, Gordon Moore predicted that the number of transistor in an *integrated circuit* (IC) will double approximately every 18 months. This prediction is called *Moore's law* [2]. Figure 1.1 shows the transistor count in ICs in the period from 1971 to 2011. In Figure 1.1, the x-axis shows the year of production of the IC and the y-axis shows the number of transistor inside the IC. Each dot on the figure represents a particular IC. Figure 1.1 shows how the number of transistors in the manufactured ICs between 1971 to 2011 has tracked Moore's law prediction. Moore's law not only predicted the rate of growth of *integrated circuits* (ICs), but also, and perhaps more importantly, has set expectations that the IC industry has striven to meet every 18 months. As technology advances, the demand for higher performance and lower power ICs has increased. Hence, chip manufacturers aggressively work on improving the speed, power dissipation, energy consumption and area characteristics of their ICs, while at the same time, maintaining high *yields*. In the past, the combined outcome of research in materials, device physics, interconnect and lithography has allowed the IC industry to meet expectations of Moore's Law, with each new technology node. However, as we approach material dimensions of a few atomic layers, shrinking device features becomes extremely challenging, especially given the processing variations. Hence, making the transition into a new technology node, in recent times, is a longer and expensive process. To address the longer turnaround times for adopting a new technology node, the VLSI industry has continually sought new fabrication technologies and new circuit design styles. The goals

in this quest are improved design performance metrics (delay, power, energy and area), while supporting massive manufacturing scales that are immune to process variations. To appreciate this challenge, we must first look into the important digital design metrics and study some of the tradeoffs involved in improving these metrics.

I.2 Design Performance Metrics

some of the key design metrics of an IC includes its delay, power, energy and area. Usually, the designer defines one or more of these aspects as the design target (e.g. high performance design, low power design, small design footprint, etc.). Achieving an improvement in all of these aspects is extremely rare, and in most cases, improving one of these metrics in a design results in a degradation of another metric. For example, achieving high performance comes at the expense of high power dissipation and vice versa. On the other hand, moving to a newer (smaller feature size) technology node, or a different fabrication technology, can achieve improvements in all of these metrics.

I.2.1 Delay

There are different types of delays in a digital circuit [3]. In a combination circuit, *transition delay* (or *slew rate*) is the time it takes for a signal to rise from 10% to 90% at the output of a logic gate (called *risetime* or t_r), or fall from 90% to 10% at the output of a logic gate (called *falltime* or t_f). *Propagation delay* is the time it takes a transition at the input of a logic gate to appear at the output of that logic gate (i.e. propagate through a logic gate). The rising propagation delay t_{pr} is the time it takes a transition at the input of a logic gate

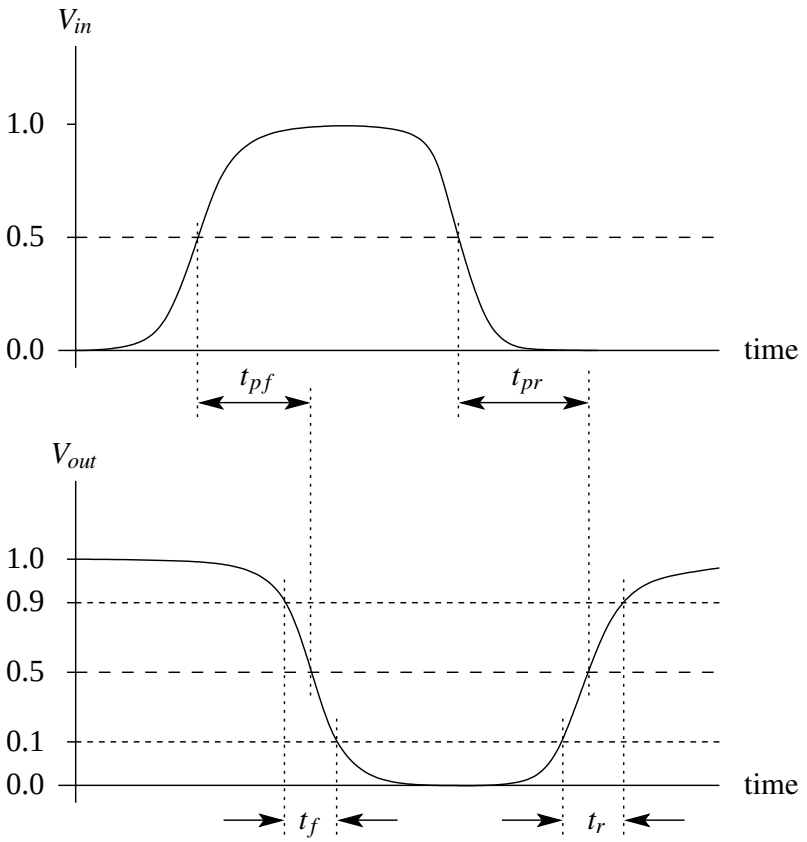


Figure 1.2: Transition and Propagation Delays

to cause a rising transition at the output of the logic gate. The falling propagation delay t_{pf} is the time it takes a transition at the input to cause a falling transition at the output of the logic gate. Figure 1.2 shows the transition and propagation delays for a logic gate. The x-axis in the figure shows the time and the y-axis shows the voltage level. In Figure 1.2, the top curve is the voltage signal waveform seen at the input of the logic gate, and the bottom curve is the voltage signal waveform seen at the output of the logic gate. The delays t_r, t_f, t_{pr} and t_{pf} are illustrated on Figure 1.2. In sequential circuits, we pay attention to other types of delays as well. *Setup time* (t_{su}) is the time the data needs to be valid at the input

of the flip-flop before the rising edge of the clock, in order to guarantee that the flip-flop captures the data and avoids meta-stability. The *hold time* (t_h) is the time that the data at the input of the flip-flop needs to stay valid after the rising edge of the clock to guarantee that the flip-flop captures the data and avoids meta-stability. Finally, the *clock-to-q* delay (t_{cq}) is the time it takes the data to propagate through the flip-flop after the rising edge of the clock.

The delay of a design is typically measured using a *static timing analyzing* (STA) tool. A good STA tool uses *dynamic programming* to recursively calculate the maximum delay of all the outputs in the design and report the maximum delay (the *critical path* delay). The critical path is the slowest path in the design, and is used to set the maximum clock frequency of the design. Delay optimization is the process of reducing the delay of the paths that violate a specified timing requirement. A last resort to reducing delay (at the expense of extra power dissipation) is to increase the supply voltage [4, 5, 6, 7].

I.2.2 Power Dissipation

There are two components to power dissipation in a digital circuit [3]. The first component is *dynamic power* ($P_{dynamic}$ or P_{dyn}). The term "dynamic" refers to the nature of the dynamic power. Dynamic power is the power dissipated while the circuit is switching. Examples of dynamic power sources are:

- Charging and discharging load capacitances in the circuit during a transition.
- Short-circuit current between VDD and GND, produced when both the NMOS and PMOS transistors in the circuit are partially ON, during a transition.

The second component of power dissipation is *static power* (P_{static}). Static power is the power dissipated while the circuit is static (no transitions are taking part). Examples of static power sources are:

- Subthreshold leakage.
- Gate leakage (through the gate dielectric).
- Junction leakage (source/drain diffusions).

The total power dissipation is the sum of both these two components.

$$P = P_{dynamic} + P_{Static} \tag{I.1}$$

There are various ways of performing power optimization on a circuit. One way is to perform *power gating*, which is shutting the power source of the inactive modules in a design (which drastically reduces leakage power [3, 8]). Reducing power is important, because it eases the thermal design of the IC.

I.2.3 Energy Consumption

As discussed earlier, design delay reduction is generally achieved on the expense of higher power. Similarly, design power reduction can be achieved on the expense of longer delay. Hence, we use the power-delay product to generally find the delay and power tradeoff for the design. Since

$$Energy = Power \times Delay \tag{I.2}$$

therefore, energy is often used as a metric to judge the quality of a design [9]. Reducing the energy is important, especially for mobile devices, since the battery size is directly

proportional to the energy requirement of a system.

The pareto-optimal realizations of any design are those that minimize Eq. I.2. The only way to further reduce the energy is to choose a different fabrication process.

I.2.4 Area

Area is an important design metric due to the limitation in the chip die area. Larger chips are more susceptible to failure due to process variations, and are therefore more expensive. Hence, area is a crucial design metric in VLSI circuit design. In early design stages, *active area* (the total area occupied by the channels of all transistors) can be used for area comparisons and area estimates. However, active area is not a representation of the physical area of the design (the actual area occupied by the design). This is because the active area ignores structural information and important physical details such as number of contacts in the design and their locations, well and substrate connections, etc. A more accurate area metric is *cell area*, which is the area occupied by the entire cell, including the transistors and the local interconnects with respect to the cell.

Some of the factors that play a role in reducing the area are:

- Regularity of the circuit structures.
- Pitch matching.
- Sharing of diffusion nodes.

Each new technology node has an approximately 30% smaller feature size compared to the previous technology node. In general, this allows the designers to pack twice the number of transistors in the new technology node [3].

I.3 Digital Design Challenges

In this section, we will discuss two important issues faced in IC manufacturing. The first issue is process variations which is the main issue affecting yield, and translates directly to profit margins [10]. The second issue we will discuss in this section is design regularity. Design regularity reduces complexity during the lithography stages of the design [11, 12, 13, 14].

I.3.1 Process Variations

The performance of different instances of digital ICs varies tremendously in practice. Designers model these variations using different ”process corners” (statistical models). The designs are simulated against all process corners to guarantee operation across different process variations. They also use worst case performance models to guarantee minimum performance targets. However, when ICs are fabricated, their performance can vary from their target (in some cases up to 70% faster than worst case [15]). The goal of fabrication plants is to obtain high yields in terms of functional ICs and not necessarily faster ICs. Therefore, to guarantee improved speeds, we must rely on the fabrication plant to enhance the process and make it more predictable [15] or use structures that reduce the effect of process variations.

I.3.2 Design Regularity

Design rules for sub-90nm technology nodes have become much more complicated [16]. This is due to the fact that new lithography techniques are needed [17]. As

we move to new technology nodes with a smaller feature size, process variations become more challenging. As we shrink dimensions, the generation of patterns on silicon has become more challenging, resulting in many manufacturing issues that can affect yield and time-to-market, which are two very important economical aspects in IC design. One of the best ways to alleviate pattern generation problems is to increase the regularity of the design. Design regularity reduces the number of unique patterns in the design, improving the reliability of the fabricated design [16]. In standard cell-based designs, the large variation of cells in the standard cell library reduces the regularity of the design. In addition to that, in order to meet certain delay constraints, cell placement tools might place cells in a way that minimizes delay and not necessarily achieve high design regularity [11, 12, 13, 14, 18, 19, 20, 21, 22, 23, 24, 25].

I.4 Digital Design Approaches

Digital circuits can be designed using different approaches that that range from full-custom, fully automated or a mixture of these design styles. The tradeoff in choosing a certain design style is quality of design versus turnaround time. Also, digital circuits can be static (e.g. standard cell-based design) or dynamic logic based (e.g. domino logic).

I.4.1 Full-custom Designs

In full-custom designs, the designer hand-crafts all the transistors and the layout polygons representing the design. This is a very expensive and time-consuming process since it involves heavy manual work. However, it produces the best results since the timing

paths are custom-tuned and design area is minimized, since the designer who has full control over the design.

I.4.2 Fully Automated Designs

In this design style, tools are used to convert *register-transfer level* (RTL) designs into layout. The process goes through synthesis, mapping, place and route. It uses the least amount of customization. However, a design can be customized by using full-custom components in the design or by hand crafting parts of the design at the final stages to improve performance. This is the most common approach to designing logic circuits and rely heavily on the use of standard cell libraries and a chain of *computer aided design* (CAD) tools.

I.5 Static CMOS Versus Dynamic Logic Designs

Digital circuits can either be *static* or *dynamic*. In static CMOS designs [26], there is always a low impedance path from the output to either VDD or GND. Figure 1.3 shows a NAND gate implemented in static CMOS. In Figure 1.3, the output is always driven to VDD (through one of the transistors M_2 and M_3, or both), or to GND (through both of the transistors M_0 and M_1). The output will never float in a static CMOS NAND gate. However in dynamic logic, the output is precharged first (in the *precharge* cycle) and then evaluated (in the *evaluate* cycle). Figure 1.4 shows a NAND gate implemented in dynamic logic. Dynamic designs use a clock (ϕ in Figure 1.4) to regulate the precharge and evaluate cycles. Since dynamic circuits precharge their output to VDD during each precharge cycle

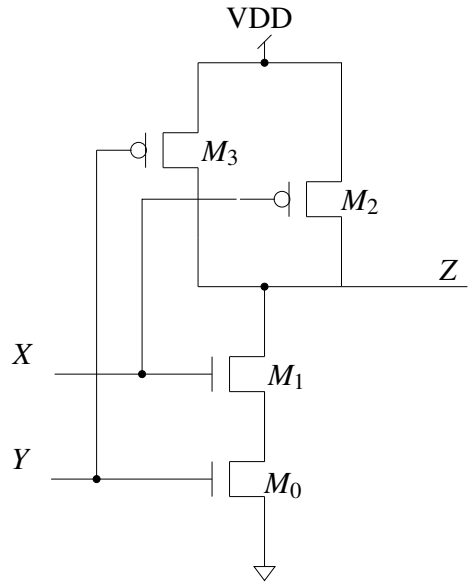


Figure 1.3: Static CMOS NAND Gate

regardless of their input state, a single PMOS transistor can be used as the pullup structure (M_3 in Figure 1.4). This M_3 is driven by the clock signal (ϕ). The pulldown structure of the dynamic circuit has all the NMOS transistors necessary to implement the logic (M_1 and M_2 in Figure 1.4), in addition to one evaluate transistor in series (M_0 in Figure 1.4). The evaluate transistor (M_0) is used to turn off the pulldown path during a precharge cycle, and hence to prevent a short circuit path between VDD and GND.

In general, dynamic circuits [26] are known to be faster, have smaller footprint (since they only implement the pulldown structure) and have higher power dissipation than static CMOS circuits [15, 27]. Therefore, dynamic circuits are sometimes used to implement speed-critical portions of the design to improve performance without significantly increasing the overall power dissipation of the design.

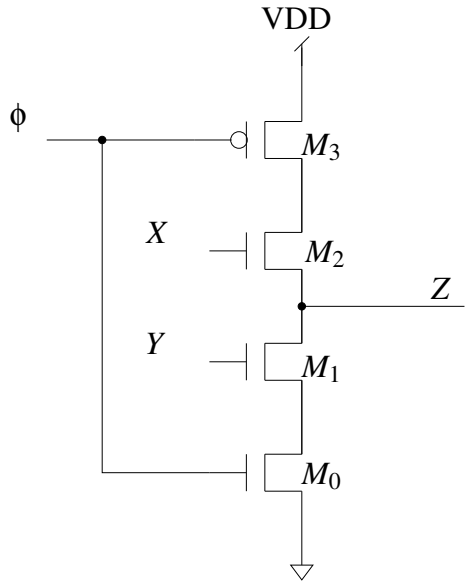


Figure 1.4: Dynamic Logic NAND Gate

Unlike static CMOS designs, which are supported by CAD tools and standard cell libraries, dynamic logic designs are usually manually designed and require careful design to avoid errors and glitches.

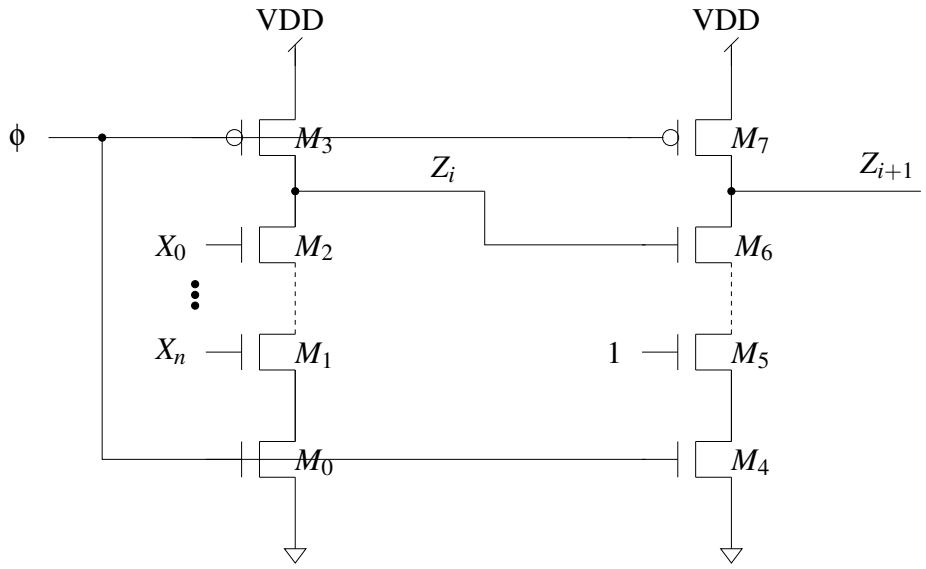


Figure 1.5: Premature Discharge in Dynamic Logic

One important issue in dynamic logic design is premature discharge. Consider Figure 1.5. In a cascaded design, when the outputs of the i^{th} level of logic in the design (Z_i) are precharged and drive the NMOS transistors of the $(i+1)^{th}$ level of logic (M_6), a race condition can occur. This can force the output of the gate at level $(i+1)$ (Z_{i+1}) to discharge before previous levels of logic have evaluated their results. This is also called *premature discharge*. One of the ways to prevent premature discharge in dynamic logic circuits is to add an inverter at the output of each stage of the dynamic logic circuit, as shown in Figure 1.6. Since the premature discharge issue arises due to the fact that outputs are precharged, adding an inverter at the output prevents turning on the NMOS transistors of the next stage of logic until the previous stage of logic have fully evaluated. This approach is called *domino logic*.

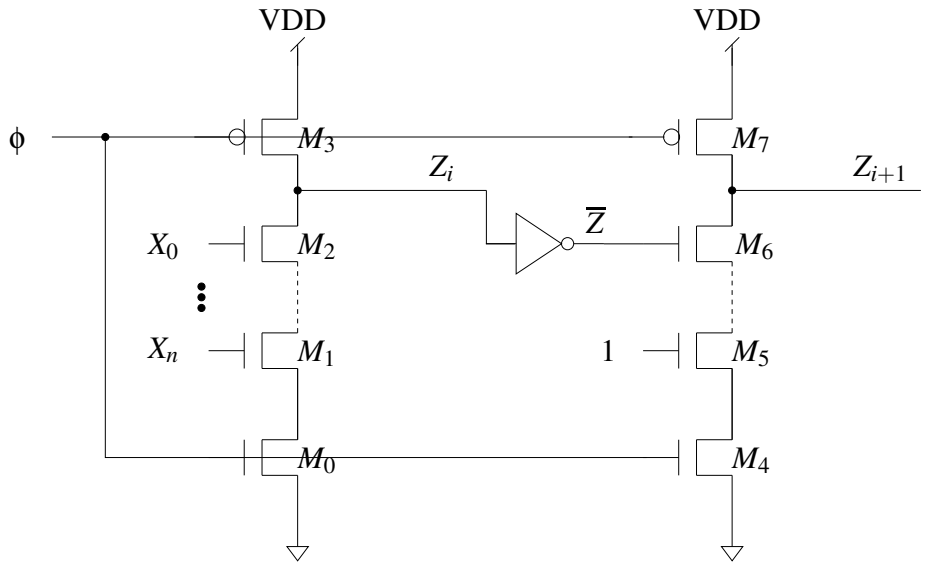


Figure 1.6: Domino Logic Circuit

I.6 Application Specific Integrated Circuits

Application Specific Integrated Circuits (ASIC) [3] are ICs designed to serve a particular purpose rather than being generic and customizable. Since the manufacture of an ASIC requires a full set of lithographic masks for fabrication, ASICs are known to have a high *non-recurring engineering* (NRE) cost. However, when produced in high volume, the high NRE cost is amortized over the number of ASICs manufactured, making the design cost-effective. Figure 1.7 shows the ASIC design flow. ASIC designs go through two major stages – a) *front-end design* and b) *back-end design*. In the front-end design stage, the design behavioral and functional specifications are generated based on the product requirements. After this, *register transfer level* (RTL) design describing the product is written, in a behavioral language. Afterwards, the front-end designers hand the behavioral RTL to

the back-end designers. During the back-end design stage, the structural specifications is generated, and then the physical synthesis is performed, which in turn generates a set of physical specifications (including chip layout, which used to fabricate the chip). At the end of the back-end stage, the chip layout is sent to the factory to fabricate the chip.

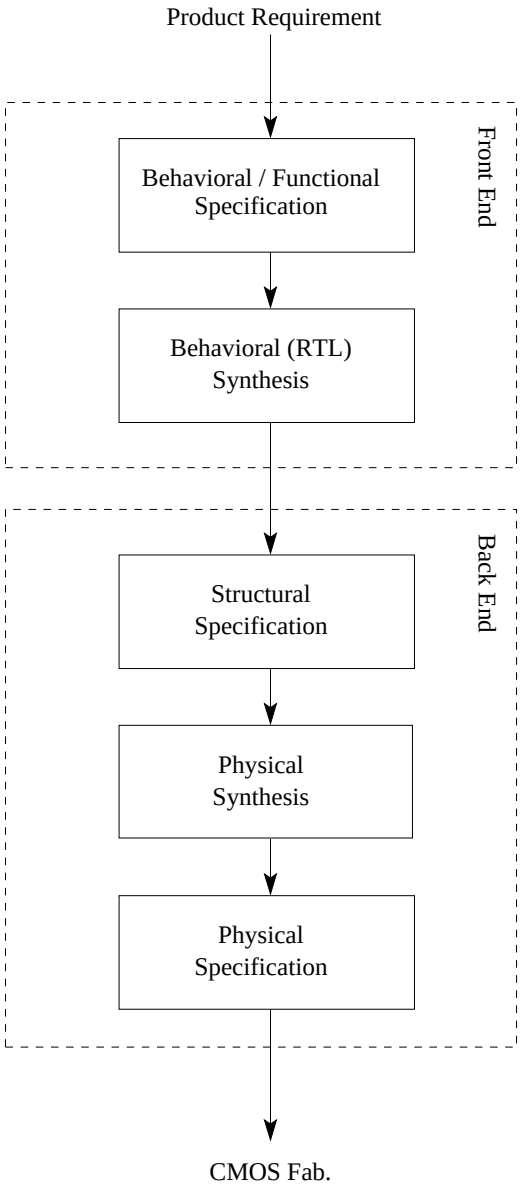


Figure 1.7: ASIC Design Flow

ASICs can be either full-custom, structured or fully automated. Full-custom ASICs are hand-crafted and optimized by hand. They also have the highest NRE cost. Structured ASICs [20, 28] use pre-defined and pre-manufactured layers of diffusion and polysilicon, leaving the metalization to be defined by the user to form the final IC design. The user maps their design to the target structured ASIC platform and the tools generate the required metal masks to be used with the structured ASIC platform. This design style reduces turnaround time and cost, since only metal layer masks need to be generated [28]. Fully automated ASICs are designed using CAD tools and have minimal customization, except for what the tools allow.

I.7 Programmable Logic Arrays

A *programmable logic array* (PLA) is a circuit structure that implements combinational circuits. PLAs implement a logic function represented in a sum of product format directly. PLAs implement any m-input, n-output logic function $F_{m,n}$. A PLA that implements the logic function $F_{m,n}$, has an AND-plane with $2 \times m$ vertical wires representing the m inputs and their complements, and an OR-plane of n wires representing the n outputs of the function $F_{m,n}$. The AND plane implements the minterms (or cubes) of the function $F_{m,n}$. Hence, the PLA can have up to 2^m AND "gates". The OR-plane performs a logical OR of the minterms (or cubes) that implement each output. Hence, the PLA has n OR "gates". PLAs natively allow cube sharing, which results in reduction of the overall size of the circuit.

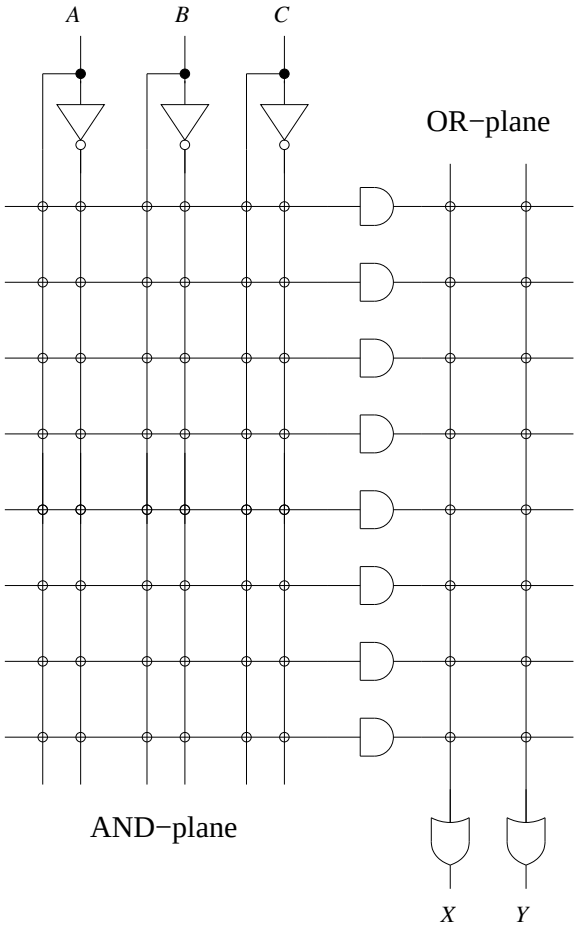


Figure 1.8: Generic PLA Structure

When implementing a function $F_{m,n}$ using a PLA, we first run Espresso-MV [29, 30] to minimize the function. Espresso-MV is a multi-valued version of Espresso [29], which is a PLA logic minimization tool that uses heuristic algorithms to minimize the number of cubes of a logic function. Table 1.1 shows an example of a function $G_{3,2}$ after being minimized using Espresso.

Figure 1.9 shows a PLA that implements the logic function $G_{3,2}$ shown in Table 1.1. Notice that the number of AND ”gates” in Figure 1.9 equals the number of input cubes

Inputs			Outputs	
A	B	C	X	Y
0	-	1	1	-
1	1	1	1	1
-	0	-	-	1
0	1	-	1	-

Table 1.1: Espresso-MV Minimization Output of the Function $G_{3,2}$

shown in Table 1.1. Also the number of OR "gates" in Figure 1.9 equals the number of outputs in Table 1.1.

I.8 Field Programmable Gate Arrays

The popularity of *field programmable gate arrays* (FPGAs) [31, 32, 33, 4, 6, 34, 35, 36] as a means to implement digital designs has been growing rapidly. The main reason for this is the effectiveness of the FPGA based design approach for low and mid volume designs. Compared to ASIC and custom designs, FPGAs exhibit a faster design turnaround time and lower NRE cost. It is conjectured that the recent reduction in the number of ASIC designs [37] (due to the increasing cost of generating IC fabrication masks), has added to the growing popularity of FPGAs.

FPGAs consist of a number of logic islands. Each island implements a small part of the entire logic programmed on the FPGA. Logic islands are made up of one or more look-up tables (LUTs). Each LUT can implement any logic function of up to n inputs (where n varies from 4 to 6 depending on the FPGA device).

However, due to their reconfigurable nature, FPGAs are usually not the best imple-

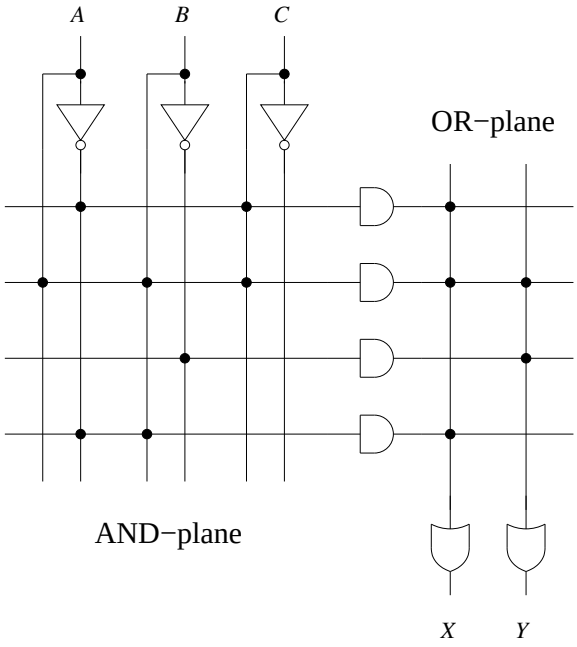
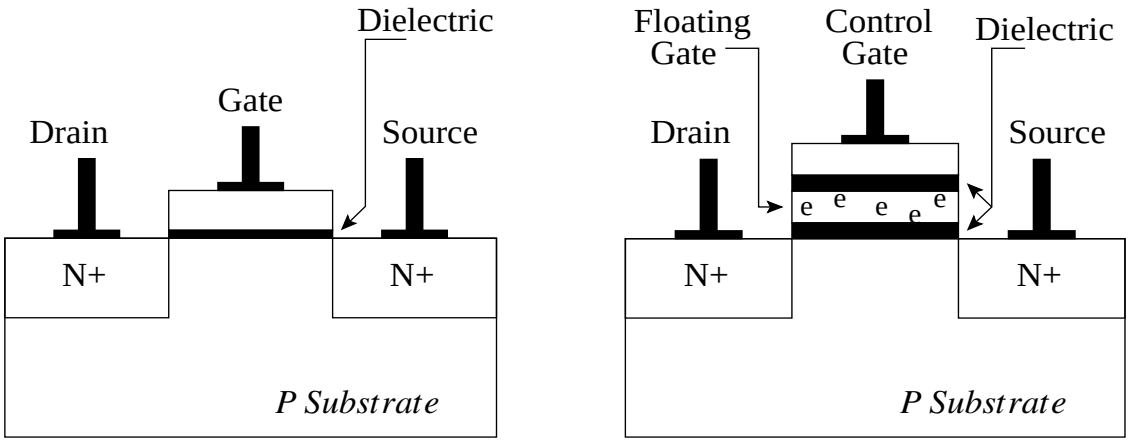


Figure 1.9: Example PLA Structure That Implements the Function $G_{3,2}$

mentation platform from a speed or energy perspective. This is because of the redundant logic and wiring that is present in the FPGA. Hence, FPGAs cannot be used in the prototyping of extreme low power designs. It was reported in [38] that FPGAs when compared to ASICs have 4.5× larger delay, 54× larger area, 14× more dynamic power and 87× more static power. However, the simplified CAD flow, faster turnaround time and lower NRE cost drive designers to use FPGA as their implementation platform of choice.



(a) Cross Section of an NMOS Transistor (b) Cross Section of a Flash Transistor

Figure 1.10: NMOS and Flash Device Structures

I.9 Floating Gate (Flash) Transistors

I.9.1 Device Structure

Figure 1.10a shows the cross section of an NMOS transistor. The NMOS transistor is a field effect transistor (FET). It has four terminals. Three of these terminals (drain, gate and source) are shown in Figure 1.10a, while the body terminal is not shown in the figure. When the voltage difference between the gate and source terminals is high enough (higher than the threshold voltage of the NMOS transistor), an inversion layer is formed in the P-type substrate between the N-type source and drain of the NMOS transistors. This inversion layer allows the electrons to flow between the source and drain. Figure 1.10b shows the cross section of a flash transistor. Flash transistors are field effect transistors (FET) devices with two gates. In addition to a control gate which is similar to a regular transistor's gate both physically and functionally, a flash transistor has an additional *float-*

ing gate. The floating gate is buried within the device's structure (between the substrate and the control gate). As the name suggests, the floating gate is not contacted, preventing it from being driven directly. Since the floating gate is placed between two dielectric layers, current cannot flow into or out of the floating gate, unless electrons are forced to enter (leave) the floating gate from (to) the substrate. The phenomenon by which electrons tunnel through a barrier is called Fowler-Nordheim (FN) tunneling [39].

I.9.2 Programming the Flash Transistor

Fowler-Nordheim tunneling is used to program and erase a flash transistor. When a flash transistor is "erased", electrons are forced to tunnel from the floating gate into the substrate, resulting in a drop in the threshold voltage (V_T) of the transistor. The threshold voltage of an erased flash transistor is typically below zero. When a flash transistor is "programmed", electrons are forced to tunnel from the substrate into the floating gate. This increases the transistor's threshold voltage. The threshold voltage of a flash transistor can be adjusted with a fine granularity, allowing the designer to program the flash transistor's threshold voltage with high accuracy. This fine granularity threshold voltage control is accomplished by controlling the duration of the programming pulse used to program the flash transistor. The ability to control the threshold voltage of a flash transistor allows us to control the transistor behavior when a signal is applied to the control gate. For example, if a flash transistor is programmed with a threshold voltage V_1, the transistor will turn on only if the signal driving the control gate is higher than V_1, otherwise the transistor will turn off. In flash-based memory applications, two, four or eight threshold voltages are

used (resulting in the storage of one, two or three bits per flash transistor respectively).

Programming a flash transistor is performed by holding the bulk, source and drain terminals at ground and applying a high voltage (10-20 Volts) to the control gate terminal of the flash transistor. This creates an electric field that forces electrons to tunnel from the substrate into the floating gate, hence increasing the threshold voltage of the flash transistor. The value of the threshold voltage is modified by controlling the duration of the programming pulse. Once electrons are trapped in the floating gate, they remain trapped for several years [40, 41], or until removed by an erase operation. Erasing a flash transistor is performed by holding the control gate voltage at ground, floating the drain and source terminals, and applying a high voltage at the bulk of the transistor. This results in an electric field that forces the electrons to tunnel from the floating gate back to the substrate. Unlike programming, when erasing is performed, all the flash transistors that share the same bulk are erased at once, provided their drain, source and gate are connected as mentioned above.

I.9.3 Flash Transistors in Memory

The ability to electrically control the threshold voltage of the floating gate transistor has traditionally been used to build flash memory. Flash memory is used to build secure digital (SD) memory cards, universal serial bus (USB) flash drives and solid state drives (SSDs), which are compact and fast non-volatile block storage devices (compared to traditional non-volatile block storage devices such as hard disk drives (HDDs)). These flash memories are used to store a *block* of data, where each block consists of N flash transis-

tors connected in series (in a NAND configuration), and M such series stacks, where M is greater than the page size of the memory. Although M can be equal to the page size of the memory, it is usually chosen to be larger than the page size, so as to have redundant columns that can be switched in when cells in the flash block wear out. This NAND configuration results in a dense memory array, because the shared diffusions in the NAND stack are not contacted. The memory is accessed a page at a time (i.e. the i^{th} flash transistor of each of the M stacks is accessed, resulting in a page of data being read). A NOR flash arrangement can also be envisioned, for random access memory, but finds limited use in practice, due to it's lower layout density compared to a NAND flash memory.

Due to advancements in flash technology, it is projected that SSDs will completely replace HDDs in the near future. Current flash memory devices are implemented using single-level cell (SLC), multi-level cell (MLC) or triple-level cell (TLC) flash devices. SLC flash devices store only 1-bit per cell (transistor) by programming the transistor to one of two threshold voltages, typically a high threshold (program) voltage to represent a logic '1' and a low threshold (erase) voltage to represent a logic '0'. Since we can program the threshold voltage of a flash transistor at a fine granularity, we can store 2-bits per cell in MLC flash devices or 3-bits per cell in TLC flash devices. This requires four and eight threshold voltages respectively. MLC and TLC technologies enable more dense flash memory devices.

I.9.4 Read and Write Disturb

The issue of read and write disturb is an important issue in NAND flash memories, especially in MLC flash and TLC flash since multiple V_T levels are used to store data, versus two V_T levels for SLC flash. The structure of NAND flash memory consists of many adjacent, long series stacks of flash transistors, increasing the effect of write disturbs. *Write disturb* in NAND flash memories (also known as *program disturb*) occurs during the programming of a flash transistor (i,j), due to applying a high passing voltage to all other flash transistors in column j. These transistors are called *victims*, and we want to leave their V_T's unchanged. Although the pass voltages are not high enough to program the victim transistors, they can cause a slight shift in the V_T of the victim flash transistors. Write disturbs can also affect victim flash transistors in the *same* row i as the flash transistor (i,j) we intend to *program*, due to the program pulse that is applied to the entire row [42]. Unlike a write disturb, a read disturb occurs during regular operation (i.e. while reading a NAND flash memory), which arguably makes it a more important issue to address, due to the higher number of reads (compared to the number of writes) of a flash transistor in a flash-based FPGA. *A read disturb* is the change of the V_T of the victim flash transistors in the same column of flash transistor that we intend to *read*, due to the application of a passing voltage to all of the victim flash transistors. This pass voltage (although is lower in magnitude than the program pulse) causes slight shifts in the V_T of the victim flash transistors over time, leading to a potential bit flip [42, 43].

I.9.5 Write Endurance

One limitation of flash memory is that flash transistors have a finite number of write cycles (which ranges from 10K-100K cycles [40, 41]), limiting the durability of flash memory unless architectural techniques are deployed to mitigate this problem. This limitation on the number of write cycles in flash transistors is called *write endurance*. Write endurance of flash transistors depends on the number of V_T levels. SLC flash has the highest write endurance, while TLC flash has the lowest write endurance.

CHAPTER II

THESIS OUTLINE

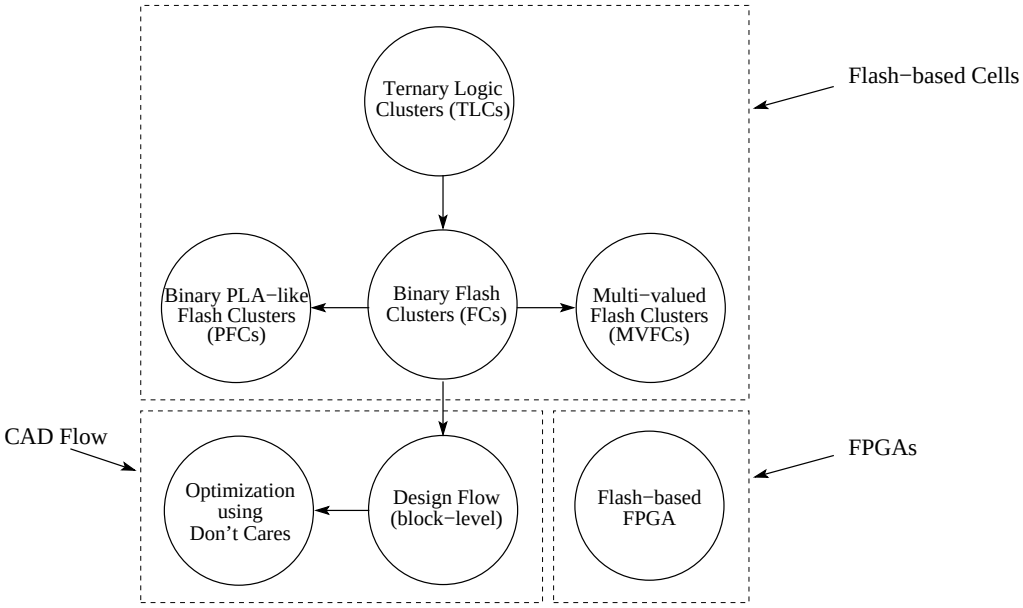


Figure 2.1: Thesis Outline Diagram

This thesis describes the use of flash transistors to implement digital designs. Both ASIC-style designs as well as FPGA designs are covered.

For ASIC designs, at the lowest level of detail, this thesis addresses the implementation of the cell, each of which implements a logic function $F_{m,n}$. We also refer to these cells as *clusters*. Here m is the number of inputs and n is the number of outputs of the function $F_{m,n}$. Various implementations of the flash cells are explored in this thesis, such as *ternary logic clusters* (TLCs), *flash clusters* (FCs), *PLA-like flash clusters* (PFCs) and

multi-valued flash clusters (MVFCs). These flash cells are discussed in the implementation chapters described in the next section, and are illustrated in the top part of Figure 2.1.

A flash-based cell only implements a small portion of the logic in a system. This thesis also discusses the implementation of flash-based digital designs at the block level. Current industry CAD flows do not support flash-based design, and hence, this thesis also develops a flash-based block-level CAD flow. The CAD flow described in this thesis maps a logic netlist into an interconnected network of flash-based logic cells. We have applied the CAD flow on the FC, however the same tool-chain can be applied to all the flash-based logic cells (TLCs, PFCs and MVFCs as well) with minor customizations. Also, in order to be able to measure the circuit characteristics (delay, power, energy and area) of the block-level implementation, we have also developed a tool-chain to perform delay, power, energy and area characterization of the final block-level design. Another contribution of this thesis is a SAT-based optimization engine using don't-cares, to improve the circuit characteristics of the flash-based block-level design. We also show how our flash-based design methodology can be applied to FPGAs as well. FPGAs implemented using our design approach achieve better delay, power and area results. This is illustrated on the bottom portion of Figure 2.1.

We conclude this thesis with a discussion of possible future extensions to this research.

The remainder of this chapter will describe the major sections of the thesis in more detail.

II.1 Implementation of Flash-based Cells

In this section, we will briefly describe the different flash-based cell types – TLCs, FCs, PFCs and MVFCs.

II.1.1 Ternary-valued, Flash-based Digital Logic Cell Implementation

In Chapter III, we describe the implementation of ternary-valued digital circuit cells using flash transistors. We show the details of how to map a binary logic function $F_{m,n}$ with m inputs and n outputs into a ternary-valued logic function $G_{p,q}$ with p inputs and q outputs, and implement $G_{p,q}$ as a *ternary logic cluster* (TLC). We also show the circuit details of the TLC, and also show how the TLC is programmed. We evaluate the delay, power, energy and physical area metrics of flash-based digital designs implemented using the TLC, and compare these metrics to a CMOS standard cell based implementation of the same digital design. We also show a representative layout of the TLC, and quantify the effect of shifting the device threshold voltage (V_T) on the delay, power and energy of the TLC.

II.1.2 Binary-valued, Flash-based Digital Logic Cell Implementation

In Chapter IV, we describe the implementation of binary flash-based digital circuit cells using a *flash cluster* (FC). The FC is a cell that can implement any binary-valued logic function $F_{m,n}$ (where m is the number of inputs, and n is the number of outputs of the logic function $F_{m,n}$). We present the circuit details of the FC as well as the FC programming details. Note that in the FC, the input minterms are grouped based on which output

minterm they generate. Afterwards, the input minterms in each group are minimized using the Espresso [29] logic minimization tool. We evaluate the delay, power, energy and physical area metrics of flash-based digital designs implemented using the FC, and compare these metrics to a CMOS standard cell based implementation. We present the layout of such an FC. We also demonstrate the ability to control binning, and mitigate aging by shifting the V_T of the flash transistor in the FC after fabrication. Finally, we quantify the delay, power and energy of flash-based digital circuits at different process corners.

II.1.3 PLA-like, Flash-based Digital Logic Cell Implementation

In Chapter VII, we explore the implementation of flash-based digital circuits using a variation of the approach of Chapter IV. The main difference between the implementation in Chapter VII and Chapter IV is the way the output is generated. In Chapter VII the input minterms of $F_{m,n}$ are minimized together, using Espresso-MV [29], and then the output is generated in a manner similar to a PLA (as shown in Section I.7). This implementation results in a reduction in the area of the design, on the account of cube sharing. We call this flash-based cell a *PLA-like flash cluster* (PFC). A PFC can implement any logic function $F_{m,n}$, where m is the number of inputs, and n is the number of outputs.

II.1.4 Multi-valued, Flash-based Digital Logic Cell Implementation

In Chapter VIII, we generalize the binary flash-based digital circuit implementation to any multi-valued function. The cell that implements multi-valued flash-based logic designs is called *multi-valued flash cluster* (MVFC). An MVFC can implement any multi-value logic function $H_{r,s}$, where r is the number of inputs and s is the number of out-

puts. In this chapter, we implement ternary-valued digital circuits using this technique, in order to compare it with the ternary implementation in Chapter III. We show that the ternary-valued flash-based cells implemented using the approach in Chapter VIII achieves improved delay results compared to the ternary-valued, flash-based designs implemented using the approach in Chapter III. This improvement is mainly due the increased V_{gs} values in the multi-valued implementation of Chapter VIII. We also compare the results obtained from implementing flash-based designs using the approach in Chapter VIII to their CMOS standard cell counterparts.

II.2 CAD Flow

Section II.1 described our approaches to implement digital logic at the cell level, using flash transistors. In this section, we describe our design flows to implement digital designs at the block level, using a flash-based approach.

II.2.1 Block-level Implementation of Binary-valued, Flash-based Digital Designs

Since the flash-based design technique is not supported by currently available CAD tools, we present, in Chapter V, a CAD flow targeting the implementation of digital circuit blocks using binary flash-based design approach. This CAD flow performs the synthesis and mapping of the design to flash-based circuit clusters (FCs). We present, through circuit simulations, the delay, power, energy and physical area results obtained by the flow, and compare these results against a traditional CMOS standard cell-based design approach. We also show the delay, power and energy curves of the flash-based designs implemented

using this flow, as a function of V_T shift.

II.2.2 SAT-based Optimization for Flash-based Digital Designs

The flash-based design approach discussed in Chapter V uses a greedy algorithm in the clustering step. In order to optimize the design, we present, in Chapter VI, a SAT-based optimization engine. This SAT-based engine uses don't-cares to reduce the size of the FCs of the design. Our optimization approach has many advantages over traditional optimization techniques that use don't-cares. These advantages include a) we target a flash-based design flow, b) we optimize clusters of logic nodes at once instead of one node at a time, c) we attempt to reduce the number of cubes instead of reducing the number of literals in each cube and d) we perform optimization on the post-technology mapped netlist, which results in a direct improvement in result quality, as compared to pre-technology mapping optimization that is typically done in the literature. We compare the delay, power and area results over several benchmark designs, implemented using the flash-based implementation with SAT-based optimization, to flash-based designs implemented using the flow presented in Chapter V. We also compare the results to a CMOS standard cell implementation of the same benchmarking designs.

II.3 Field Programmable Gate Array (FPGA)

This thesis also presents an efficient way to realize an FPGA using flash transistors. In this section, we describe our flash-based FPGA implementation.

II.3.1 Flash-based Field Programmable Gate Array (FPGA)

In current flash-based FPGAs, flash-based configuration cells are used to replace the SRAM-based configuration cells which are used in traditional SRAM-based FPGAs. In Chapter IX, we embed the configuration flash transistors in the path of the logic and interconnect, reducing the delay and area of the design. We also show both a static and a dynamic implementation of the FPGA LUT. Furthermore, we compare the delay, power and energy of the logic and interconnect in the our static and dynamic flash-based FPGA to traditional CMOS-based FPGAs as well as existing flash-based FPGAs.

CHAPTER III

TERNARY-VALUED, FLASH-BASED DIGITAL LOGIC CELL IMPLEMENTATION[1]

In this chapter, we present a method to use floating gate (flash) transistors to implement low power ternary-valued digital logic cells targeting handheld and IoT devices. We exploit the ability to fine tune the V_T of flash transistors to implement ternary-valued digital circuits. The circuit topology we utilize is a cluster of unprogrammed flash transistors arranged in a NAND flash-like configuration (we call these *ternary logic clusters (TLCs)*), which are programmed in the factory to implement the desired logic function.

In the proposed *ternary logic cluster* (TLC) structure, as the name suggests, we use three-valued logic to implement digital circuits. Since flash transistors can have multiple V_T values, we can in principle build a circuit that implements any *Multi-Valued* (MV) logic function. However, to avoid circuit complexity and to maintain a healthy noise margin, we limit our implementation to three-valued logic. In our design, we use three threshold voltage levels to create a device that can distinguish between three input voltage levels (GND, $\frac{VDD}{2}$ and VDD) *without* the use of sense amplifiers.

[1]Part of the data reported in this chapter is reprinted with permission from "A Ternary-valued, Floating Gate Transistor-based Circuit Design Approach" by Monther Abusultan and Sunil P. Khatri, IEEE Computer Society Annual Symposium on VLSI (ISVLSI) 2016, pp. 1-6, Copyright 2016 by IEEE.

III.1 Background

In our design approach, the flash devices are programmed after fabrication (in the factory), and then again to possibly adjust for aging effects (in the field). The ability to perform fine adjustments to the device V_T will also allow the manufacturer to perform fine grained speed binning at the factory. Additionally, when transistors slow down as a result of aging, another round of fine grained threshold voltage adjustment can be performed, to bring the IC performance back within specifications. In addition, by leaving a portion of flash devices unprogrammed, our circuit has the ability to support post-manufacturing engineering change orders (ECOs). This is not possible in present-day CMOS technology, since ECOs in CMOS technology require metal mask changes, which can be expensive. Also, in present-day CMOS, speed binning is completely dependent on process variations, since device V_T's cannot be changed after fabrication. Finally, in-field performance adjustment (to counteract aging problems) is not possible in present-day CMOS designs.

Although flash transistors have a finite number of write cycles (10K to 100K [40, 41]), which is an issue for flash memory devices, this will not be an issue when using flash transistors to implement digital circuits. This is because the number of write cycles needed to realize the desired digital circuit will be very limited (a handful at most).

It is very important to note that our proposed structure is *not* an FPGA-like reprogrammable structure. Our structure is *not fully programmable* like in an FPGA, because the metalization of the design is fixed (i.e. interconnects are hardwired and not reconfigurable after fabrication like in an FPGA). Also, we note that the flash fabrication process

is inherently compatible with the CMOS fabrication process. In fact, flash memories use both flash and CMOS transistors simultaneously on the same die [44]. In flash memories, the CMOS devices are used for the controller, addressing logic, driver logic, and analog functions such as programming voltage pulse generation. Our work assumes that both flash and CMOS devices are present on the same die.

III.2 Previous Work

There have been several research efforts which study flash devices and their use in memory [45, 46, 47, 48, 49, 50, 51, 52]. These papers report details of flash devices and their characterization. However, they do not describe the use of flash transistors for implementing logic circuits. To the best of our knowledge, there has been no prior work that uses flash devices to realize digital circuit structures.

A good deal of work has been reported in the area of architectural techniques to increase flash endurance. Some representative works include those on *wear leveling* techniques, which are used in flash-based memory blocks [53], to compensate for the fact that flash transistors typically have a finite (10k - 100k) number of times they can be written [40, 41]. In traditional flash memory, wear leveling is performed at the architectural level to spread the wear of the cells.

None of these research efforts describes an approach to design ternary-valued digital circuits using flash transistors. To the best of our knowledge, this is the first work to report the use of ternary-valued flash transistors to realize digital circuits.

III.3 Approach

III.3.1 Overview

In this section, we first discuss the details of the approach to convert a CMOS standard cell based digital circuit design into an equivalent (dynamic) ternary-valued flash-based digital circuit design (in Section III.3.2). The ternary-valued flash-based design consists of several Ternary Logic Clusters (TLCs). Each ternary logic cluster implements an q-output (ternary) function with up to p ternary-valued inputs. The circuit details of a TLC are described in Section III.3.3. In our implementation, we choose the number of outputs $q = 2$, and the number of inputs $p = 4$. This choice is explained in Section III.3.2. Each TLC consists of several Ternary Logic Arrays (TLAs), which in turn are made up of several Ternary Logic Bundles (TLBs). Details of these components will be discussed in Section III.3.3. In Section III.3.4, we discuss the programmability of the TLCs. Results are discussed in Section III.4.

III.3.2 Ternary Logic Flash-based Design Conversion

The conversion process from a regular CMOS standard cell based digital circuit design into a ternary-valued flash-based design comprises of two parts – a) the handling of binary-valued I/O interfaces and b) the conversion of a CMOS standard cell-based design into a TLC-based design.

Figure 3.1 shows the basic idea behind our approach to handling binary-valued I/O interfaces. We use solid lines to represent binary signals and dashed lines to represent ternary-valued signals. While direct MV synthesis can be performed to produce a native

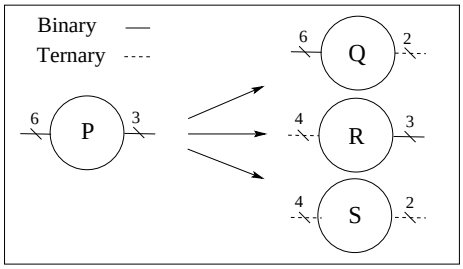
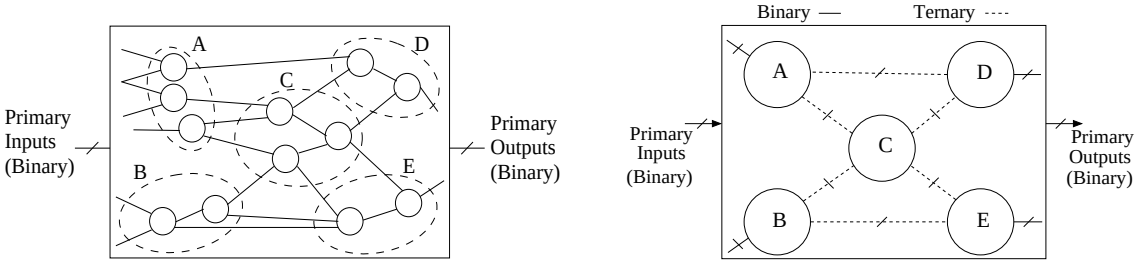


Figure 3.1: Types of Ternary Logic Nodes in Our Implementation.

three-valued logic function, this approach requires implementing the entire system (including all its subcomponents and I/O interface logic using ternary-valued logic). We avoid this approach since current technology exclusively supports binary logic I/O interfaces. Instead, we implement our system to interface with the outside world using binary logic, while internally using ternary-valued logic to implement the required logic functions. This can be realized by having three types of logic nodes, as shown in Figure 3.1. Any binary logic node P can be converted into a ternary-valued logic node Q, R or S. Node Q has binary inputs and ternary-valued outputs. It serves as the conversion node from binary to ternary-valued in our system, and would be used at the input interface. Node R has ternary-valued inputs and binary outputs in order to convert ternary-valued logic back into binary logic. It would be used at the output interface. Node S represents a logic node that has ternary-valued inputs and outputs. We use node S for the internal nodes of the ternary-valued flash-based logic block. In order to eliminate the design overhead, we use the same ternary circuit (the TLC) to implement nodes of type Q, R and S. The TLC in general implements a node of type S.

Converting a u-bit binary function into a v-bit ternary-valued function requires mapping 2^u binary minterms into 3^v ternary-valued minterms, such that $2^u < 3^v$. We found that choosing ($u = 3$), and ($v = 2$) yields the least number of "unused" ternary-valued minterms. This means that when converting a binary-valued design into a ternary-valued design, we choose binary clusters with numbers of inputs and outputs that are of multiples of 3. This yields a ternary-valued function with numbers of inputs and outputs that are of multiples of 2.



(a) Logic Netlist (b) Flash-based Digital Circuit

Figure 3.2: Converting a Logic Netlist into a Flash-based Design

Figure 3.2 illustrates the conversion of a CMOS standard cell based digital circuit into a ternary-valued digital circuit from the circuit block perspective. Starting with a CMOS-based design (Figure 3.2(a)) which can be technology mapped or technology independent, we cluster the circuit nodes (shown in the dotted circles of Figure 3.2(a)). These clusters are multi-input, multi-output structures (with up to m inputs and n outputs), where both m and n are multiples of 3. The solid circles in Figure 3.2(b) are referred to as *ternary logic clusters* (TLCs). Each TLC implements the logic function of the corresponding clus-

ters (dotted circles in Figure 3.2(a)). In other words, each TLC implements an q-output ternary-valued logic function of up to p ternary-valued inputs. We refer to this function as $G_{p,q}$. Note that $p = \frac{2}{3}m$ and $q = \frac{2}{3}n$. The solid circles labeled A, B, C, D and E in the ternary-valued digital circuit of Figure 3.2(b) have the same functionality and connectivity as the dashed ovals A, B, C, D and E in the CMOS-based binary-valued digital circuit of Figure 3.2(a). TLCs A and B are nodes of type Q (see Figure 3.1), while D and E are of type R. TLC C is of type S.

Each of the TLCs is implemented using our ternary-valued design approach. Figure 3.2 (b) shows a high-level representation of the final ternary-valued design (with binary valued I/O interfaces) after converting all the clusters of logic in the CMOS-based binary-valued design into their ternary-valued TLCs (with binary I/O interfaces). The design details of the TLC are discussed in the next section.

Note that the focus of this chapter is on the ternary-valued TLC design, and its electrical characteristics. Therefore, we will not discuss the algorithmic details of converting a CMOS-based binary-valued design into a ternary-valued design in the remainder of this chapter.

III.3.3 Ternary Logic Cluster (TLC) Circuit Design

We construct the ternary-valued digital circuit by identifying clusters of nodes in the CMOS-based circuit that implement a n-output binary logic function of up to m binary inputs ($F_{m,n}$). These clusters of logic will then be implemented using TLCs whose logic function $G_{p,q}$ is equivalent to $F_{m,n}$, where $p = \frac{2}{3}m$ and $q = \frac{2}{3}n$. A TLC is a reprogrammable

circuit structure that is capable of implementing any logic function with p ternary-valued inputs and q ternary-valued outputs ($G_{p,q}$). TLCs are equipped with the required logic for programming the threshold voltages of their floating gate devices, in addition to the circuitry needed to implement the logic of $G_{p,q}$. Figure 3.3 shows the block diagram of a TLC. As shown in this figure, the TLC is driven by ternary-valued *Primary Inputs* (left side of Figure 3.3) and ternary-valued *Primary Outputs* (right side of Figure 3.3) as well as additional signals which are used to program the function of the TLC. The programming signals of the TLC (*mode*, *row_select*, *col_select*, *prog_ctrl* and *prog_pulse*), will be discussed in Section III.3.4. The remaining signal is the clock (*clk*), which is used for driving the precharge and evaluate transistors in the dynamic ternary-valued digital circuit. The *clk* signal is also used during programming, as we will discuss in Section III.3.4.

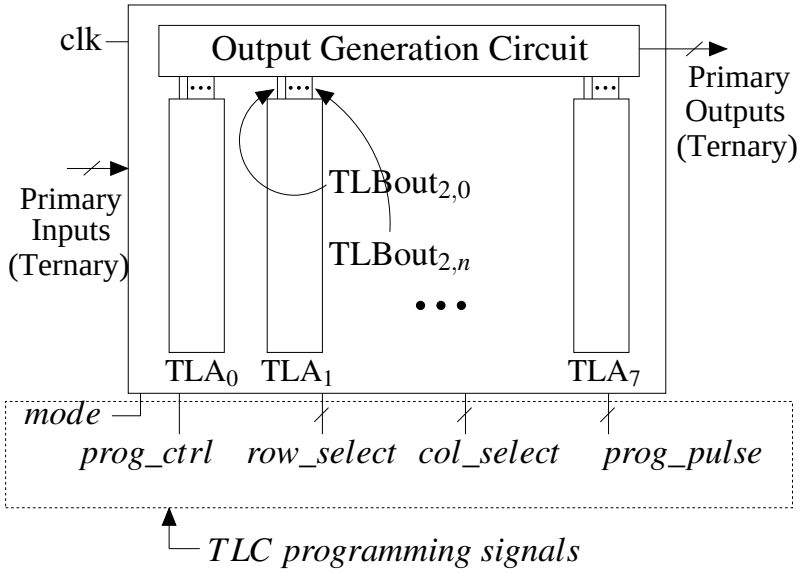


Figure 3.3: Ternary Logic Cluster (TLC)

The TLC internally consists of multiple *ternary-valued logic arrays* (TLAs) and an *output generation circuit* as shown in Figure 3.3. A TLA is a group of NAND flash-like pulldown stack structures. Each stack implements a ternary-valued logic cube of $G_{p,q}$. Each TLA implements a group of ternary-valued input cubes that correspond to a ternary-valued output minterm of $G_{p,q}$. For example, if the ternary-valued output $< 02 >$ (alternatively represented by $< 001|100 >$) has 2 ternary input cubes $< 001|010|010|011 >$ and $< 001|010|100|111 >$, then TLA_2 implements these two ternary-valued input cubes. In other words, only one TLA output pulls low when a ternary-valued input is applied to the TLC. For the i^{th} TLA, we refer to the number of cubes that this TLA implements as its *cubes per array* or CPA_i. In the example of this paragraph, ($CPA_2 = 2$). The outputs of the TLAs are connected to the output generation circuit in the TLC such that when the output of TLA_i pulls down, the output generation circuit produces the q-bit ternary-valued output vector i. For example, if $q = 2$ and if TLA_5 pulls down during evaluation, then the output generation circuit produces the 2-bit ternary output $< 12 >$ (or $< 010|100 >$). Note that outputs 2^n through $3^q - 1$ are never generated (as explained in Section III.3.2). Hence, we do not need to implement $TLA_{(2^n)}$ through $TLA_{(3^q-1)}$. Therefore, we only need to implement 2^n TLAs (shown as TLA_0, TLA_1, $\cdots$, TLA_7 in Figure 3.3, for $n = 3$ and $q = 2$).

Next, we will discuss the structural details of a TLA, and then the details of the output generation circuit.

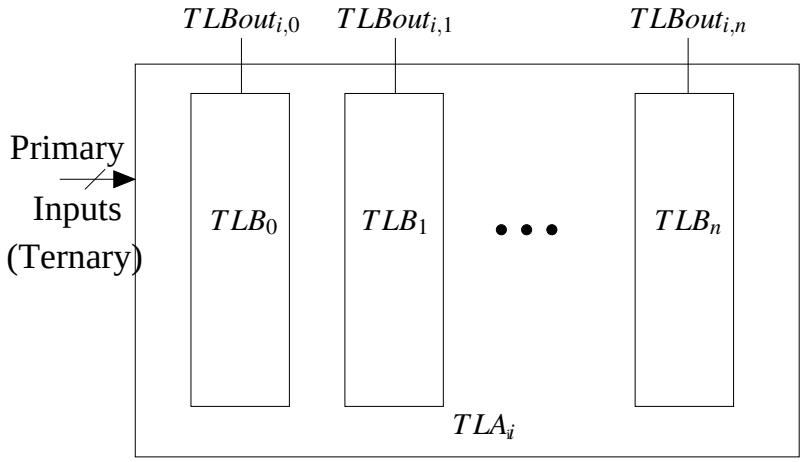


Figure 3.4: i^{th} Ternary Logic Array (TLA_i) Structure.

III.3.3.1 Ternary Logic Array

Figure 3.4 shows a block diagram of the structure of TLA_i. As shown in this figure, each TLA consists of multiple TLBs. The number of TLBs that exist in TLA_i is $\lceil \frac{CPA_i}{CPB} \rceil$, where CPA_i is the number of cubes implemented in TLA_i, and CPB is the maximum number of cubes that can be implemented in a TLB. Note that while CPA_i is determined by the logic function $G_{p,q}$, CPB is a design parameter that can be optimized to improve circuit delay, power, energy and physical area. The number of outputs of TLA_i is equal to the number of TLBs in TLA_i, (namely $\lceil \frac{CPA_i}{CPB} \rceil$). At most one of the TLB outputs ($TLBout_{i,k}$) is pulled down when a ternary-valued input is applied to the TLA inputs.

III.3.3.2 Ternary Logic Bundles

Figure 3.5 shows the circuit details of a TLB. A TLB consists of a number of NAND flash-like pulldown stacks that share the same output. Each pulldown stack implements

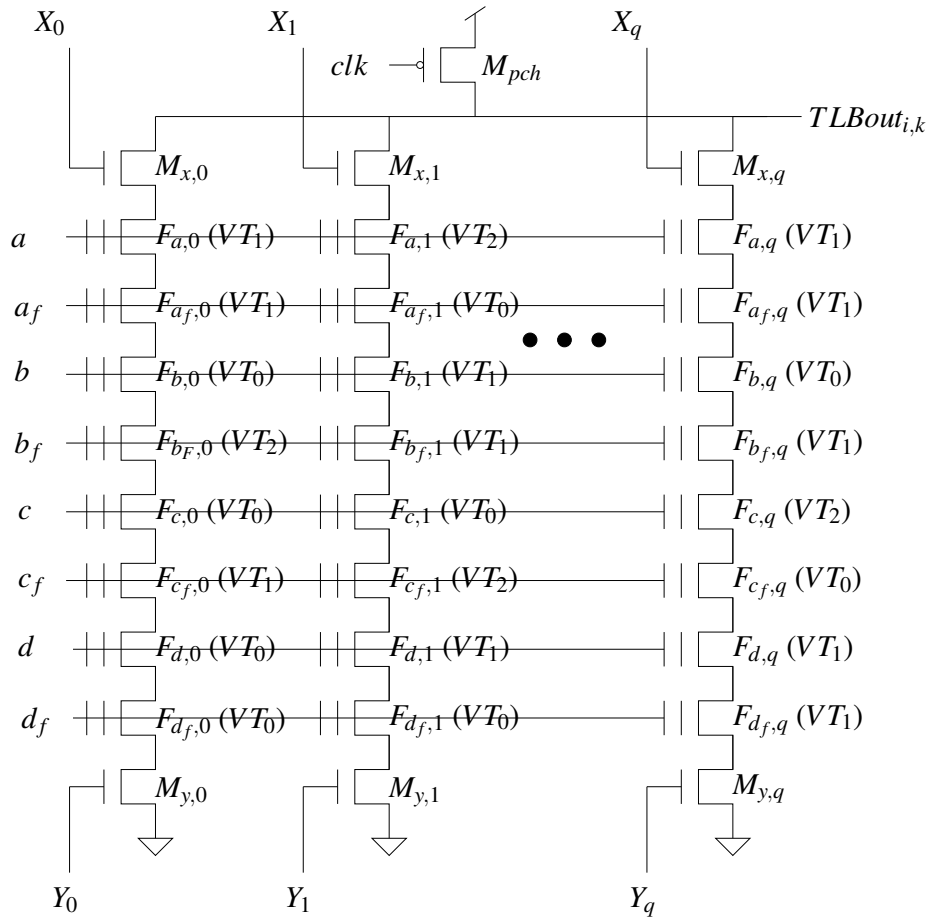


Figure 3.5: k^{th} Ternary Logic Bundle in TLA_i ($\text{TLB}_{i,k}$)

one cube of $G_{p,q}$. The maximum number of NAND flash like pulldown stacks in each TLB is CPB, where CPB is the maximum number of cubes that can be efficiently implemented in a TLB. Since the number of cubes implemented by TLA_i is not necessarily a multiple of CPB, the last TLB in TLA_i may have a smaller number of pulldown stacks than CPB.

Each pulldown stack has $2 \cdot p$ flash transistors and 2 regular NMOS transistors (as shown in Figure 3.5), where p is the number of ternary-valued inputs of the function $G_{p,q}$. The reason we need two flash transistors for each input is discussed next. The flash

transistors are programmed to implement cubes, while the regular transistors have a dual purpose. Since the ternary-valued implementation proposed in this chapter is dynamic, both precharge and evaluate transistors are required. The shared regular PMOS transistor shown at the top of Figure 3.5 (M_{pch}), serves as the precharge transistor for all pulldown structures of the TLB. When it turns on, it pulls up $TLBout_{i,k}$ during the precharge (low) phase of the clock signal (clk). The regular NMOS transistors ($M_{y,i}$) shown at the bottom of each stack in Figure 3.5 are the evaluate transistors which are off during the precharge (low) phase of clk and only turn on during the evaluate (high) phase of clk, to allow the pulldown stack to evaluate the output $TLBout_{i,k}$. Both the top and bottom regular NMOS transistors in each pulldown stack ($M_{x,i}$ and $M_{y,i}$ respectively) are also utilized during the programming operation of the NAND flash stack. Programming will be discussed in detail in Section III.3.4. The transistors $M_{x,i}$ are always 'ON' during regular operation of the ternary-valued design.

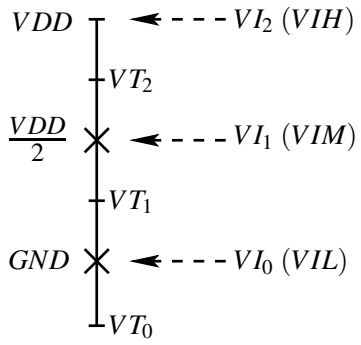


Figure 3.6: Flash Transistor Threshold Voltages

Since we are implementing a ternary-valued logic circuit, we need three threshold

voltages to distinguish between the three states of each input. Figure 3.6 shows the input voltages and the threshold voltage levels used in our design. Flash transistors are driven with either a VDD (representing the ternary-valued literal $< 100 >$), $\frac{VDD}{2}$ (representing $< 010 >$) or GND (representing $< 001 >$), and can be programmed to VT_2, VT_1 or VT_0 as shown in Figure 3.6. Since a flash transistor will turn on when driven by an input voltage higher than its programmed V_T, we have to design the pulldown stack in a way that allows us to distinguish between an input of VDD, $\frac{VDD}{2}$ or GND applied at the gate of a flash transistor. For example, we cannot program a flash transistor such that it will *only* turn on when the gate input is $\frac{VDD}{2}$, because *both* VDD and $\frac{VDD}{2}$ will turn on this flash transistor which is programmed at VT_1. This problem is averted in multi-level flash memory by reading each transistor's response multiple times, each time with a different input voltage. This solution is infeasible for implementing logic circuits, due to the increased delay it would entail. We overcome this issue by using two flash transistors for each input. One transistor is driven by the input x and the other is driven by x_f. The input x_f is a *flipped* version of x as discussed in Table 3.1. The two transistors are then programmed based on the value of the ternary-valued literal x as shown in Table 3.2. In Figure 3.5, the left-most stack is programmed to implement the cube $< abcd > = < 010|001|011|111 >$, based on Table 3.2. This is the ternary-valued cube $a^1b^0c^{0,1}$. Similarly, the second stack of Figure 3.5 implements the cube $< 100|010|001|110 >$. The rightmost stack of Figure 3.5 implements the cube $< 010|011|100|010 >$.

Input Voltage Level	Flipped Input Voltage Level
VDD	GND
$VDD/2$	$VDD/2$
GND	VDD

Table 3.1: The 'Flip' Function.

Literal	V_T of F_x	V_T of F_{x_f}	Function
001	VT_0	VT_2	ON when x = GND
010	VT_1	VT_1	ON when $x = VDD/2$
100	VT_2	VT_0	ON when x = VDD
011	VT_0	VT_1	ON when x = GND or $VDD/2$
110	VT_1	VT_0	ON when $x = VDD/2$ or VDD
111	VT_0	VT_0	Always ON
101	–	–	Split cube into 001 and 100
000	–	–	Null literal, invalid

Table 3.2: Programming States of Each Transistor Pair F_x and F_{x_f} for Each Literal.

III.3.3.3 Output Logic

As discussed earlier in Section III.3.2, each TLA in the TLC drives an output generation circuit that generates the final outputs of the TLC as shown in Figure 3.3. Figure 3.7 shows the circuit details of the output generation circuit. Each output of the function $G_{p,q}$ is represented using a horizontal line which is pre-discharged by an NMOS transistor (shown at the left side of the output line). This NMOS transistor is driven by the clock signal ($\overline{clk}$). Each of the output lines is selectively pulled up based on which TLA output is pulled down. For example, for $q = 2$, if $TLBout_{7,k}$ pulls down for any k, then the outputs need to drive to $< 2, 1 >$. This means that the MSB output (f in Figure 3.7) needs to be

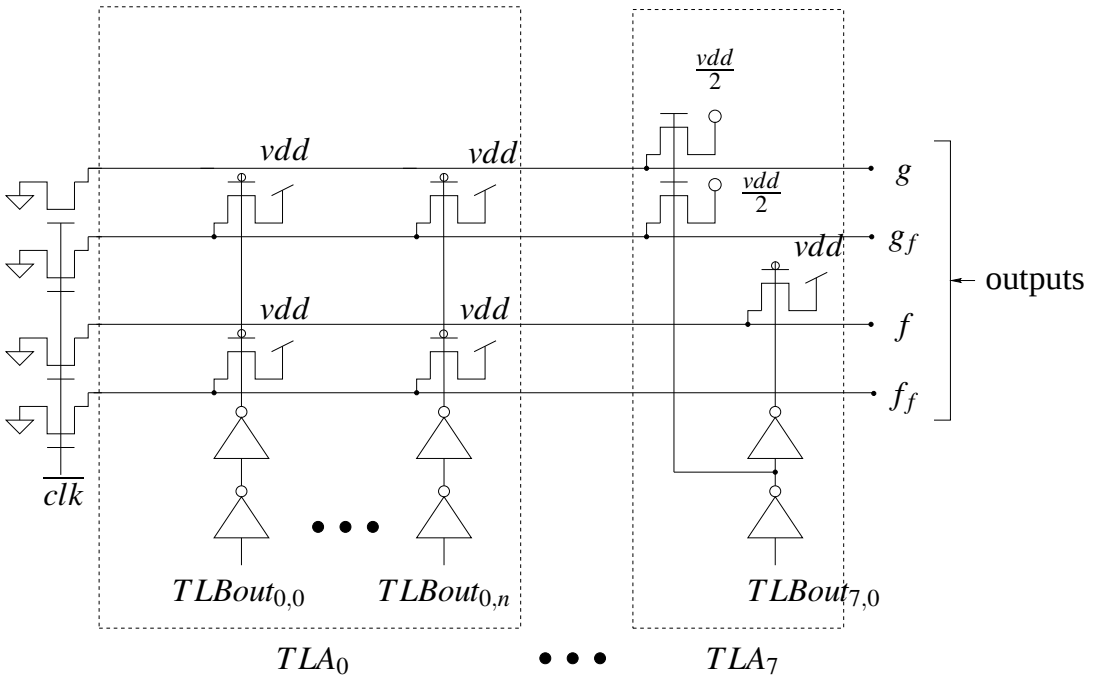


Figure 3.7: Flash Output Generation Circuit

driven to VDD, and the other output (g in Figure 3.7) needs to be driven to $\frac{VDD}{2}$. Note that exactly one $TLBout_{i,k}$ pulls down for any applied input to the TLC. All the transistors driving the outputs of the function $G_{p,q}$ shown in Figure 3.7 are sized appropriately to drive a fan-out of 3 TLC input loads.

III.3.4 Programming the Flash Ternary Logic Cluster

Consider Figure 3.5. In any TLC, all the flash transistors share a common bulk. As a result, the erase operation of all flash transistors of the TLC is performed by applying a high voltage to the bulk node of the TLC, and floating the source and drain terminals of each flash transistor (by turning off $M_{y,i}$ and $M_{x,i}$). The gates of all transistors are driven to GND. This results in the erasure of all the flash transistors in the TLC, and their new

threshold voltage is the erase threshold (VT_0), as shown in Figure 3.6.

For programming, assume that $F_{c,0}$ and $F_{c,1}$ need to be programmed. In this case, the c line is driven to a programming voltage for a sufficiently long duration. The gates of transistors $M_{y,0}$ and $M_{x,0}$, as well as $M_{y,1}$ and $M_{x,1}$ are driven high. All other $M_{y,i}$ and $M_{x,i}$ are driven low. This disables programming of all but the first and second NAND stacks of the TLB. All inputs other than c (i.e. a, a_f, b, b_f, c_f, d and d_f) are driven high to a pass voltage, and the common bulk is held to GND. The duration of the programming pulse is determined based on the final desired V_T. This results in a programming of $F_{c,0}$ and $F_{c,1}$, to the desired V_T (VT_1 or VT_2), while the thresholds of all other transistors in the TLB are unaltered and stay at the erase threshold voltage (VT_0).

The *mode* signal of Figure 3.3 switches between regular operation and programming. The *prog_ctrl* signal switches between the erase and program operations. The signals *row_select* and *col_select* determine which row and column of the TLB is to be programmed, while the *prog_pulse* signal is the programming voltage pulse that is applied to accomplish programming.

III.3.5 Logic Minimization of the TLC

Given a binary cluster with $m = 6$ and $n = 3$, we take all the input minterms $\{M_j\}$ of each output minterms j (there are 8 output minterms in all) and map them to the j^{th} ternary-valued output minterm. We now use Espresso-MV [29] to minimize the TLC. The resulting multi-valued cover is used to realize the ternary-valued circuit.

III.4 Experiments

In this section, we first present the simulation environment used in evaluating our ternary-valued digital circuit design approach. Then we discuss the ternary-valued digital circuit implementation details. Finally, we present the details of our experiments and a discussion of the results.

III.4.1 Simulation Environment

The designs presented in this chapter are implemented in a 45nm process technology. The CMOS-based digital circuits were synthesized and mapped to 45nm Nangate FreePDK45 Open Cell Library [54, 55] using Synopsys Design Compiler [56]. The mapped designs were simulated using Synopsys HSPICE [56] circuit simulation tool and the 45nm PTM [57] card. The nominal supply voltage for the 45nm PTM card is 1V. We used custom scripts to generate the TLC (the ternary-valued digital circuit). We back annotate the TLC circuit with layout parasitics. For CMOS devices in the TLC, we used a 45nm PTM process [57], while for flash devices, we derived our model card from the device-level measurements presented in [46, 45] and validated our models using [58, 45]. We describe our model card construction in Section III.4.2. We simulated the ternary-valued digital circuit in HSPICE and verified the correct logical operation of the ternary-valued digital circuit through exhaustive simulation. Custom layouts for the ternary-valued digital circuit were generated using Cadence Virtuoso to compare the physical area of the ternary-valued digital circuits to their CMOS-based counterparts. We generated 20 random circuit designs to evaluate our ternary-valued digital circuit design approach. The layout of our

TLCs used design rules for flash devices that were obtained from the ITRS [59].

III.4.2 Flash Model Card Regression

We derived our flash model card from the device-level measurements presented in [46, 45]. The basic idea is to emulate the states of a floating-gate device with three separate PTM model cards, one that models the flash FET in the low V_T state (we call this value VT_0), and another for the flash FET in the medium V_T state (we call this value VT_1), and a third for the flash transistor in the high V_T state (we call this value VT_2). We used the gate and oxide thicknesses, and doping levels from [46, 45]. We then took a base 45nm PTM CMOS model card and modified it so that the threshold voltages of the three derived model cards would be VT_2, VT_1 and VT_0, respectively, and the I_{ds}-V_{gs} curve slopes match that in [46] to model circuit delay and power accurately. We verified that the electrical characteristics of our derived model cards substantially agree with measured device characteristics of industrial flash devices reported in [58].

We also modeled the gate capacitance of the flash transistors. Flash transistors have a lower gate capacitance due to the difference in their gate structure compared to a regular NMOS transistor. The dielectric insulator separating the floating gate from the substrate is Silicon Dioxide (SiO_2) which what is used in regular NMOS transistors, to separate the gate from the substrate. However, the thickness of this insulating layer in a flash transistor is 7nm compared to 1nm in the corresponding 45nm regular NMOS transistor [45, 57, 59]. The other insulating layer in a flash device separates the control gate from the floating gate, and is not found in regular NMOS transistors. This layer consists of a stack of three layers

of insulators. The layers from top to bottom are 4nm SiO_2, 4nm Silicon Nitride (Si_3N_4) and another 6nm SiO_2 [45]. These two differences in the gate structure between the flash transistor and the regular NMOS transistor contribute to lowering the gate capacitance of the flash transistor. We calculated the gate capacitance of the flash transistor and found it 20× smaller than the gate capacitance of the corresponding regular NMOS transistor, and validated this reduction with existing literature which reported reduction of ∼25-30× for a 45nm technology node [45]. The lower gate capacitance of the flash transistors results in a reduced input capacitance of the TLC.

III.4.3 Ternary-valued, Flash-based Implementation Details

In this section, we present the implementation details of the ternary-valued digital circuits. The logic function implemented in the CMOS-based binary-valued digital circuit had $m = 6$ binary inputs and $n = 3$ binary outputs, which was converted into a 4 input (p=4) and 2 outputs (q=2) ternary-valued function to be implemented as a ternary-valued digital circuit. We found that these values of m, n, p and q provide the best tradeoff of delay, power, energy and physical area. The results we present are a comparative study over 20 random functions implemented in both a CMOS standard cell based approach and our ternary-valued digital circuit. We verified the logic correctness of both implementations though exhaustive simulations. The TLC used to implement the logic functions was configured to have TLBs with 3 stacks ($CPB = 3$). The target programmed threshold voltages used in our designs are (VT_0 = -0.5 V), (VT_1 = 0.225 V) and (VT_2 = 0.725 V).

III.4.4 Results and Analysis

Design	D_{max} Ratio	P_{avg} Ratio	*Eng* Ratio	Cell Area Ratio
des00	2.28×	0.12×	0.27×	0.92×
des01	2.25×	0.09×	0.21×	0.89×
des02	2.69×	0.10×	0.27×	0.95×
des03	1.75×	0.17×	0.29×	0.88×
des04	2.50×	0.22×	0.54×	1.04×
des05	2.34×	0.09×	0.22×	1.04×
des06	2.02×	0.19×	0.37×	0.94×
des07	2.63×	0.09×	0.24×	0.94×
des08	2.29×	0.15×	0.34×	0.84×
des09	2.43×	0.09×	0.22×	0.85×
des10	2.96×	0.11×	0.32×	0.86×
des11	2.76×	0.10×	0.27×	0.88×
des12	2.63×	0.09×	0.24×	0.92×
des13	2.18×	0.15×	0.32×	0.96×
des14	2.58×	0.10×	0.25×	1.03×
des15	2.71×	0.18×	0.49×	0.94×
des16	2.78×	0.10×	0.26×	1.10×
des17	2.27×	0.10×	0.23×	0.96×
des18	2.28×	0.14×	0.31×	1.02×
des19	2.35×	0.10×	0.24×	0.97×
Average	2.43×	0.12×	0.30×	0.95×
Stdev	0.29×	0.04×	0.09×	0.07×

Table 3.3: Delay, Power, Energy and Area Ratios of Ternary-valued Logic Circuits Relative to CMOS Standard Cell Based Circuits

Table 3.3 shows the delay, power, energy and physical area ratios of 20 randomly generated logic functions implemented using our TLC compared to a CMOS standard cell based implementation. The delay reported in the table (D_{max} Ratio) is the ratio of the maximum delay of any transition seen at any primary output of the circuit of the TLC versus the

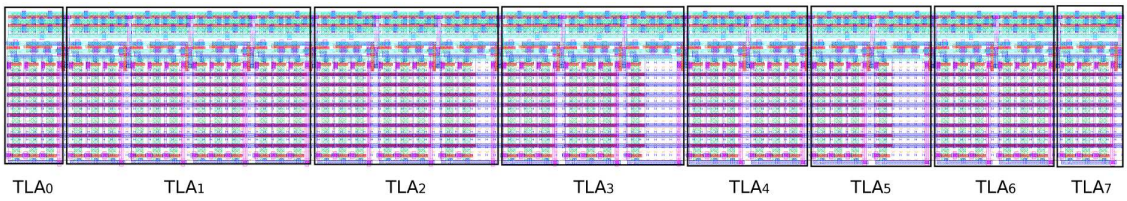


Figure 3.8: Layout View of a TLC (des00)

standard cell design. Since the ternary-valued implementation is dynamic, we accounted for the precharge delay in all the results presented in this chapter. Table 3.3 shows power dissipation (of 0.12×) when implementing the digital circuits using our ternary-valued logic compared to CMOS standard cell based implementation. We also show the energy utilization of our ternary-valued implementation compared to the CMOS standard cell based implementation. On average, the energy utilization of ternary-valued digital circuits is about 0.3× of the CMOS standard cell based implementation. As shown in the table, the delay of the ternary-valued digital circuits ranges from 1.75× to 2.96× of the CMOS standard cell based digital circuit delay, with an average of 2.43×. In other words, the ternary-valued flash-based digital circuit will run at 0.41× the clock rate that an equivalent CMOS-based digital circuit while consuming 0.3× the energy from the battery, which makes it an ideal candidate for applications that run at lower clock rates to conserve the battery life. Table 3.3 also reports the standard deviation of the delay (0.29×), power (0.04×), energy (0.09×) and area (0.07×) ratios. The standard deviation numbers shows that the power, energy and area of the flash-based designs are predictable, while the delay tends to vary based on the design.

We also report the area ratio of both implementations. The area reported for the CMOS standard cell based implementation is the sum of physical cell areas, while the area of our ternary-valued flash-based approach is the layout area obtained from layout generation experiments. Design rules for flash were obtained from the ITRS 45nm flash technology node [59]. Digital circuits implemented in a TLC use $0.95\times$ the physical area of a CMOS-based design, on average. Figure 3.8, shows the representative layout of a TLC for the design des00.

It is well known that dynamic designs consume more power than static CMOS designs. Our TLC based design consumes less power for several reasons. Despite being dynamic, the number of nodes being precharged is smaller than a CMOS (domino or other dynamic) approach. Further, the long transistor stacks (since we choose $m = 4$) result in smaller evaluation currents, reducing power. Also, in our design, exactly one TLB pulls down during every evaluation, reducing switching activity and power consumption. Finally, the I_{ds} of a flash FET is lower than that of a MOSFET, which results in a lower power consumption.

III.5 Chapter Summary

Flash transistors are the workhorse technology for non-volatile data storage applications today. However, there has been no previous research in the use of flash technology to implement ternary-valued digital logic. This chapter presented the first approach, to the best of the authors' knowledge, to use ternary-valued flash transistors to implement digital

circuits. The threshold voltage of flash devices can be modified with a fine granularity during programming, which results in several advantages. First, speed binning at the factory can be done with precision. Secondly, an IC can be re-programmed in the field, to diminish or eradicate effects such as aging. We present the details of the circuit topology that we use in our ternary-valued, flash-based digital circuit approach. Our HSPICE simulations show that our approach yields improved power ($\sim$88%) lower, energy ($\sim$70%) lower and area ($\sim$5%) lower characteristics while operating at ($\sim$59%) lower clock rate compared to a traditional CMOS standard cell based approach, when averaged over 20 designs. Our flash-based design approach to implement ternary-valued digital circuits is intended to target extreme low power/energy applications with modest speed requirements.

CHAPTER IV

BINARY-VALUED, FLASH-BASED DIGITAL LOGIC CELL IMPLEMENTATION[1]

In this chapter, we present the cell-level circuit implementation details for a binary-valued, flash-based digital logic cell. We compare the proposed design to a CMOS standard cell based design approach. Our focus is on the design of the flash-based cell at the combinational cell level. Our design style is fully compatible with conventional sequential elements used in digital circuits.

In this work, we use the ability to fine tune the threshold voltage of flash transistors, to implement arbitrary logic functions. The circuit structure we employ is a multitude of flash transistors in a NAND-like configuration. We call these *flash clusters* (FCs). FCs are programmed in the factory to implement the desired logic function, and can be reprogrammed in the field to perform minor performance adjustments.

IV.1 Background

Flash transistors have a finite number of write cycles (10K to 100K [40, 41]), which is an issue for flash memory devices. This is not an issue when using flash transistors to implement digital circuits. Our flash-based circuit requires a limited number of write cycles to implement the desired digital design. The flash devices in the flash-based designs

[1]Part of the data reported in this chapter is reprinted with permission from "Implementing Low Power Digital Circuits using Flash Devices" by Monther Abusultan and Sunil P. Khatri, 34nd IEEE International Conference on Computer Design (ICCD) 2016, pp. 109-116, Copyright 2016 by IEEE.

will be programmed at the factory after fabrication. However, the flash-based designs can also be programmed in the field to perform performance tuning or to compensate for aging effects. As the transistors slow down due to chip aging, the performance of the ICs can be brought back within specifications by programming the flash transistors and adjusting their threshold voltage. Adjusting the IC's performance in the field, to counteract aging problems is not currently feasible in CMOS designs. The ability to perform fine adjustments to the device V_T enables the manufacturer to perform precise speed binning at the factory. Process variations in present day CMOS control the speed binning. Finally, our flash-based design approach facilitate performing post-manufacturing *engineering change orders* (ECOs). This is done by adding redundant flash devices. ECOs in CMOS designs are expensive since they require metal mask changes and are limited to interconnect adjustments.

The market need for dense and compact flash memory has fueled the advancement in flash technology. The technology node of flash devices has recently been able to track the CMOS technology node. Currently memory devices are being fabricated at sub-20nm minimum feature dimensions similar to CMOS technology node [60, 61, 62, 63]. In this work, we only develop our design approach with a 45nm technology because electrical characteristics of fewer technology nodes are not easily available. Also, commercial grade standard-cell libraries for 45nm are easily available, making our comparison realistic.

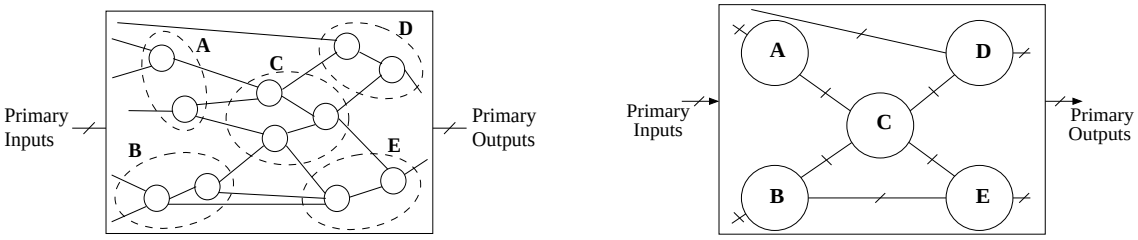
IV.2 Previous Work

As mentioned in Section III.2, previous research efforts have focused only on the use of flash transistors to implement non-volatile memory [45, 46, 47, 48, 49, 50, 51, 52]. The use of ternary-valued logic described in Chapter III and [64] is aimed towards increasing the logical ”expressiveness” of each NAND flash stack. However, the use of multiple threshold voltages results in reduced V_{gs} values, which results in increased worst-case delays. Another factor that contributes to the increased delays in the ternary-valued approach is the large number of transistors connected in series in each of the NAND stacks, since each variable needs 2 series transistors. In contrast, the binary-valued approach of this chapter uses fewer flash transistors in series (only one per input variable) compared to the ternary-valued approach. We use single-level cells (SLC) cells instead of multi-level cells (MLC) cells which were used in the ternary-valued approach, to implement our flash-based digital circuits. This results in higher V_{gs} values and improved delays, and makes our design more immune to read and write disturbs as well. To the best of our knowledge, there has been no prior work that uses flash devices to realize binary-valued digital circuit structures.

IV.3 Approach

A flash-based design consists of several binary *flash clusters* (FCs). Each FC implements an n-output Boolean function with up to m inputs. In Section IV.3.1, we present the general flow of implementing digital design using an interconnected network of FCs. In

Section IV.3.2, we present the circuit details of an FC. Each FC consists of several *flash logic arrays* (FLAs), which consist of several *flash logic bundles* (FLBs). Details of these components will also be discussed in Section IV.3.2. In Section IV.3.3, we discuss the programmability of the FCs, while the immunity to read and write disturbs is discussed in Section IV.3.4.



(a) Logic Netlist (b) Flash-based Digital Circuit

Figure 4.1: Converting a Logic Netlist into a Flash-based Design

IV.3.1 Flash-based Design Conversion

Figure 4.1 illustrates the conversion of a logic netlist into a binary flash-based digital circuit. The process is similar to the conversion of a logic netlist into a TLC-based netlist discussed in Section III.3.2, except that we use FCs to implement the logic functionality. Starting with a logic netlist (Figure 4.1(a)), we first cluster the circuit nodes (in the dotted circles of Figure 4.1(a)). The initial netlist can be technology mapped, or technology independent. These clusters are multi-input, multi-output structures (with up to m inputs and n outputs). The clusters are shown as solid circles in Figure 4.1(b). The solid circles in Figure 4.1(b) are referred to as FCs, and are implemented monolithically. The flash-based

digital circuit implements the logic function of each cluster (dotted circles in Figure 4.1(a)) as an FC (solid circle in Figure 4.1(b)). In other words, each FC implements a logic function $F_{m,n}$, where m is the number of inputs and n is the number of outputs of the logic The solid circles (A, B, C, D and E) in the flash-based digital circuit of Figure 4.1(b) have the same functionality and connectivity as the corresponding dashed ovals (A, B, C, D and E) in the logic netlist of Figure 4.1(a). We discuss the design details of the FC in the next section.

Note that the focus of this chapter is the flash-based FC cell, and its electrical characteristics. We defer the discussion on algorithmic details of converting a logic netlist into a flash-based netlist to the next chapter.

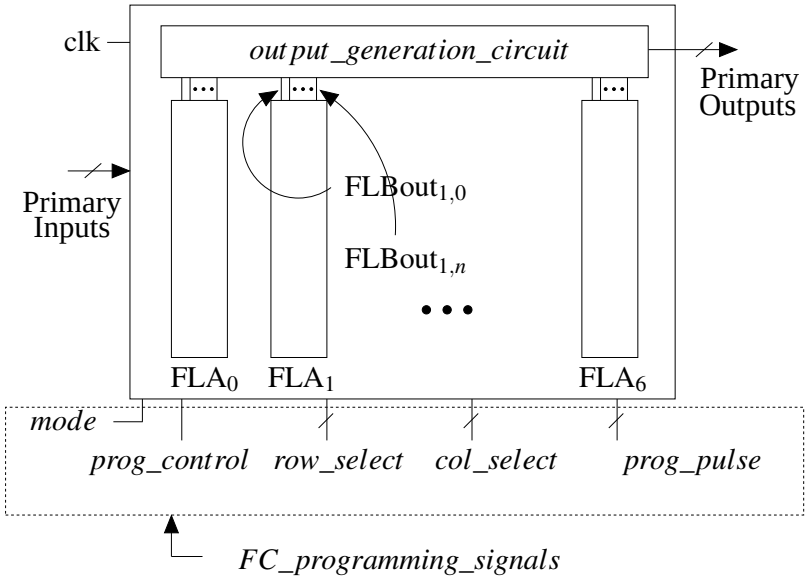


Figure 4.2: Flash Cluster

IV.3.2 Flash Cluster Circuit Design

As mentioned earlier, we construct the flash-based digital block by identifying clusters of nodes in the logic netlist that implement an n-output logic function with up to m inputs ($F_{m,n}$). These clusters of logic will then be implemented using FCs. An FC is a generic circuit structure that can implement any logic function $F_{m,n}$ (with m inputs and n outputs). FCs are also equipped with the required logic for programming the threshold voltages of their constituent floating gate devices. Figure 4.2 shows the block diagram of the FC. As shown in the figure, the FC is driven with a group of signals (for programming purposes) in addition to signals that are used during normal operation. The FC signals used during normal operation are *Primary Inputs* (left side of Figure 4.2) and *Primary Outputs* (right side of Figure 4.2). The additional programming signals are required to program the functionality of the FC. The programming signals of the FC (*mode*, *row_select*, *col_select*, *prog_control* and *prog_pulse*), will be discussed in Section IV.3.3. Since the FC is a dynamic circuit, a clock signal (*clk*) is used for gating the precharge and evaluate transistors in the FC. The *clk* signal is also used during programming, as we will expound in Section IV.3.3.

Figure 4.2 shows the internal components of the FC. The FC is composed of a group of *flash logic arrays* (FLAs) and an *output generation circuit*. An FLA is an array of NAND flash-like pulldown stack structures. Each stack implements a logic cube of $F_{m,n}$. There are $2^n - 1$ FLAs in every FC. Each FLA implements the input cubes that correspond to an output minterm of $F_{m,n}$. For example, if the FC output $< 010 >$ has 2 input cubes

$< 011011 >$ and $< 110 - 11 >$, then FLA_2 implements these two input cubes. The cubes implemented in each FLA do not share minterms with cubes implemented in a different FLA, and hence, exactly one FLA pulls down when any input combination is applied to the FC. We refer to the number of cubes implemented in FLA_i as its *cubes per array* (CPA_i). In the example discussed earlier in this paragraph, $CPA_2 = 2$. The output generation circuit in the FC is driven by the outputs of the FLAs. When the output of FLA_i pulls down, the corresponding output vector (i) is generated at the final output of the FC. For example, if $n = 3$ and if FLA_3 pulls down during evaluation, then the output generation circuit produces the 3-bit output $< 011 >$. Since the FC is a dynamic circuit, its default (precharged) output state is $2^n - 1$ (or $< 111 >$ for $n = 3$). Therefore, we do not need to implement $FLA_{(2^n-1)}$, and hence we only need to implement $2^n - 1$ FLAs (FLA_0, FLA_1, $\cdots$, FLA_6, for $n = 3$, as shown in Figure 4.2).

The circuit details of the FLAs are discussed next.

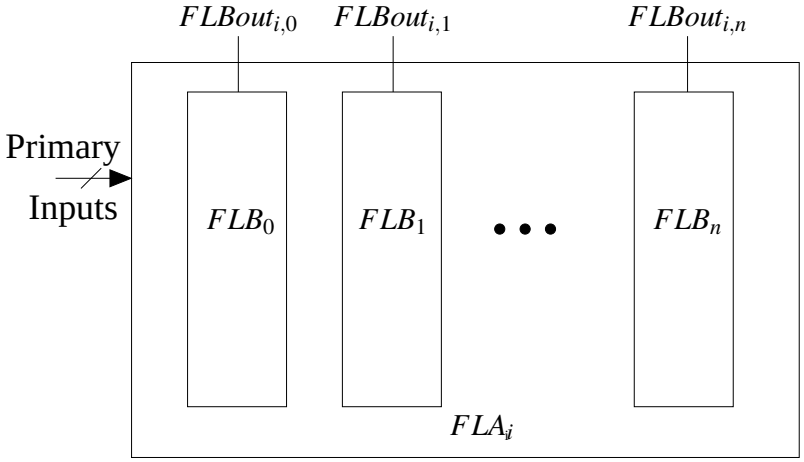


Figure 4.3: Flash Logic Array i (FLA_i) Structure

IV.3.2.1 Flash Logic Array

As mentioned earlier, an FLA comprises of an array of NAND flash-like pulldown stacks, where each one of these stacks implements a cube. The delay, power, and energy of an FLA are degraded as the number of cubes implemented in an FLA increases. Therefore, to maintain healthy delay, power and energy characteristics of the FLA, we split the FLA into a group of *flash logic bundles* (FLBs). Each one of these FLBs can implement a limited number of cubes. We call this number *CPB* and it represents the maximum number of cubes that can be implemented in an FLB. We also refer to the total number of cubes in the i^{th} FLA (FLA_i) as *cubes per array* (CPA_i). Hence, the number of FLBs that exist in FLA_i is $\lceil \frac{CPA_i}{CPB} \rceil$, which is also the number of outputs of FLA_i. Note that while CPA_i is determined by the logic function $F_{m,n}$, *CPB* is a design parameter that can be optimized to improve circuit delay, power, energy and physical area.

IV.3.2.2 Flash Logic Bundles

As mentioned earlier, we limit the number of NAND flash-like pulldown stacks implemented in an FLB to *CPB*, to maintain healthy delay, power and energy characteristics of the FC. Since FLAs can have an arbitrary number of cubes, the actual number of pulldown stacks (cubes) implemented in an FLB is sometimes less than *CPB*. Figure 4.4 shows the circuit details of an FLB.

Each one of the pulldown stacks implemented in an FLB has m flash transistors and 1 regular NMOS transistor (as shown in Figure 4.4), where m is the number of inputs of the function $F_{m,n}$. The flash transistors are programmed to implement cubes, and the

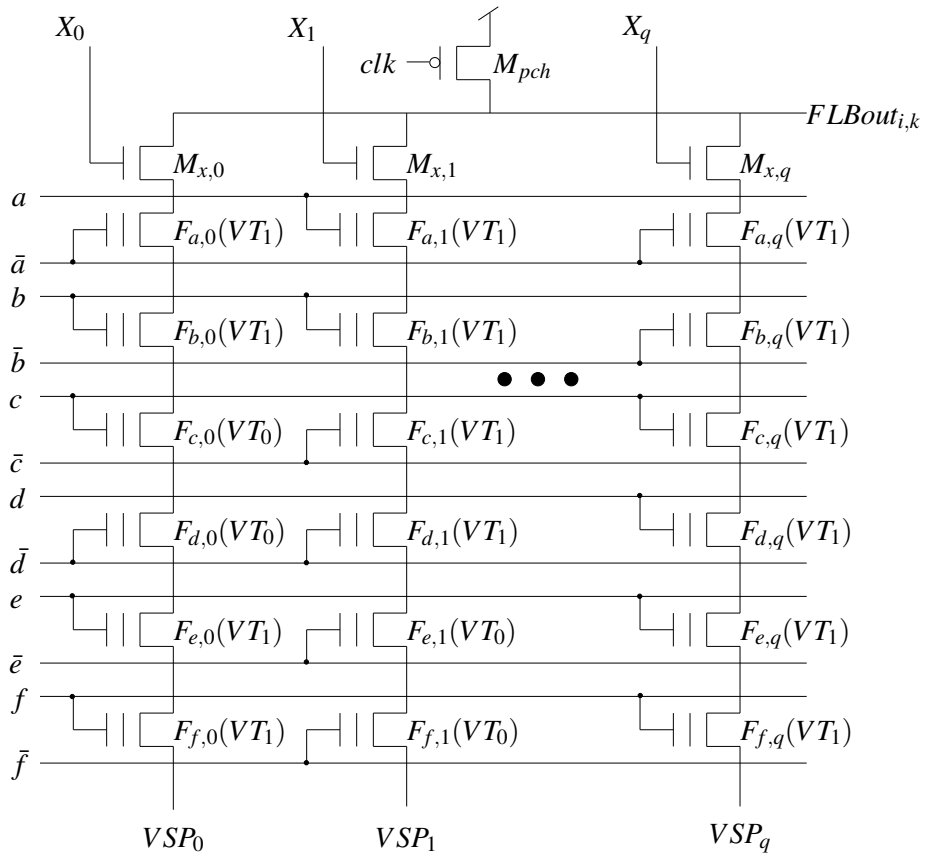


Figure 4.4: Flash Logic Bundle i, k (FLB$_{i,k}$)

regular transistors are used for programming as well as operation purposes. The flash-based implementation proposed in this work is based on dynamic logic, and hence, both a precharge and an evaluate transistor are needed. To precharge the FLB, a PMOS transistor (called M_{pch}) is used. M_{pch} is driven by the clock signal (clk), and hence, is turned on during the precharge (low) cycle of clk. This precharges the output of the k^{th} FLB of FLA_i ($FLBout_{i,k}$) to VDD. The lines VSP_0 to VSP_q are connected to ground during operation to allow the NAND stacks to pull down when they evaluate, but also have a special purpose during programming, which will be discussed in Section IV.3.3. The regular NMOS tran-

sistors ($M_{x,i}$) shown at the top of each stack in Figure 4.4 are used during chip operation as the evaluate transistors which are off during the precharge (low) phase of *clk*. They only turn on during the evaluate (high) phase of *clk*, to allow the pulldown stack to evaluate the output $FLBout_{i,k}$. The NMOS transistors ($M_{x,i}$) are also used for programming purposes. Programming will be discussed in detail in Section IV.3.3.

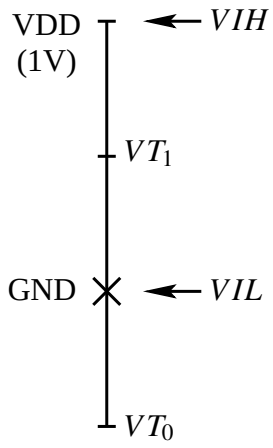


Figure 4.5: Flash Transistor Threshold Voltages Used in an FC

Cubes are implemented in an FLB by programming the flash transistors of each of the NAND flash-like pulldown stacks to implement a cube. Figure 4.5 shows the input voltages applied at the gate of a flash transistor in an FC (V_{IH} and V_{IL}, which are set to VDD and GND respectively). Figure 4.5 also shows the V_T levels that the flash transistors are programmed to. These V_T levels are the *erase* threshold (VT_0) and the *program* threshold (VT_1). Each NAND flash-like stack is configured to implement a cube by programming each of the flash transistor in that stack to VT_0 or VT_1. For example, the left-most stack of

Figure 4.4 implements the cube $\overline{a}bef$. Note that transistors $F_{c,0}$ and $F_{d,0}$ are programmed to a threshold voltage VT_0 which is below GND (see Figure 4.5). Therefore, these two transistors are *on* irrespective of the values of the signals c and d. Now, transistors $F_{a,0}$, $F_{b,0}$, $F_{e,0}$ and $F_{f,0}$ are programmed to a threshold voltage VT_1, which is between VDD and GND (see Figure 4.5). This means that these devices turn on only when their gate signal (respectively $\overline{a}$, b, e and f) are greater than VT_1. As a consequence, the left-most stack of Figure 4.4 implements the cube $\overline{a}bef$.

Similarly, the second stack of Figure 4.4 implements the cube $ab\overline{c}\overline{d}$. The rightmost stack of Figure 4.4 implements the cube $\overline{a}\overline{b}cdef$. Note that all of its transistors are programmed to the VT_1 threshold, which is why it implements a minterm in the m-input space of $F_{m,n}$.

In other words, the design of the FLB resembles the input plane of a NOR-NOR PLA. Unlike a traditional NOR-NOR PLA, the FLB is different because the cubes that it implements are not shared between other FLBs or FLAs. This results in the FC needing to implement more cubes in general than in a NOR-NOR PLA. However, based on our initial experiments, the NOR-NOR PLA is less efficient from a delay, power and energy point of view, since the output logic may have a larger loading (diffusion capacitance) caused by cubes being shared across outputs.

IV.3.2.3 Output Logic

When an input value is applied to the FC, exactly one of the outputs of the FLAs pulls down. After this, the corresponding output state are generated using the output generation

circuit. Figure 4.6 shows the circuit details of the output generation circuit. Each output of the function $F_{m,n}$ is driven by an output buffer (sized to drive a fan-out of 3 FC input loads). The unbuffered outputs of the output logic are represented using a horizontal line which is precharged by a PMOS transistor (shown at the left side of the output line). The precharge PMOS transistor is driven by the clock signal (clk). Each of the unbuffered output lines is pulled down based on which FLA_i output ($FLBout_{i,k}$) is pulled down. Note that exactly one $FLBout_{i,k}$ pulls down for any input to the FC. For each output minterm in the function $F_{m,n}$, if the output j is 0, an NMOS transistor is inserted for that output line in the output logic. Otherwise, the output line will by default stay high after the precharge cycle. Since the outputs of the FLAs are active low, we insert an inverter for each $FLBout_{i,k}$ before driving the gate of the pulldown NMOS devices in the output logic.

For example, if any of $FLBout_{0,i}$ pull down, then all three f, g and h will pull down. If any of $FLBout_{3,i}$ pull down, on the other hand, then the output will be $< f,g,h >$ = $< 011 >$, assuming that f is the most significant output bit. In this case, there will be NMOS devices pulling down the output of f in the output logic, and no NMOS devices connected to g and h, which will stay precharged, resulting in the output minterm $< f,g,h >$ = $< 011 >$ being produced. Finally, the output minterm $< 111 >$ does not need to be produced, and therefore all input cubes mapping to the output minterm $< 111 >$ are never implemented.

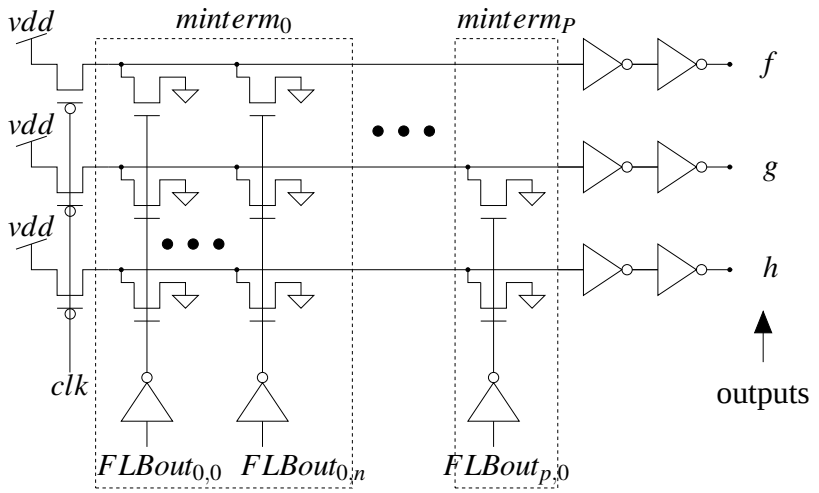


Figure 4.6: Flash Output Generation Circuit

IV.3.3 Programming the Flash Cluster

The programming (and erasure) of the flash transistor in the FC is similar to the programming of the flash transistors in the TLC, which was discussed earlier in Section III.3.4. The main difference in the programming of the FC is that we are using dual purpose signals VSP_i (shown in Figure 4.4), instead of adding an extra NMOS transistor at the bottom of the flash stack (as discussed in Section III.3.4). We will go over the erase and programming of the FC only to show how the VSP_i signal is used in the erasure (and programming) of the FC, however, all the remaining steps of the erasure and programming of the FC is similar to that of the TLC (covered in Section III.3.4).

The flash transistors in an FC share a common bulk, and hence, all the flash transistors are erased at once. Consider Figure 4.4. The erase operation is performed by applying a high voltage (10V-20V) to the bulk node of the FC, and floating the source and drain

terminals of each flash transistor (by turning off the $M_{x,i}$ transistor and floating the signal VSP_i). The gates of all transistors are driven to GND. These conditions are applied long enough to guarantee that the V_T levels of all the flash transistors in the FC have reached VT_0 (the erase V_T level), as shown in Figure 4.5.

Before starting the programming of an FC, we always perform an erase operation, in order to reset the V_T's of the flash devices to VT_0, and then only program the the flash transistors to VT_1 according to the configuration procedure described in the FLB part of Section IV.3.2.2. For example, assume that $F_{c,1}$ and $F_{c,q}$ in Figure 4.4 need to be programmed to VT_1. In this case, the c and $\overline{c}$ lines are driven to a programming voltage. The transistors and $M_{x,1}$ and $M_{x,q}$ are turned on by driving X_1 and X_q high), while leaving the other X_i low. Also, the lines VSP_1 and VSP_q are driven low and the remaining VSP_i lines are floated. This disables programming of all but the 2^{nd} and q^{th} NAND stacks of the FLB. All inputs other than c and $\overline{c}$ (i.e. a, b, d, e and f and their complements) are driven to a pass voltage. After applying the programming pulses for a sufficient duration, the threshold voltages of $F_{c,1}$ and $F_{c,q}$ are programmed to the VT_1 threshold voltage, while all other transistors in the FLB are unaltered and stay at the erase threshold voltage (VT_0). Programming of an FLB requires the application of a maximum of 6 programming pulses (since $m = 6$ in our design). Note that by controlling the duration of the programming pulse, the value of the threshold voltage VT_1 can be adjusted with a fine granularity.

The *mode*, *prog_control*, *row_select*, *col_select* and *prog_pulse* signals in an FC (shown in Figure 4.2) have the same functionality as those used in a TLC and are described

in Section III.3.4.

It is important to note that although high voltages are required for programming the flash transistors, we restrict operating voltages to 1V, which is the nominal supply voltage for 45nm CMOS technology node.

IV.3.4 Read and Write Disturb

One of the issues of using flash transistors in NAND flash memories is read and write disturb. Read and write disturbs were described earlier in Section I.9.4. In our flash-based design approach, we suppress the issue of read and write disturbs by:

- Using SLC cells only, which have exponentially higher immunity to read and write disturbs [43].
- Limiting the number of flash transistors in series in our structure to 6 flash transistors (compared to 100s of flash transistors in NAND flash memories).
- Limit the operating supply voltage in our implementation to 1V, which results in reduced electric fields, thus drastically reducing read disturbs to adjacent flash transistors.
- Unlike NAND flash memories, our flash-based design approach does not require the use of a passing voltage (which is higher than the read voltage) during regular operation (i.e. we are always reading all the flash transistors in the same series stack).

Output minterms	0	1	2	3	4	5	6	7	Total
No. input minterms	11	6	8	9	9	6	3	12	64
No. input cubes	8	3	5	4	6	6	3	N/A	35

Table 4.1: Example of Input Minterm and Cube Distribution of an m-input and n-output Logic Function

IV.3.5 Mapping a CMOS-based Design into a Flash-based Design

We start the conversion of each FC (A, B, C, D and E of Figure 4.1) from CMOS into flash by constructing a table of all the 2^n output minterms and their corresponding input minterms. Now each of the input minterms for each output minterm are minimized separately using Espresso [29]. The total number of input minterms for the n-output function represented by the FC of interest (we call this function $F_{m,n}$) is 2^m. This enumeration is inexpensive since m and n are small (6 and 3 respectively in our work).

Table 4.1 lists the output minterms (in row 1) for a representative function $F_{m,n}$ (with m=6 and n=3). The number of input minterms for each output minterm are shown in row 2. The resulting number of cubes for each output minterm (after running Espresso) are shown in row 3. The output minterms corresponding to the '7' output are not implemented, since the FC is a precharged circuit.

IV.4 Experiments

In this section, we first present the simulation environment used in evaluating our flash-based digital circuit design approach. Then we discuss the flash-based digital cir-

cuit implementation details. Finally, we present the details of our experiments and the discussion of the results.

IV.4.1 Simulation Environment

Our FC-based design approach (presented in this chapter) is compared to a CMOS standard cell based design approach. We used a 45nm process technology to implement both the designs. For the CMOS standard cell based implementation, the digital designs were synthesized and mapped to the industry grade 45nm Nangate FreePDK45 Open Cell Library [54, 55] using Synopsys Design Compiler [56]. The mapped designs were simulated at the circuit level using the Synopsys HSPICE [56] circuit simulation tool and the 45nm PTM [57] card (nominal VDD is 1V). For the FC-based designs, we used our in-house tool-chain to generate the FCs representing the same digital designs as those implemented using CMOS. The flash-based circuits were back annotated with layout parasitics, and then simulated using HSPICE [56]. Both of the CMOS-based and the flash-based designs operate at VDD of 1V. However, the flash-based designs use 10V-20V for programming purposes only.

For the flash devices, we follow the same technique found in [64] to model the gate capacitance of the flash devices and derive our flash model card, which was also described in Section III.4.2. The only difference is that FCs only use two V_T levels while the TLC (in Section III.4.2) used three V_T levels. We verified the correct logical operation of the flash-based digital circuit through exhaustive simulations, and generated custom layouts for the flash-based digital circuits using Cadence Virtuoso [65]. We used the generated

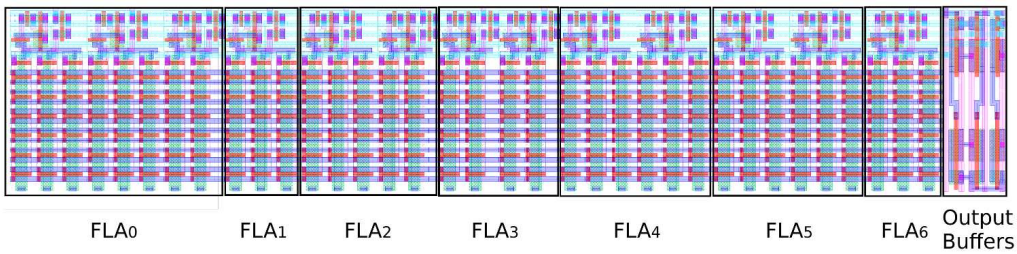


Figure 4.7: Layout View of an FC (des00)

layouts of the flash-based designs to compare the physical area of the flash-based digital circuits to their CMOS-based counterparts. We generated 20 *randomly generated* circuit designs to evaluate our flash-based digital circuit design approach. The layout of our FCs used design rules for flash devices that were obtained from the ITRS reports [59].

IV.4.2 Flash-based Implementation Details

We implemented logic functions with 6 inputs (m = 6) and 3 outputs (n = 3) using both of the flash-based approach described in this chapter as well as the CMOS standard cell based approach. For the flash-based designs, the FCs are implemented using FLBs of size 3 (CPB = 3), and the target threshold voltages used are VT_0 = -0.5 V and VT_1 = 0.5 V.

IV.4.3 Results and Analysis

Table 4.2 reports the delay, power, energy and physical area results ratios of our flash-based designs compared to CMOS standard cell based implementation. The delay reported in the table (D_{max} Ratio) is maximum delay of any transition seen at any primary output of the circuit. Since the flash-based implementation is dynamic, we accounted for the precharge delay in the reported delay shown in the table. The precharge delay is about

Circuit	D_{max} Ratio	P_{avg} Ratio	Eng Ratio	Cell Area Ratio
des00	0.81×	0.34×	0.28×	0.50×
des01	0.75×	0.31×	0.24×	0.50×
des02	0.81×	0.35×	0.28×	0.59×
des03	0.74×	0.39×	0.28×	0.51×
des04	0.89×	0.38×	0.34×	0.62×
des05	0.71×	0.33×	0.23×	0.48×
des06	1.04×	0.34×	0.35×	0.58×
des07	0.83×	0.36×	0.30×	0.58×
des08	0.80×	0.35×	0.28×	0.56×
des09	0.87×	0.31×	0.27×	0.49×
des10	0.93×	0.38×	0.35×	0.54×
des11	0.87×	0.40×	0.35×	0.50×
des12	0.92×	0.38×	0.35×	0.53×
des13	0.89×	0.38×	0.34×	0.58×
des14	0.80×	0.33×	0.26×	0.51×
des15	1.01×	0.40×	0.40×	0.53×
des16	0.88×	0.34×	0.30×	0.59×
des17	0.77×	0.34×	0.27×	0.56×
des18	0.83×	0.34×	0.28×	0.55×
des19	0.69×	0.36×	0.24×	0.52×
Average	0.84×	0.35×	0.30×	0.54×
Stdev	0.09×	0.03×	0.05×	0.04×

Table 4.2: Delay, Power, Energy and Cell Area Ratios of Flash-based Digital Circuits Relative to Their CMOS Standard-cell Based Counterparts

25% of the total delay. In most digital circuits, the delay path consists of multiple levels of logic (about 5 levels of logic in recent designs). Since we only need to precharge once then evaluate the 5 logic levels, the total delay of the logic path becomes (5× evaluate delay + 1× precharge delay). This will result in further reducing the D_{max} ratio (by about 20%) compared to the numbers reported in the table. As shown in the table, the delay of the flash-based digital circuits ranges from 0.69× to 1.04× of the CMOS standard cell-

based digital circuit delay, with an average of 0.84×. The standard deviation of the relative flash-based design results is shown in Table 4.2. The standard deviation of the the delay (0.09× of CMOS), power (0.03× of CMOS), energy (0.05× of CMOS) and cell area ratios (0.04× of CMOS) indicate that the FC has predictable characteristics and in general will outperform CMOS designs in all the design metrics.

The key reasons for the reduced delay are:

- Lowered gate capacitance of the flash FET (20× lower than a MOSFET), as explained in Section III.4.2.
- The increased parasitics of the standard cells (due to the use of NMOS as well as PMOS devices which are both driven by the inputs) causes higher delays for the CMOS standard-cell implementation.
- The use of shared (un-contacted) diffusions in the NAND stack reduces parasitics significantly, thus reducing delays in the flash-based circuits.
- Our design is dynamic while the CMOS standard cell-based design is static. This typically yields a 15-20% lower delay.

We also report the power dissipation (average of 0.35× of CMOS) when implementing the digital circuits using our flash-based logic compared to CMOS standard cell-based implementation. We also show the energy utilization of our flash-based implementation compared to the CMOS standard cell-based implementation. On average, the energy utilization of flash-based digital circuits is about 0.3× of the CMOS standard cell-based implementation.

It is well known that dynamic designs consume more power than static CMOS designs. Our FC based design consumes less power for several reasons. Despite being dynamic, the number of nodes being precharged is smaller than a CMOS (domino or other dynamic) approach. Further, the long transistor stacks (since we choose $m = 6$) result in smaller evaluation currents, reducing power. Also, in our design, exactly one FLB pulls down during every evaluation, reducing switching activity and hence power consumption. Finally, the I_{ds} of a flash FET is lower than that of a MOSFET, which results in a lower power consumption.

We also report the area ratio of both implementations. The area reported for the CMOS standard cell-based implementation is the sum of physical cell areas, while the area of our flash-based approach is the layout area obtained from layout generation experiments. We expect the area ratio to be more favorable in practice, when CMOS wiring areas are taken into account. Design rules for flash were obtained from the ITRS 45nm flash technology node [59]. Digital circuits implemented in an FC use $0.54\times$ the physical area of a CMOS-based design, on average. In Figure 4.7, we show the representative layout of a cluster of the FC for the design des00. Note that the FC shown in Figure 4.7 does not implement FLA_7, since FLA_7 is implemented by the precharge state.

The ability to change threshold voltages after fabrication in flash-based designs enables adjusting the design's speed to compensate for circuit aging. To show the benefits of this, we performed a 10000 point Monte Carlo simulation to model process variation (W and L) with a 1-sigma of 5% of the nominal parameter value. Figure 4.8 shows histograms

of the maximum delay (D_{max}) of a CMOS design (labeled as "CMOS"), a flash-based design programmed with nominal V_T (labeled as "flash (nominal)"), and the same flash-based design subsequently programmed with a lower V_T (labeled as "flash (fast)"). The lower V_T value was 50 mV lower than the nominal value of VT_1. The figure shows that the delay of the flash-based design programmed with lower V_T (labeled as "flash (fast)") is shifted to the left. This indicates that in flash-based designs, in-field compensation of aging effects can be achieved by programming the flash-based design to a lower V_T to decrease the delays as desired.

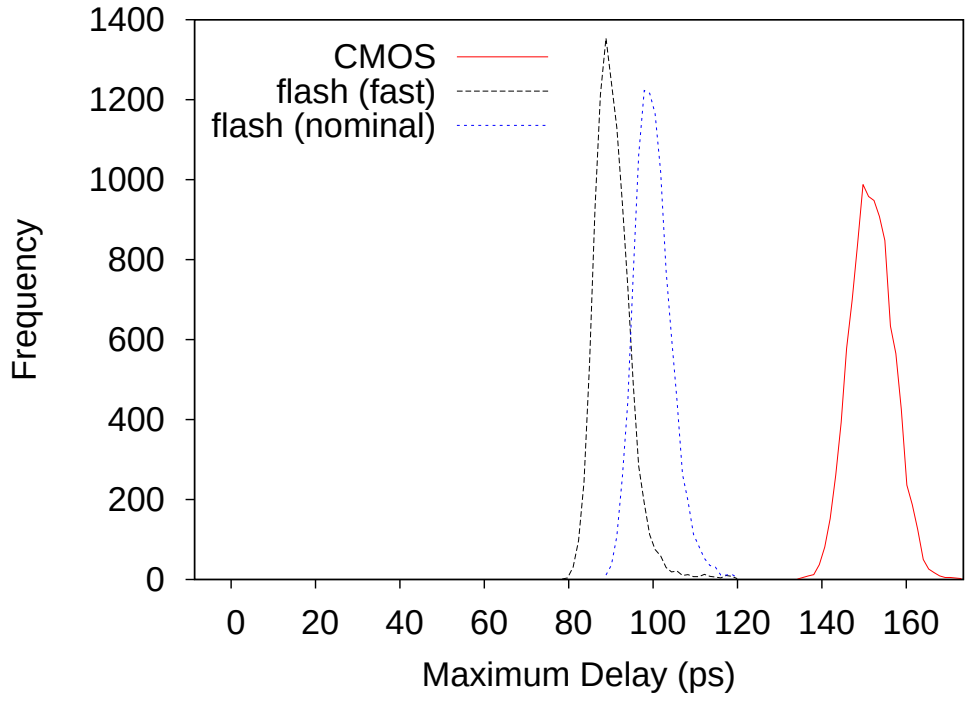


Figure 4.8: Histograms of CMOS and Flash-based Designs (Before and After Aging Compensation)

Flash-based digital circuits have the ability of tuning their delay, power and energy characteristics. This is done through shifting the V_T of the flash transistors in the circuit. The ability to shift V_T offers the flash-based digital circuits huge advantages over the traditional CMOS standard-cell based circuits when it comes to speed binning at the factory, mitigation of circuit aging and performing post-manufacturing ECOs.

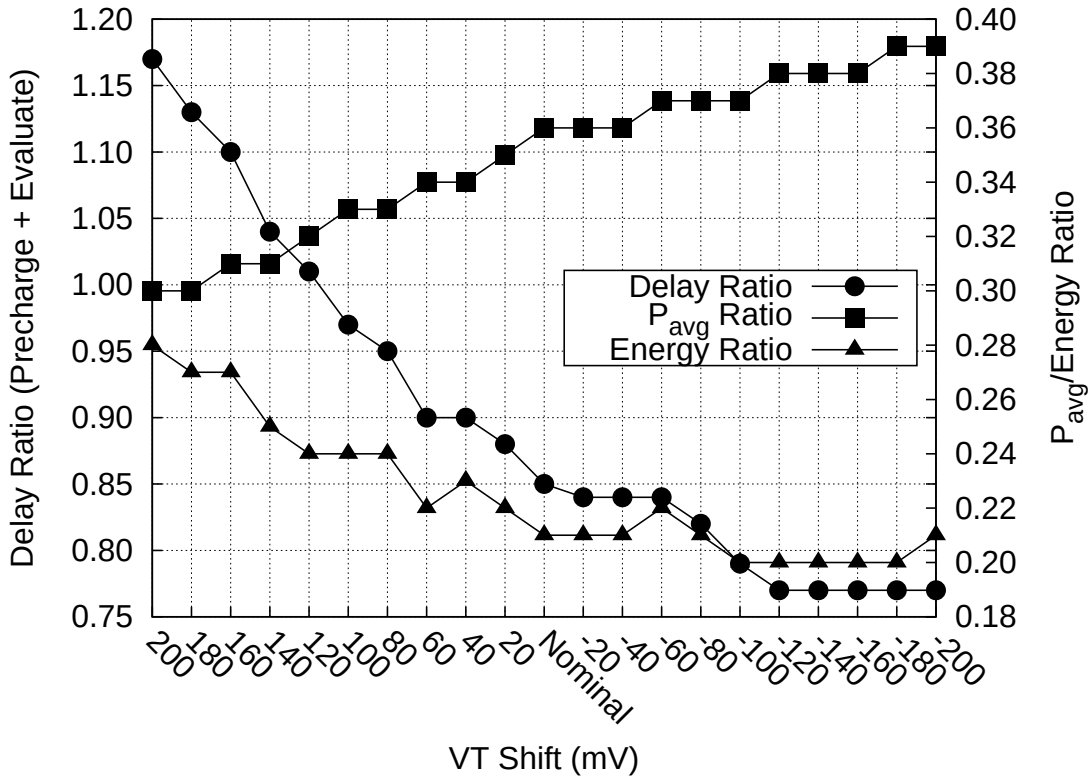


Figure 4.9: Delay, Power and Energy of the Flash-based Designs as V_T is Shifted.

Figure 4.9 demonstrates the ability of tuning the delay, power and energy of the flash-based circuits by shifting the threshold voltage of the flash transistors in the flash-

based circuits. The x-axis in the figure shows the V_T shift in mV, where the nominal V_T is chosen at the V_T values described in Section IV.4.1. The left y-axis shows the delay ratio of the flash-based design compared to the CMOS standard-cell based design as the V_T of the flash transistors is shifted. The delay of the flash-based design is the sum of the precharge and the evaluate delays. The right y-axis shows the average power and energy ratios of the flash-based designs compared to the standard-cell based designs as the V_T of the flash transistors is shifted. The delay, power and energy shown in Figure 4.9 are averaged across all the benchmark designs shown in Table 4.2. Figure 4.9 shows that by shifting the V_T of the flash transistors to a lower value than its nominal value, the delay of the design decreases and the power dissipated increases, while the energy consumption is decreased. Conversely, when the V_T is shifted to a higher value than the nominal V_T value, the delay increases and the power dissipated decreases, while the energy consumption is increased. These results confirm our ability to control the circuit delay, power and energy characteristics by fine tuning the threshold voltage of the flash-based design. This allows the manufacturer to do precise speed/power binning of parts in the factory.

The delay, power and energy of fabricated digital ICs vary broadly due to process variations. We study this effect by simulating our flash-based designs at different process corners. The process corners that are modeled in our work are shown in Table 4.3. We also show the effect of shifting the V_T of the flash devices by +/- 50 mV. The delay ratios of our flash-based designs are shown in Figure 4.10, the power ratios are shown in Figure 4.11 and the energy ratios are shown in Figure 4.12. The *x*-axis of in these figures show the

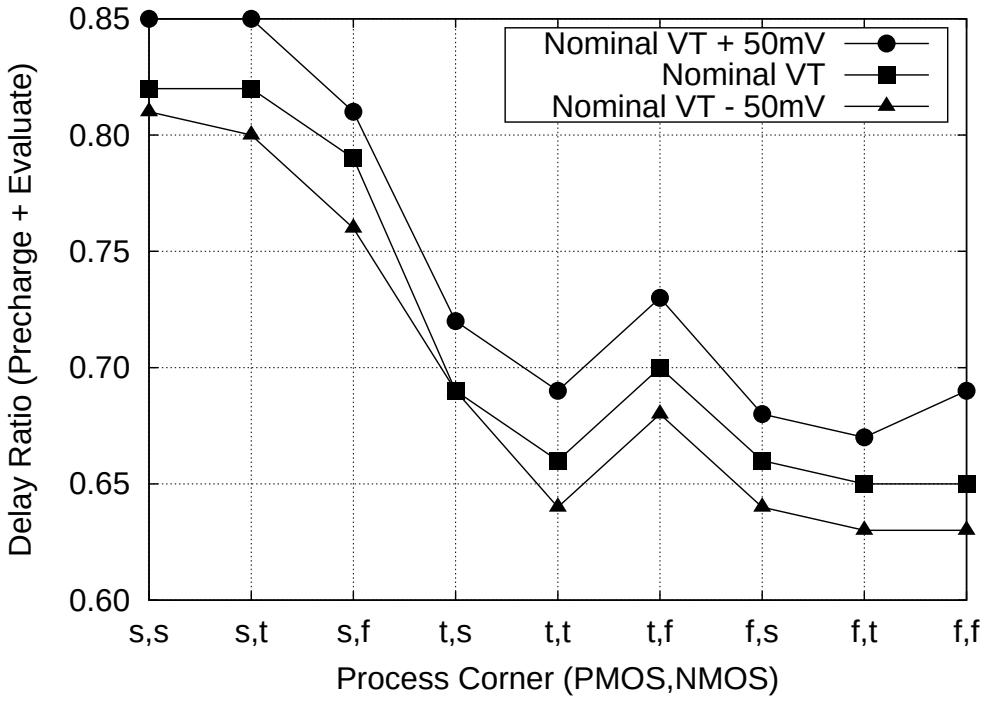


Figure 4.10: Delay of Flash-based Designs at Different Process Corners and V_T Levels.

process corner and the *y*-axis shows the average delay ratio (in Figure 4.10), average power ratio (in Figure 4.11) and average energy ratio (in Figure 4.12) of the flash-based designs compared to their CMOS standard cell counterparts. As shown in the figures, shifting the V_T of the flash devices after fabrication reduces the effect of process variations on the delay, power and energy of the flash-based designs.

IV.5 Chapter Summary

The device structure of flash transistors has made them the technology of choice for implementing non-volatile memory. This chapter presented an approach to use flash

Process corner	PMOS	NMOS
s,s	Slow	Slow
s,t	Slow	Typical
s,f	Slow	Fast
t,s	Typical	Slow
t,t	Typical	Typical
t,f	Typical	Fast
f,s	Fast	Slow
f,t	Fast	Typical
f,f	Fast	Fast

Table 4.3: Process Corners

transistors to implement digital logic circuits. The threshold voltage of flash devices can be modified at a fine granularity during programming, which results in several advantages such as controlling the speed/power binning of integrated circuits, aging mitigation as ICs slow down over the years and performing ECOs. We present the details of the circuit topology that we use in our flash-based digital circuit approach. Our HSPICE simulations show that, averaged over 20 designs, our approach yields 0.84× the delay, 0.35× the power, 0.3× the energy utilization and 0.54× the physical area of the equivalent circuit implemented using CMOS standard cell-based design.

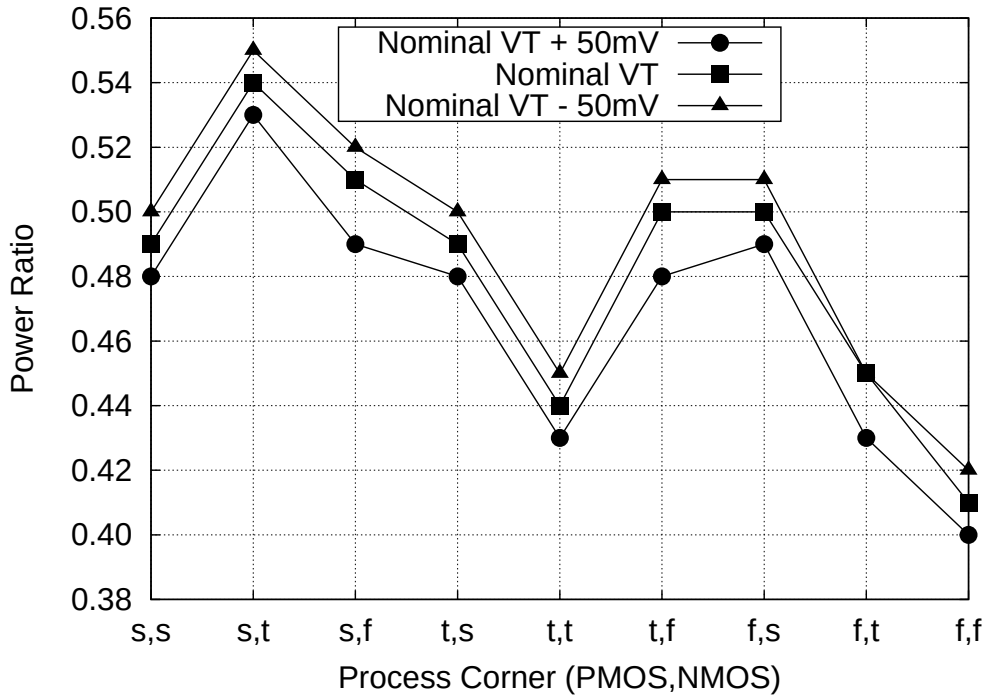


Figure 4.11: Power of Flash-based Designs at Different Process Corners and V_T Levels.

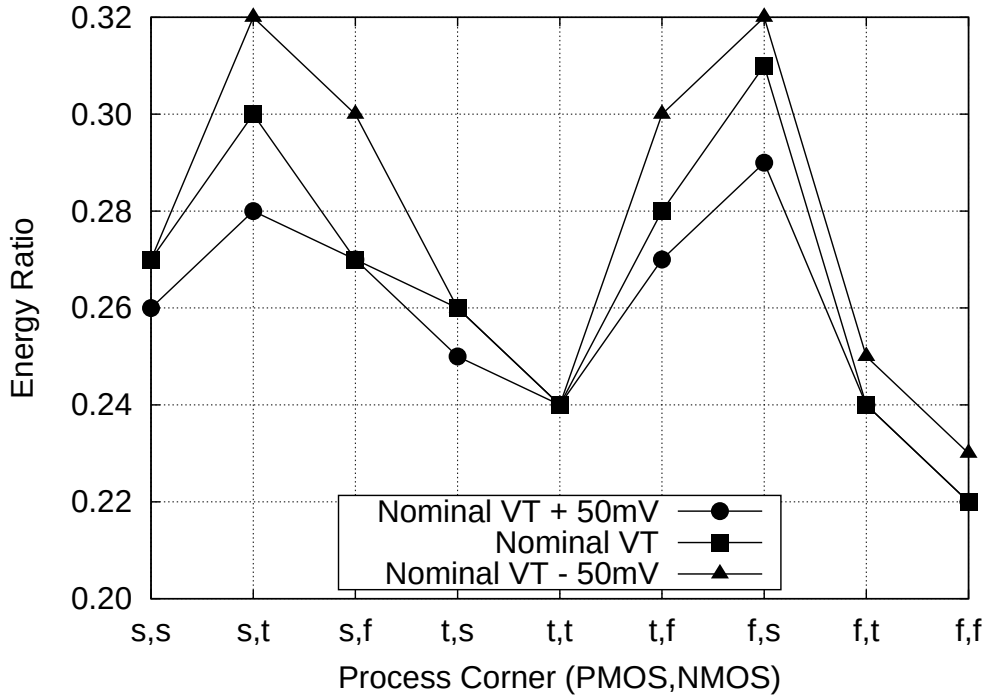


Figure 4.12: Energy of Flash-based Designs at Different Process Corners and V_T Levels.

CHAPTER V

BLOCK-LEVEL IMPLEMENTATION OF BINARY-VALUED, FLASH-BASED DIGITAL DESIGNS[1]

In this chapter, we present the details on the realization of the block-level flash-based digital design. The current chapter describes and characterizes the CAD flow to decompose a circuit block into a network of interconnected FCs.

V.1 Background

In Chapter IV, we described the use of flash transistors to implement binary-valued digital circuits. In Chapter IV, we exploit the ability to control the threshold voltage of flash transistors, to implement digital circuits. The circuit topology utilized was a cluster of unprogrammed flash transistors arranged in a NAND configuration (we call these *Flash Clusters (FCs)*), which are programmed in the factory to implement the desired logic function. There are several ways in which the current work differs from Chapter IV:

- The work of Chapter IV only describes the design, electrical details and circuit-level characterization results for an FC. An FC implements a logic function of a small number of inputs (up to 6 inputs in Chapter IV) and outputs (up to 3 outputs in Chapter IV). In contrast, the current work focuses on the design of large circuit blocks which are comprised of 1000s of interconnected FCs.

[1]Part of the data reported in this chapter is reprinted with permission from "A Flash-based Digital Circuit Design Flow" by Monther Abusultan and Sunil P. Khatri, International Conference On Computer Aided Design (ICCAD) 2016, pp. 1-6, Copyright 2016 by ACM.

- The work of Chapter IV does not cover the decomposition of a large circuit block into a network of FCs, and the electrical characterization of the resulting block. The current work focuses its attention on this aspect, covering the synthesis, mapping and electrical characterization of a large circuit block which is implemented using an interconnected network of FCs.
- The work of Chapter IV compared the electrical characteristics of several randomly generated FCs (up to 6-input, up to 3-output functions) with a standard-cell based implementation of the same function. This work, in contrast, compares the electrical characteristics of several large designs implemented using (1000s of) FCs, with a standard-cell based implementation (realized using commercial tools) of the same design.
- The relationship between Chapter IV and the current work is similar to the relationship between standard-cells and a standard-cell based design flow. One can say that Chapter IV describes the design of ”the flash standard-cell” in detail, while this work describes the entire design flow involved in synthesizing, mapping and characterizing an entire circuit block using an interconnected network of ”flash standard cells”.
- This work, in a sense ”closes the loop” and describes how the FCs of Chapter IV perform when they are used to implement a large digital circuit block.

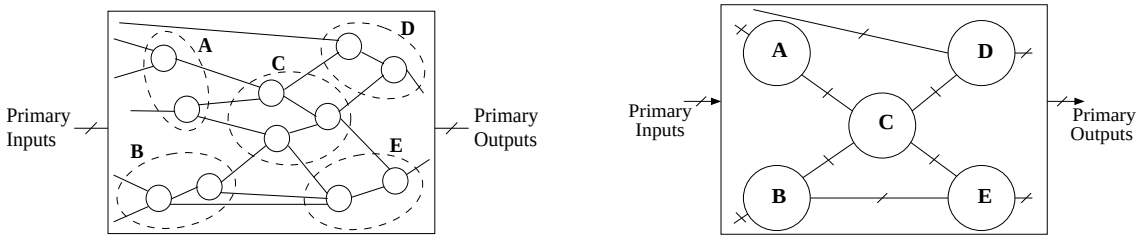
V.2 Previous Work

To the best of our knowledge, there has been no work prior to this work, which describes the synthesis, mapping and electrical characterization of digital circuits implemented as a network of flash-based circuit elements (FCs).

V.3 Approach

In this section, we present a design flow that implements digital circuits using flash transistors. However, unlike Chapter III and Chapter IV, which deal with approaches to realize flash-based designs at the cell-level, this Chapter shows how to implement flash-based designs at the block-level. In this Chapter, we use the flash cell presented in Chapter IV, called the FC, as the cell structure used to implement the entire block-level design. This choice is mainly due to the superior delay, power, energy and area results that the FC has demonstrated at the cell-level.

V.3.1 Overview



(a) Logic Netlist (b) Flash-based Digital Circuit

Figure 5.1: Converting a Logic Netlist into a Flash-based Design

The flash-based design flow entails the conversion of a technology-independent digital circuit into a flash-based digital design. Since the flow presented in this chapter uses the FC (from Chapter IV), we will only cover the algorithmic details of implementing digital design using an interconnected network of FCs. Nonetheless, familiarity with the circuit structure of the FC is very important to understand the design flow of the flash-based designs. the reader is referred to Section IV.3.2 to read about the circuit details of the FC.

V.3.2 Flash-based Design Conversion

We have conceptually discussed the top level flow that we use to convert a technology-independent digital circuit into a flash-based design (using the FC) in Chapter IV. In this section, we will briefly cover the top-level conversion process to familiarize the readers with the conversion process and give them the necessary background to understand the flash-based design flow presented in this chapter.

We illustrate the conversion process in Figure 5.1. Figure 5.1(a) shows a technology-independent logic netlist. The first step in our flow is to group the nodes in the logic netlist to form multi-input (m inputs) and multi-output (n outputs) clusters (illustrated by the dotted ovals in Figure 5.1(a)). These clusters are then implemented using FCs which are shown as solid circles in Figure 5.1(b). In other words, each FC implements an up to n-output logic function of up to m inputs. We refer to this function as $F_{m,n}$. Note that the solid circles labeled A, B, C, D and E in the flash-based digital circuit of Figure 5.1(b) have the same functionality and connectivity as the dashed ovals A, B, C, D and E in the

technology independent digital circuit of Figure 5.1(a).

This chapter focuses on flash-based implementation of a digital design using an interconnected network of FCs. We describe the synthesis, mapping and electrical characterization of the resulting design, and compare the delay, area, power and energy with a CMOS standard-cell based realization of the same design (obtained using commercial tools). It is very important to understand the structure of the FC in order to fully understand our CAD flow. The reader is encouraged to read Chapter IV, more specifically Section IV.3.2, Section IV.3.5 and Section IV.4, towards this goal.

V.3.3 FC-based CAD Flow Overview

The CAD flow to convert an input logic netlist is described next. The input logic netlist is technology independent in our experiments, but it could be technology dependent as well. There are several steps in the flow, which are briefly described next, and then explained in detail.

First, the input netlist is clustered into FCs (where FC_i implements $F^i_{m,n}$), with a goal of minimizing the wiring between FC's. In our experiments, $m \leq 6$ and $n \leq 3$. After this, we obtain a multi-level netlist of interconnected FCs.

Next, the layout of each FC is generated. The FCs, FLAs and FLBs are extremely regular in their physical characteristics, making them amenable to the on-the-fly physical synthesis flow that we use. Based on the fanout load of the i^{th} output of FC_j, additional buffers are added for that output.

To quantify the utility of our flash-based circuit design flow, the same input netlist is

synthesized and mapped using commercial standard-cell based CAD tools. The resulting designs (flash-based and standard-cell based) are compared in terms of their delay, area, power and energy, over a number of designs.

V.3.3.1 FC-based Clustering

Problem Definition: Given an arbitrary logic netlist η, cluster η into a multi-level network η^* of FCs, subject to the following constraints:

- The network η^* is acyclic.
- Each $FC_i \in \eta^*$ has a logic function $F^i_{s,t}$ where $s \leq m$ and $t \leq n$.

Algorithm 1 Clustering a Logic Netlist into a Multi-level Network of FCs

```
η = decompose_network(η, p)
L = dfs_and_levelize_nodes(η)
FC* = 0
η* = 0
while get_next_element(L) != NIL do
   FC* = FC* ∪ get_next_element(L)
   if (num_input(FC*) ≤ m) && (num_output(FC*) ≤ n) then
      continue
   else
      Q = remove_last_element(FC*)
      η* = η* ∪ FC*
      FC* = Q
   end if
end while
η* = wiring_recovery(η*)
```

Algorithm 1 outlines our clustering strategy. We first decompose η into an equivalent network of nodes, with at most p inputs. If this were not done, we could encounter a situation where the number of inputs to some node in η is greater than m, making it

impossible to create the multi-level FC-based netlist. We choose $p < m$, and in particular we found that $p = 3$ yielded good results. Now η is sorted in a depth-first manner. The resulting array of nodes is sorted in *topological*[2] order, and placed into an array L.

Now we greedily construct the logic in each FC, by successively grouping nodes from L such that the resulting implementation of the grouped nodes FC^* does not violate the input or output cardinality constraints for the FCs. If so, we attempt to include another node into FC^*, otherwise we append the last FC satisfying the height and width constraints to the result η^*.

In order to reduce the wiring between FCs, the *get_next_element* routine preferentially returns nodes in the fanout of the nodes of FC^*, provided that the inclusion of such a node into FC^* would not result in a cyclic dependency between the FCs of η^*. If such nodes are not available, the first un-mapped node from L is returned. At every step of the construction of η^*, we verify that the graph induced by the multi-level network of FCs is acyclic.

After the clustering step is completed, we invoke a procedure called *wiring_recovery*. This is a final effort in reducing the wiring between FCs. This procedure attempts to move individual nodes in L to a different FC than their currently assigned FC. If a wiring gain is realized by such a move, the move is made. If no more nodes can be gainfully moved, or if a specified number of iterations have been made through L, the procedure returns. On average, the *wiring_recovery* procedure is able to reduce wiring by

[2]Primary inputs are assigned a level 0, and other nodes are assigned a level which is one larger than the maximum level of all their fanins

about 9.6%. We note the following about this procedure:

- It is possible that a node n in L is the only node in some FC X, and if n can be moved to another FC, then FC X can be eliminated from η^*. We came across a few instances where a FC was removed in this manner.
- *wiring_recovery* returns when no node can be moved without increasing the wiring cost of the multi-level network of FCs. At this point, it is still possible that more than one node can simultaneously be moved to realize a gain in wiring. However, this condition is not checked.

The functional correctness of the resulting multi-level network of FCs was verified at the end of the clustering step.

V.3.3.2 On-the-fly Layout Synthesis

Once the multi-level netlist of FCs is generated in the previous step, we next generate the layout for each $FC^i \in \eta^*$. First, for each FC^i, we construct a table of all the 2^n output minterms o_p and their corresponding input cubes $C_p = \Sigma c_{p,q}$. This construction is inexpensive in practice, since m and n are small (6 and 3 respectively in our experiments). The set of cubes $\{C_p\}$ form a partition of the points in B^m, where $B = \{0,1\}$. This table is constructed from the truth table of $F^i_{m,n}$, simply by grouping all the input minterms for each output minterm. Now the input minterms for each output minterm are minimized using Espresso [29]. The output minterm which has the largest number of cubes is not implemented, and is mapped to the default output of the FC when it is precharged, as discussed in Section IV.3.5.

Table 5.1, shows the number of input minterms (and cubes) that correspond to each output minterm for a representative function $F_{m,n}$ with $m = 6$ and $n = 3$. The cubes corresponding to the '7' output are not implemented, since the number of cubes for this output is the largest, and can be mapped to the default output of the FC, since it is a precharged circuit.

Output minterm	0	1	2	3	4	5	6	7	Total
# Input minterms	8	5	8	11	6	7	7	12	64
# Input cubes	8	3	5	4	6	6	3	9	44

Table 5.1: Example of Minterm Distribution of an n-output Logic Function with m Inputs

For each $FC^i \in \eta^*$, our layout synthesis algorithm adds larger output buffers for output x whenever the fanout load (measured in terms of the total number of pulldown stack devices that x drives) exceeds a particular value. We chose this threshold to be 96.

V.4 Experiments

V.4.1 Simulation Environment

In this section we will discuss the methodology we used in reporting the results obtained through our flash-based design flow compared to the results obtained from an equivalent CMOS standard cell based design flow. The designs presented in this thesis are implemented in a 45nm process technology. This is because an industry grade CMOS standard cell library in 45nm technology is easily obtained and serves as a realistic candi-

date to compare our flash-based design flow with. The CMOS standard cell based digital circuits are synthesized and mapped using a 45nm Nangate FreePDK45 Open Cell Library [54, 55] using Synopsys Design Compiler [56]. The delay, power and area of the CMOS standard cell based digital circuits are extracted using Design Compiler. The flash-based digital circuits were generated using custom scripts. For CMOS devices, we used a 45nm PTM process [57]. For the flash devices, we derived the 45nm flash device models from the measurements results presented in [46] and validated our models using [58, 45]. The details regarding our model card regression was discussed earlier in Section III.4.2. However, we only use two V_T's for the flash transistors (as described in Section IV.4, since the design flow described in this chapter uses the FC as the building cell of the flash-based design. The target programmed threshold voltages used in our designs are (VT_0 = -0.5 V) and (VT_1 = 0.5 V). We simulated the flash-based FCs in HSPICE and also verified the correct logical operation of the flash-based digital circuit, which is realized as a network of interconnected FCs. Custom layouts for the FCs were generated using Cadence Virtuoso [65] to compare the physical area of the flash-based digital circuits to their standard cell based counterparts. We obtained the layout of our FCs using design rules for flash devices that were obtained from the ITRS reports [59].

V.4.2 Flash-based Analysis Details

In this section, we describe the methodology we used to extract the delay, power and area of our multi-level flash-based design.

We first characterized a generalized FC, and generated delay, power and area models

for the FC in terms of m, n, and several other parameters of the FC. We use these models to estimate the delay, power and area of the mapped multi-level network of FCs. The parameters that determine the delay, power and area of FC^i are:

- The total number of cubes C^i_{tot} implemented in the FC (i.e. total number of pulldown stacks over all the FLAs in FC^i).
- The maximum number of cubes C^i_{FLB} in any FLB of FC^i (this number is bounded by CPB).
- The number of outputs N^i of the FC^i.

The delay of FC^i is proportional to both the total number of cubes in the FC (C^i_{tot}) and the maximum number of cubes over the FLBs of FC^i (C^i_{FLB}). The power of FC^i is proportional to the total number of cubes (C^i_{tot}) and the number of output of FC^i (N^i). The area of the layout of FC^i depends on all the three factors. We fix the number of inputs of the FC (m) to 6, in order to preserve the regularity of the FC and make it easier to place and route.

To characterize the delay, area and power of an FC, we constructed a library of FCs with number of outputs (N^i) varying from 1 to 3, and number of cubes per FC (C^i_{tot}) ranging from 1 up to 56 (this is the maximum number of cubes for a 6 input function, assuming that we perform the on-the-fly layout synthesis which maps the output minterm with the largest number of cubes to the default output minterm ($< 111 >$)), and the C^i_{FLB} varying from 1 to CPB (which is 3 in our work)

V.4.2.1 Delay Characterization and Estimation

For delay characterization, we run HSPICE simulations to measure the delays of FCs with C^i_{tot} ranging from 1 up to 56, and for C^i_{FLB} varying between 1 and CPB (which is 3). The latter condition corresponds to the case when the output of the FC^i is discharged through an FLB with 1, 2 or 3 cubes respectively. We also measure the maximum precharge delay D_{Pch} across all the FC^i configurations. The following equation explain how the delay of the flash-based design is computed.

$$Delay = D^{max}_{Pch} + \sum_{FC^i \in \Pi} \left[D_{FC}\left(C^i_{tot}, C^i_{FLB}\right) + (D_{OB}) \times \alpha_i \right] \tag{V.1}$$

Equation V.1 summarizes the critical path delay calculation methodology. D^{max}_{Pch} is the pre-characterized maximum precharge delay for all the FCs. For all the FC^i on the critical path Π, we look up the delay D_{FC} of the FC itself, and the delay D_{OB} of the output driver (if the FC drives a load greater than a threshold). C^i_{tot} is the total number of cubes (pulldown stacks) in FC^i and C^i_{FLB} is the maximum number of cubes among all the FLBs in FC^i. D_{OB} is the delay of the output buffer, which is only added if the fanout of FC^i exceeds a certain threshold. The binary variable α_i is set to 1 when the fanout of FC^i exceeds the threshold, and $\alpha_i = 0$ otherwise. In any FC, the output generation circuit is capable of driving an equivalent load of a 4×-5× buffer (which is equivalent to driving up to 100 gates of flash transistors in pulldown stacks). Recall that the flash transistor has 20× smaller input capacitance than a regular MOSFET, as discussed earlier. During layout synthesis, our CAD tool inserts an output buffer for each FC that has one or more outputs with a load higher than 96 flash transistor gates. This output buffer is a 4× buffer,

which is capable of driving a load equivalent to the input load of a CMOS buffer of size 16×-20×, effectively guaranteeing that our output buffer is strong enough to drive about 400 flash transistor gates, which is larger than any load encountered in our experiments. The total delay is equal to the summation of the delays of the FCs in the critical path Π, the maximum precharge delay, and the sum of the delays of the inserted output buffers. The critical path delay is found by running a static timing analysis tool which we implemented, using a dynamic programming model.

V.4.2.2 Power Characterization and Estimation

We first characterize the power of any FC by measuring the power (precharge as well as evaluation) for general FC configurations using HSPICE simulations. The configurations varied C^i_{tot} from 1 to 56 and N^i from 1 to 3. The results of these characterization runs are stored.

Equation V.2 shows the details of how power estimation is performed. Here, AF is the logic activity factor, which is taken to be 15%, a representative number for logic designs [3]. N^i is the number of outputs in FC^i. The total power is computed as the sum of the power of all the FCs in the design.

$$Power = AF * \left[\sum_{\forall FC_i} P(C^i_{tot}, N^i) + (P_{OB}) \times \alpha_i \right] \tag{V.2}$$

Benchmark	CMOS	Flash				CMOS	Flash	CMOS	Flash	CMOS	FLASH
	# Stdcells	# FCs	m_{Avg}	n_{Avg}	Avg # Cubes	Delay (ns)	Delay Ratio	Power (mW)	Power Ratio	Cell Area $(\mu m)^2$	Cell Area Ratio
b17	47500	6658	5.61	2.32	5.49	7.67	0.90×	25.97	0.49×	56964.97	0.67×
b20	22983	2901	5.60	2.32	6.02	6.11	0.50×	30.80	0.18×	27114.98	0.60×
b21	23324	2963	5.62	2.32	6.03	6.11	0.49×	30.93	0.18×	27521.69	0.61×
b22	34693	4362	5.61	2.33	5.97	6.16	0.47×	43.67	0.19×	40926.49	0.60×
s13207	2828	460	5.60	2.38	4.83	1.88	0.43×	0.98	0.91×	3365.70	0.67×
s15850	3735	594	5.50	2.30	4.90	2.63	0.67×	1.82	0.62×	4357.35	0.65×
s35932	10661	1290	5.24	2.60	6.62	0.74	0.29×	12.06	0.22×	12964.31	0.51×
s38417	10771	1593	5.68	2.22	5.04	1.48	0.89×	6.82	0.43×	12256.22	0.60×
s38584	13895	2077	5.59	2.16	4.93	2.03	0.90×	6.19	0.61×	16048.05	0.60×
multiplier	46363	5821	5.67	2.48	6.56	18.80	0.47×	105.41	0.11×	56460.10	0.57×
voter	22453	2708	5.38	2.53	6.10	5.91	0.58×	37.93	0.14×	26738.59	0.56×
square	38009	5109	5.63	2.49	5.61	18.26	0.49×	63.05	0.16×	46558.78	0.58×
Average	23101	3045	5.56	2.37	5.68	6.48	0.59×	30.47	0.35×	27606.43	0.60×
Stdev	15582.94	2038.51	0.13	0.13	0.64	6.08	0.20×	30.31	0.26×	18934.88	0.05×

Table 5.2: Delay, Power, Energy and Cell Area Ratios of Flash-based Digital Circuits Relative to Their CMOS Standard-cell Based Counterparts

V.4.3 Results and Analysis

We evaluated our flash-based digital circuit design approach by implementing a set of 12 of the largest benchmarks from ISCAS89 [66], ITC99 [67] and EPFL [68] benchmark suites. We compare the delay, power and area results of the flash-based implementation to a CMOS standard-cell based implementation of the benchmarks. Table 5.2 shows the benchmark name (column 1), the number of cells used to implement the design using a traditional CMOS standard-cell based approach (column 2), the number of FCs used to implement the design using the flash-based approach (column 3), the average number of inputs over all the FCs (column 4), the average number of outputs of the FCs (column 5), the average number of cubes in the FCs (column 6), the CMOS delay and the flash-based design delay ratio (columns 7 and 8), the CMOS power and the flash-based design power ratio (columns 9 and 10), the CMOS area and the flash-based design area ratio (columns 11 and 12). The delay, power and area ratios of the flash-based designs are relative to their CMOS standard-cell based counterparts. The average and the standard deviation of the results are shown in Table 5.2.

Comparing the average number of standard cells to the average number of FCs across all the benchmarks we observe that on average, each FC is equivalent to $\sim$7.6$\times$ standard cells. These FCs have an average of 5.56 inputs and 2.37 outputs, which are close to the maximum number of inputs (6) and outputs (3) in our design. The average number of cubes implemented in each FC, however, is low (5.68) compared to the maximum possible number of cubes for a 6-input logic function.

Table 5.2 shows that the flash-based design approach is ∼41% faster operation and consumes ∼65% lower power on average, compared to a traditional CMOS standard-cell based design approach. This is a significant improvement, and results in an energy improvement of ∼5× over the standard-cell based approach.

The key reasons for the reduced delay are:

- Lowered gate capacitance of the flash FET (20× lower than a MOSFET), as described in Chapter IV.
- The increased parasitics of the standard cells (due to the use of NMOS as well as PMOS devices, and the need for inputs to drive the gates of both types of devices).
- The use of shared (un-contacted) diffusions in the NAND stack reduces area as well as parasitics significantly, thus reducing delays.
- The FCs used to implement the benchmarks have relatively small sizes (the average number of cubes implemented in each FC is 5.68) which reduces the input capacitive loads.

The work shown in Chapter IV and [69] showed that the FCs had an improved delay (by ∼16%), power (by ∼65%) and area (by ∼46%) than the same logic implemented with standard-cells. The delay improvement for the flash-based design reported in Table 5.2 is lower than the results reported in Chapter IV and [69]. The lower delay is due the much smaller number of cubes per FC (average of 5.68 cubes per FC) in this work compared to the FCs (which had ∼ 35 cubes per FC on average). Further, due to the lower number of cubes per FC in the current work, output buffers are rarely needed, reducing the delay

further.

It is well known that dynamic designs consume greater power than static CMOS designs. Our FC based design consumes less power for several reasons. Despite being dynamic, the number of nodes being precharged is smaller than in a CMOS (domino or other dynamic) approach. Further, the long transistor stacks (since we choose $m = 6$) result in smaller evaluation currents, reducing power further. Also, in our design, exactly one FLB pulls down during every evaluation, reducing switching activity and hence power consumption. Finally, the I_{ds} of a 45nm flash FET is lower than that of our 45nm MOSFET, which results in a lower power consumption. The power improvements reported in Chapter IV and [69] match the power improvement reported in this work.

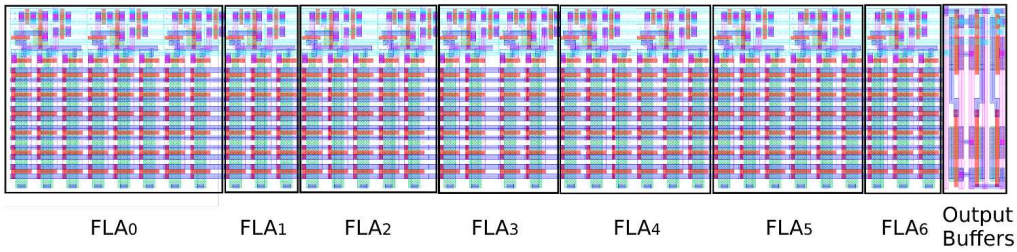


Figure 5.2: Layout View of an FC

We also report the area ratio of both implementations. The area reported for the CMOS standard cell based implementation is the sum of cell layout areas, while the area of our flash-based approach is the sum of the layout areas of all the FCs in the design. Design rules for flash were obtained from the ITRS 45nm flash technology node [59].

Digital circuits implemented in an FC use 0.6× the physical area of a CMOS-based design, on average (compared to 0.54× in Chapter IV and [69]). The slight increase in area in the current work arrives from the fact that there are fewer cubes per FC, making it harder to pack the FLAs. The area reported in Table 5.2 does not include the area overhead of the circuitry used to generate the high voltages for programming. We estimate that the circuitry used to generate high voltages will incur ∼2% area overhead. In Figure 5.2, we show the representative layout of a cluster of the FC which has 35 pulldown stacks.

We note that the standard deviation of the flash-based designs show more predictable delay (0.20× of CMOS), power (0.26× of CMOS) and area (0.05× of CMOS) compared to the standard deviation reported for CMOS, which is (6.08ns or 0.94× CMOS average) for delay, (30.31mW or 0.99× CMOS average) for power and (18934.88μm^2 or 0.69× CMOS average) for area.

V.4.4 FC Statistics

Figure 5.3 shows 4 histograms showing the distribution of some key FC design parameters for design b17. Studying the distribution of these parameters for a design enables us to optimize the CAD flow, since they allow us to determine the design impact of any CAD optimization that is performed.

Figure 5.3 reports the number of cubes per FC (C^i_{tot}). This parameter contributes significantly to the delay, power and area characteristics of the design. The number of cubes per FC histogram shows that, for benchmark b17, almost all of the FCs have less than 10 cubes in total, which suggests that in most cases, the output buffer will not be

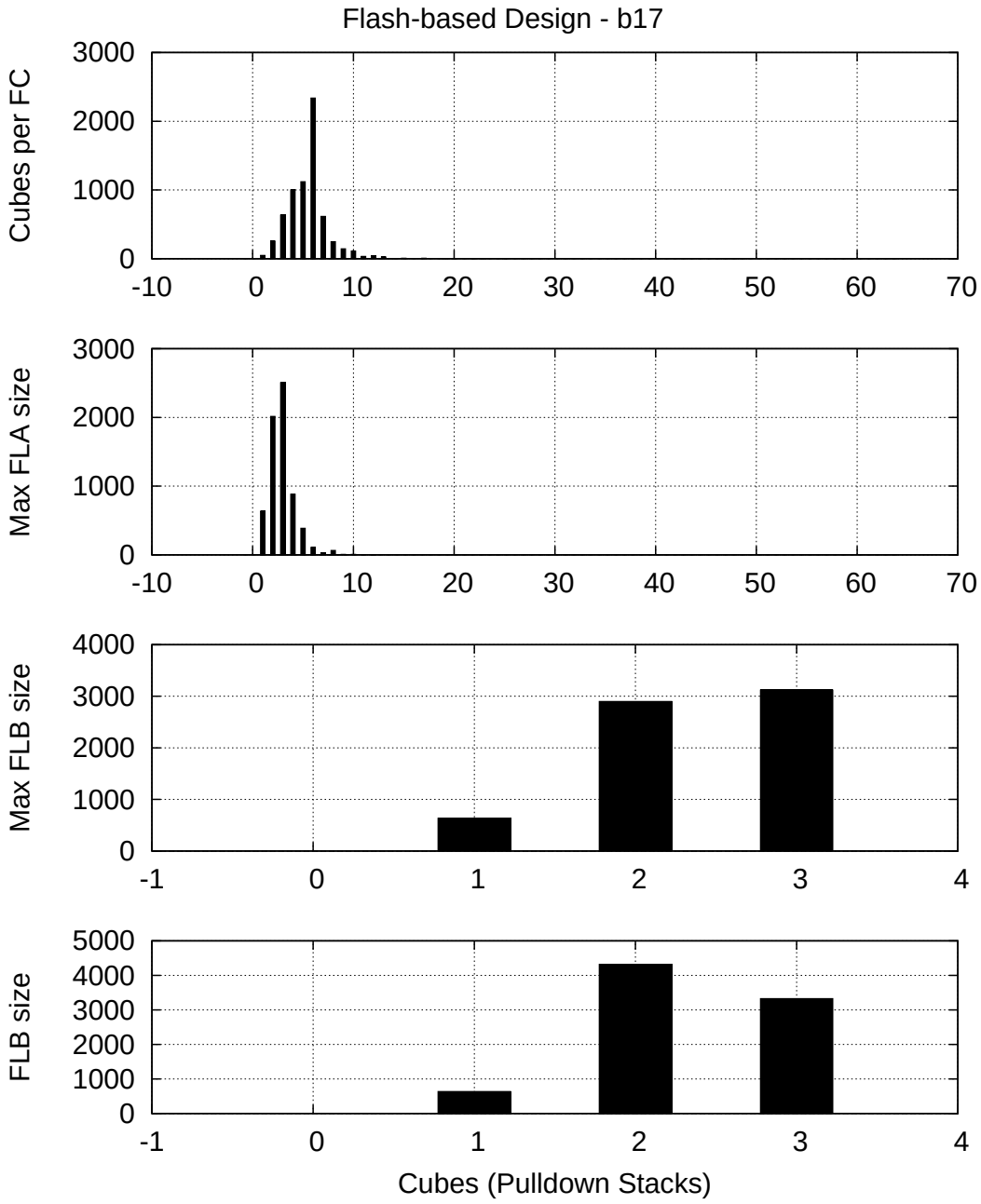


Figure 5.3: Histograms of FCs for the b17 Benchmark

inserted to the FCs in the design (except for high-fanout nodes). On the other hand, the "Max FLA size" histogram shows that most FLAs have 1 to 4 cubes. Since about 50% of the FLAs have 1, 2 or 4 cubes, there would be small FLBs in the design (since the optimal size of an FLB is 3 cubes). Hence an increased area overhead can result, as shown in the layout view of a representative FC (Figure 5.2). In this figure, note the wasted area in implementing FLA_3 (which has 4 cubes, 2 in each of its FLBs).

The "Max FLB size" is a histogram of the largest FLB of each FC. We note that max FLB sizes of 2 and 3 occur more frequently.

The "FLB size" histogram shows the distribution of all the FLBs in the design. A majority of FLBs have 2 cubes, as the histogram shows.

V.4.5 Shifting the Threshold Voltage

Flash-based digital circuits have the ability of tuning its delay and power characteristics, which is done through shifting the threshold voltage of the flash transistors in the circuit. This ability offers the flash-based digital circuits huge advantages over the traditional CMOS standard-cell based circuits when it comes to speed binning at the factory, aging mitigation and performing post-manufacturing ECOs. Figure 5.4 demonstrates the ability to tune the delay and power of flash-based circuit blocks by shifting the threshold voltage of the flash transistors in the flash-based circuits. The x-axis in the figure shows the V_T shift in mV, where the nominal V_T is chosen at the V_T values described in Section V.4.1. The left y-axis shows the delay ratio of the flash-based design compared to the CMOS standard-cell based design as the V_T of the flash transistors is shifted. The delay

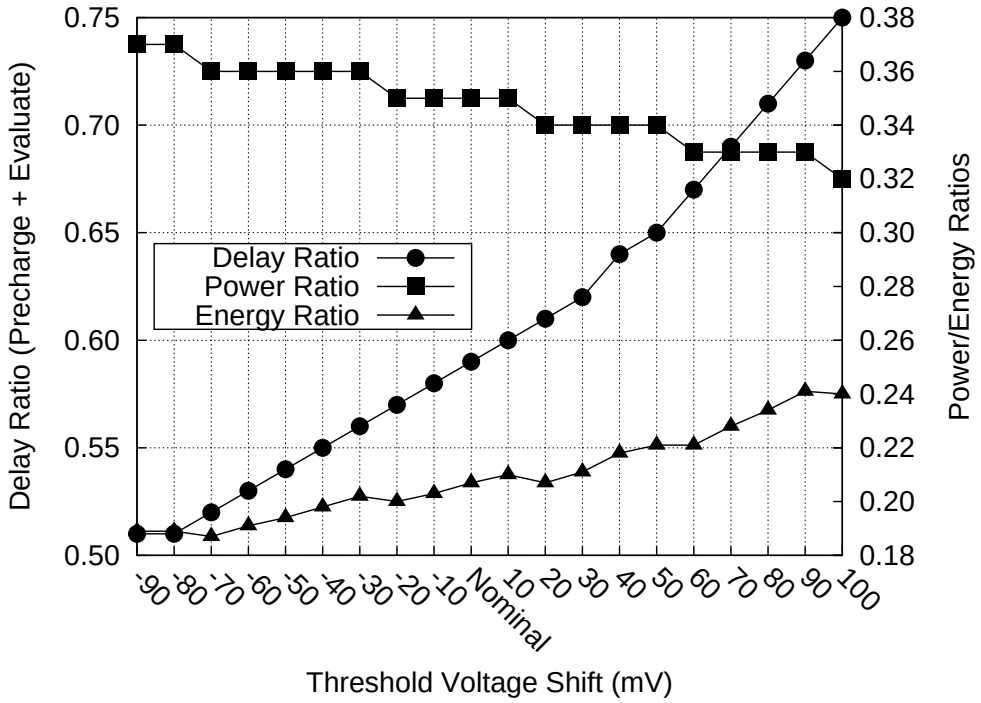


Figure 5.4: Delay, Power and Energy Characteristics of Flash-based Blocks as V_T is Shifted

of the flash-based design is the sum of the precharge and the evaluate delays. The right y-axis shows the average power and energy ratios of the flash-based designs compared to the standard-cell based designs as the V_T of the flash transistors is shifted. The delay, power and energy shown in Figure 5.4 are averaged across all the benchmark designs shown in Table 5.2. Figure 5.4 shows that by shifting the V_T of the flash transistors to a lower value than its nominal value, the delay of the design decreases and the power dissipated increases, while the energy consumption is decreased. Conversely, when the V_T is shifting to a higher value than the nominal V_T value, the delay increases and the power dissipated decreases, while the energy consumption is increased. These results confirm our ability to control the circuit delay, power and energy characteristics by fine tuning the threshold voltage of the flash-based design.

V.5 Chapter Summary

It was shown in Chapter IV how flash transistors can be used to implement digital circuits in an ASIC-like manner. However, the focus was on a single flash cluster (FC) which implements a function with a small number of inputs and outputs.

Work presented in this chapter, is the first, to the best of the authors' knowledge, to use flash transistors to implement complete digital circuit blocks, in the form of an interconnected network of FCs. The focus of this chapter is on logic clustering, on-the-fly physical synthesis of all the FCs of a design, and the automatic characterization of the delay, power and area of the resulting circuit.

Our characterization results show that, averaged over 12 large designs, our approach yields 0.59× the delay, 0.35× the power, and 0.60× the area of the equivalent circuit implemented using CMOS standard cell-based design. It is generally rare that a circuit methodology yields results that are better than existing commercial standard-cell based flows in terms of delay, area, power *and* energy, and in this sense, we submit that our results are significant.

CHAPTER VI

SAT-BASED OPTIMIZATION FOR FLASH-BASED DIGITAL DESIGNS

In this chapter, we present a SAT-based optimization engine that improves the current flash-based CAD flow. This engine uses don't-cares to perform multi-node, multi-level logic optimization of the flash-based design.

VI.1 Background

Although the optimization technique presented in this chapter does not explicitly compute the don't-care set of the design, it uses concepts from multi-level don't-care theory to perform optimization. Therefore, we will begin with a brief background about multi-level don't-cares.

A logic function can be either a *completely specified function* (CSF) or an *incompletely specified function* (ISF). A CSF f is a function $f : B^n \rightarrow B$, where $B = \{0,1\}$. In other words, for any input minterm, the output is specified to be 1 or 0 (i.e. all minterms are *care minterms*). Conversely, an ISF g is a function $g : B^n \rightarrow \{0,1,*\}$, such that at least one input minterm $m \in B^n$ is mapped to the $*$ (don't care) value. In other words, a subset of the input minterms do not map the output of the function to 1 or 0. These minterms are called the *don't-care* minterms. The complete set of don't-care minterms in an ISF is referred to as the *complete don't-care* (CDC) set. When a function is implemented in

hardware, these don't-care minterms are mapped to $\{0,1\}$. This mapping process plays an essential role in multi-level logic optimization. Careful assignment of the don't-care minterms of a logic function to logic 1 or 0 can lead to significant improvement in the implementation characteristics of the function.

Any combinational Boolean logic network η forms a *directed acyclic graph* (DAG), in which each node i of the graph represents a logic function f_i. The node i also has an output variable y_i associated with it such that $y_i = f_i$. There is an edge in the DAG from node i to node j, if f_j is a function of the variable y_i (i.e. f_j depends on y_i). In this case, we call the node i a *fanin* (FI) of node j. Also, we call the node j a *fanout* (FO) of node i. In general, a node i falls in the *transitive fanin* (TFI) of a node j in a logic network η, if there is a path from node i to node j in the DAG representing the network η. Also, a node k falls in the *transitive fanout* (TFO) of node j, if there is a path from node j to node k in the DAG representing the network η. The set of internal nodes $\{y_i\}$ is referred to as Y. The set of primary inputs $\{x_i\}$ is referred to as X, and the set of primary outputs $\{z_i\}$ is referred to as Z.

In a multi-level logic netlist, the don't-cares can take the form of *satisfiability don't-care* (SDC), *observability don't-care* (ODC) or *external don't-care* (XDC).

The SDC of a network is $SDC = \sum_{j=1}^{m}(y_j \oplus f_j)$, where $|Y| = m$. An SDC minterm $m_i^{SDC} \in B^{|Y|+|X|}$. In other words, if we apply all the possible combination of inputs at the primary inputs of the logic netlist, the minterm m_i^{SDC} can never be observed.

An ODC minterm $m_{j,k}^{ODC}$ with respect to a logic node j and an output k, is a primary

input minterm such that if we change the value of the node y_i corresponding to $m_{j,k}^{ODC}$, the change will not be observed at the primary output k of the network. In other words, $ODC_{j,k} = \{x \in B^{|X|} \;\; s.t. \;\; z_k(x)|_{y_j=0} = z_k(x)|_{y_j=1}\}$. For any node j, $ODC_j = \prod_k (ODC_{j,k})$. When a logic node is minimized against its ODCs, the change in the node may require the recomputation of the ODCs of the other nodes in the same network. A subset of the ODCs are called *compatibility observability don't-cares* (CODCs). CODCs have a "compatibility" property that allows them to be easily used in logic minimization. In other words, optimizing a logic node using CODCs is compatible with any previous CODC-based optimizations done on other nodes.

For each primary output, XDCs are a set of minterms of the primary input for which the value of the primary output is disregarded.

Multi-level logic optimization using don't-cares has been done in the past for traditional logic networks aiming towards reducing the number of literals of a logic node. The SIS package [70] has a *full_simplify* command that builds a *reduced ordered binary decision diagram* (ROBDD) [71]. Also, the work in [72] uses approximate compatible observability don't-cares (CODC), while the work in [73] uses complete don't-cares (CDC) to minimize a logic node. These approaches are technology-independent, and their improvement are often erased during technology mapping [74].

VI.2 Previous Work

The work presented in Chapter IV and [69], as well as the work presented in Chapter III and [64] only report flash-based implementations at the cell level. Chapter V and [75] present a design flow to implement flash-based digital circuits at the block level. The flow reads a logic netlist and performs synthesis and mapping to a network of interconnected FCs. Although the work achieves impressive delay, power and area results, it uses a greedy algorithm to perform the mapping of the design. In this work, we add an additional layer of optimization, using don't-cares in a multi-level logic network setting, to improve the design characteristics. The flash-based designs implemented with our optimization engine achieve an additional 10% lower delay, 5% lower power, 15% lower energy and 8% lower physical area when compared to their counterparts that are implemented using the approach in [75]. Note that these are post technology mapping improvements, making them more significant, since logic optimizations for standard cell-based designs are often erased during technology mapping.

Work in the area of logic optimization includes [70, 72, 73]. The SIS package [70] performs logic simplification of a logic netlist. SIS optimizes one node at a time, on a technology independent netlist, unlike our approach. In the simplification process (*full_simplify* command), SIS builds an ROBDD [71] for the entire design. This often causes memory issues, since ROBDDs require high and unpredictable memory utilization, limiting the command to designs which are much smaller than those we evaluate in our work. In contrast, our work scales elegantly since it performs the optimization over a

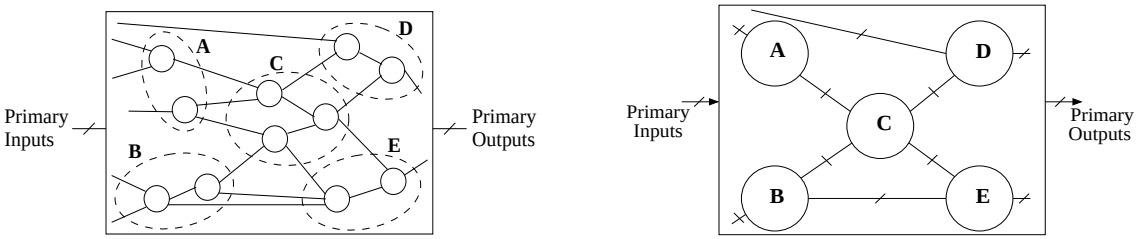
small cut in the circuit, and uses a SAT-based algorithm, which is much faster than building ROBDDs. Another key advantage to our approach is that we perform post-technology mapping optimization, which results in guaranteed improvements in the design characteristics. Technology independent optimization can be lost after technology mapping [74]. Finally, our approach optimizes a cluster of nodes at once, unlike the optimization commands in SIS.

In [72], the authors present a solution to ROBDD-based don't-care optimization by using approximation. They make a cut in the circuit around a particular node, thus creating a sub-network that is tractable in size. The optimization of that particular node is done by constructing an ROBDD representing the sub-network. Since they only build ROBDDs of small sub-networks, the solution can scale to industry size designs. Their approximation technique achieves a result quality that is very close to that obtained by building an ROBDD for the entire design, with a fraction of the memory requirement. Our approach presented in this chapter has several advantages over the work in [72] such as, i) our approach performs post technology mapping optimization which is more effective than technology independent optimization, ii) we perform the optimization over a cluster of logic nodes at once, thereby exploiting greater optimization flexibility and iii) we use a SAT-based engine instead of ROBDDs to perform the optimization. SAT-based engines are generally faster and have lower and predictable memory requirements compared to ROBDDs.

The work in [73] uses a SAT-based algorithm to perform the logic optimization for a

node. Similar to [72] they start by making a cut around a node N and build a SAT instance that represents the sub-network inside the cut. Then they create a duplicate sub-network with an altered node N^*. Node N^* is annotated with CDCs that are computed through random simulations. Finally, they use miters to check the equivalence of the two sub-networks using a SAT-based checker. Our work distinguish itself from [73] by performing post technology mapping optimization and perform the optimization considering a cluster of nodes, rather than one node at a time. Finally, [73] calculates the don't care set by doing an all-SAT computation, while the goal of our approach is to simply check if a cube in a cluster can be reassigned to a different output. This requires a simple SAT computation.

VI.3 Approach



(a) Logic Netlist (b) Flash-based Digital Circuit

Figure 6.1: Converting a Logic Netlist into a Flash-based Design

VI.3.1 Overview

The optimization technique presented in this chapter builds upon the flash-based design flow presented in Chapter V. The top-level flash-based design flow to convert a

technology-independent logic netlist into a network of interconnected FCs was covered in Section V.3.2. The main difference between the work presented in this chapter and the work presented earlier in Chapter V is that we apply an additional step in the design flow to optimize the final design. The flash-based netlist conversion performs the mapping from the clusters of nodes as illustrated by the dotted ovals in Figure 6.1(a) into the FCs depicted by the solid circles in Figure 6.1(b). The work of this chapter further optimizes the FCs shown in Figure 6.1(b).

After each cluster R is formed during the clustering process, we invoke our optimization engine which first creates a list of candidate optimizations for cluster R. This is done by examining the structure of the cluster R. This step will be discussed further in Section VI.3.3.1. If the cluster R meets the criteria for optimization, our engine will form a *cluster cut* around the cluster R in the logic netlist, as shown in Figure 6.2. Afterwords, the optimization engine will perform *cluster cut-based* (CCB) optimization on the cluster R. The details of how the cluster cut is formed as well as the CCB optimization are described in detail in Section VI.3.3.2.

This chapter targets the optimization of the FCs generated by our flash-based design flow presented in Chapter V. In this chapter we will only cover our SAT-based optimization approach. However, it is very important to understand the structure of the FC as well as our flash-based CAD flow in order to fully understand our SAT-based optimization approach. The reader is encouraged to read Chapter IV (specifically, Section IV.3.2, Section IV.3.5 and Section IV.4) as well as Chapter V (specifically, Section VI.3.3 and

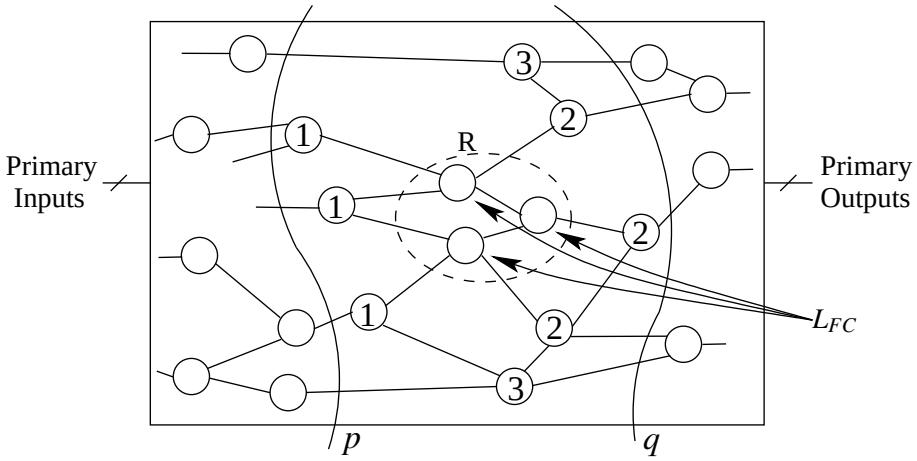


Figure 6.2: Cluster Cut Based (CCB) Optimization

Section V.4).

VI.3.2 Flash Cluster Circuit Design

The work presented in this chapter is based on the FC presented in Chapter IV. It uses the same FC structure presented in Section IV.3.2. The reader is encouraged to read about the structure of the FC in Section IV.3.2. The key point that we want to reiterate is that the FC is a dynamic circuit, so its default (precharged) output state is $2^n - 1$ (or $< 111 >$ for n = 3), where n is the number of outputs of the function implemented in the FC. Hence we do not need to implement $FLA_{(2^n-1)}$ (or FLA_7 in our example), and only need to implement $2^n - 1$ FLAs (shown as FLA_0, FLA_1, $\cdots$, FLA_6 in Figure 4.2, for n = 3). From an optimization standpoint, it would benefit us if a cube assigned to FLA_i ($i \neq 2^n - 1$), is reassigned to FLA_{2^n-1}, since it will not need to be implemented. This is one of the optimizations we explore. A different optimization step that we perform is

based on the structure of the FLA (described earlier in Section IV.3.2.1). The FLA consists of multiple FLBs, each of which implements a maximum number of cubes defined by the variable *CPB* (which was determined to be 3, and covered in Section IV.3.2.2). If FLA_i has (say) $3x+1$ cubes and FLA_j has (say) $3y+2$ cubes (where $x, j \geq 0$) then if we move a cube from FLA_i to FLA_j, we can reduce the area of the FC. We will further describe this optimization step in Section VI.3.2.1 and Section VI.3.3.1.

VI.3.2.1 Layout Analysis

As discussed earlier in Section IV.3.2.2, each FLB can implement up to 3 cubes in order to maintain healthy delays. Figure 6.3 shows the layout view of the three types of FLB (based on the number of cubes implemented in an FLB). FLB-A implements three cubes, FLB-B implements two cubes and FLB-C implements one cube. As shown in Figure 6.3, the physical areas of the three FLB types do not scale linearly with the number of cubes implemented in each FLB. This is due to the fixed area of the output generation circuit (shown on top of the FLB) For example, FLB-A implements 3 cubes and has an area of 2.48 $(\mu m)^2$, FLB-B implements 2 cubes and has an area of 1.94 $(\mu m)^2$ and FLB-C implements 1 cube and has an area of 1.94 $(\mu m)^2$. Note that while FLB-C implements half the number of cubes that FLB-B implements, it occupies the same area. We clearly see a great opportunity for optimization by eliminating FLBs of type FLB-C, and moving the cubes they implement into another FLB-C to form an FLB-B, thereby reducing the area by 1.94 $(\mu m)^2$. Note that from an optimization standpoint, all cubes in $FLB_{i,x}$ are candidates for being moved in case there exists a k such that $FLB_{i,k}$ is of type FLB-C. We

can also move a cube from FLB-C to an FLB-B to form an FLB-A which will result in area reduction of 1.4 $(\mu m)^2$. Finally, all cubes in any implemented FLB can be reassigned to the precharged ($< 111...1 >$) output, thus eliminating the FLB completely, since the precharged output is never implemented.

VI.3.3 FC-based CAD Flow

The SAT-based optimization presented in this chapter is performed on the output of the flash-based CAD flow described earlier in Section V.3.3. In the original CAD flow (described Section V.3.3), the input netlist is clustered into FCs (where FC_i implements $F^i_{m,n}$), with a goal of minimizing the wiring between FC's. We choose $m \leq 6$ and $n \leq 3$. The CAD flow uses a greedy algorithm that successively groups the nodes from the technology-independent logic netlist to form clusters. After this, we obtain a multi-level netlist of interconnected FCs. We run our optimization engine (described in this chapter) at the end of the clustering step, after we have generated the multi-level netlist of FCs.

The FCs, FLAs and FLBs are extremely regular in their physical characteristics, making them amenable to our *cluster cut-based* (CCB) optimization engine. After performing CCB optimization, we inspect the fanout load of the i^{th} output of FC_j, and additional buffers are added for that output, as necessary.

To quantify the utility of the flash-based circuit design flow, the same input netlist is synthesized and mapped using commercial standard-cell based CAD tools. The resulting designs (flash-based and standard-cell based) are compared in terms of their delay, area, power and energy, over a number of designs.

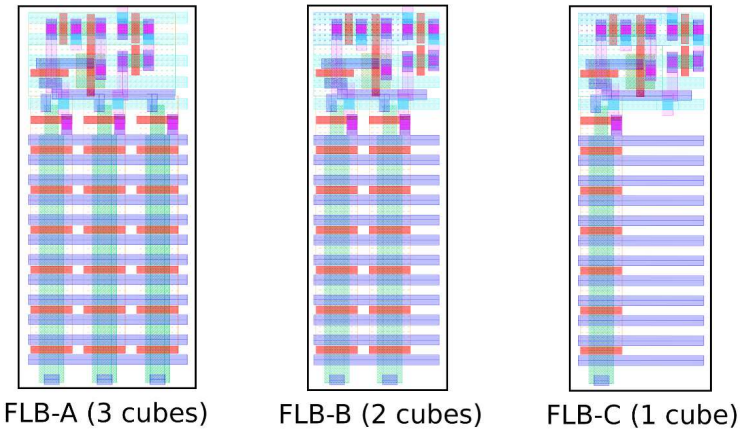


Figure 6.3: Layout View of Possible FLB Structures

VI.3.3.1 Optimization Candidate List

Our optimization is applied to every mapped FC produced by the clustering CAD flow of Section VI.3.3. We first create a list of ways that the current FC can be optimized in decreasing order of benefit. Our optimization algorithm is focused on reducing the number of cubes implemented in an FC, with a goal of reducing physical area. These possible changes explored for any FC are based on the fact that an FLB-C (which implements 1 cube) occupies the same area of an FLB-B (which implements 2 cubes) as discussed earlier in Section VI.3.2.1. Therefore, if we move a cube from an FLB-C to another FLB-C (which then becomes an FLB-B), we reduce the FC's size by 1 FLB-C. We can also reclaim some area back by moving a cube from an FLB-C to an FLB-B (which then becomes an FLB-A). Also, in Section VI.3.2, we have shown that since the FC is a precharged structure, FLA_7 (that correspond to the precharge state $< 111 >$) is not implemented.

Therefore any cube that is mapped to the $< 111...1 >$ output (or FLA_7, since we use $n = 3$) is essentially removed from the FC (since it will not be implemented). In our approach, the candidate list is populated with candidate optimizations in the following order.

A Optimizations where a cube of FLB-C is moved to another cube of FLB-C.

B Optimization in which a cube of type FLB-C is moved to a cube of type FLB-B.

C Optimization where a cube of type FLB-C is moved to FLA_7.

D Optimization in which all cubes of type FLB-B are moved to FLA_7.

Note that if $FLB_{i,k}$ is a move candidate type A, B, C or D, then all cubes in $FLB_{i,x}$ are tested for moving (for all x). This step analyzes each FC for potential optimization opportunities. It records all these possible opportunities in an ordered list C. The items in the list C represent a possible optimized FC (we call it FC^*). This list is ordered based on the amount of area reduction that each optimization opportunity would yield. In this step, if the list C is empty, the optimization algorithm skips the current FC, and iteratively checks the next FC. Otherwise, if the list C is not empty, the algorithm performs cluster cut-based optimization on the current FC.

VI.3.3.2 Cluster Cut-based (CCB) Optimization

The CCB optimization algorithm is described in Algorithm 2. We start with a cluster R that has a list of nodes L_{FC}, as shown in Figure 6.2. After optimization candidate list (C) is computed for R, and assuming that C is non-empty, the algorithm forms the p and q cuts shown in Figure 6.2. The algorithm forms the cut p towards the primary outputs of the network η. The cut p is formed by collecting all the nodes in the TFI cone of each

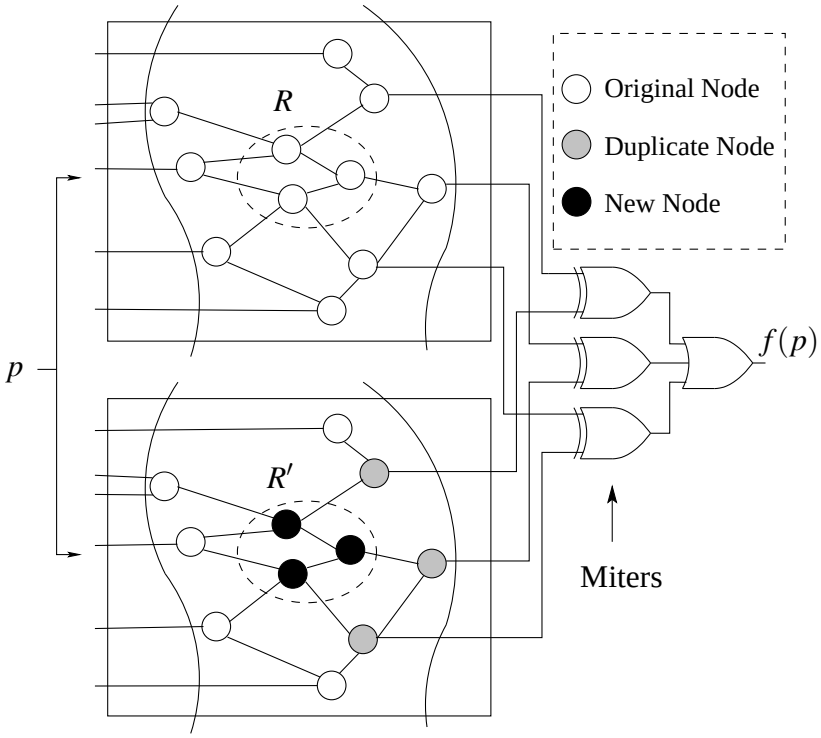


Figure 6.4: Example Miter Circuit

node in L_{FC}, up to K levels. We call this set of nodes L_{TFI} (represented by the nodes marked "1" in Figure 6.2). Here K is a design parameter. A large value of K results in bigger subnetworks and at the cost of runtime, better result quality. In our implementation, we find that $K = 2$ provides a good balance between runtime and result quality. Next, our CCB algorithm forms the cut q towards the primary outputs of the network η. The cut q is formed by collecting two sets of nodes. The first set of nodes (L_{TFO}) are the nodes that fall in the TFO cone of the nodes in L_{FC} (the nodes in cluster R), up to K levels. These are represented by the nodes marked "2" in Figure 6.2. The second set of nodes (L_{TFO_TFI}) are the recursive fanin nodes of the nodes in the set L_{TFO} up to depth K (represented by

the nodes marked "3" in Figure 6.2).

These sets of nodes L_{TFI}, L_{TFO}, L_{TFO_TFI} and L_{FC} are collected for every cluster R and form a network η_{FC} (shown at the top of Figure 6.4), which will be used to test every optimization possibility R^* in C. Another network, η_{R^*} (shown at the bottom of Figure 6.4) is formed by replacing the nodes of cluster R (L_{FC}) in η_{FC} with the nodes of R^* (L_{FC^*}), represented by the solid black circles in Figure 6.4. We also replace the nodes L_{TFO}, with a duplicate copy L_{TFO^*} (represented by the solid grey circles in Figure 6.4), since they fanout from the nodes L_{FC^*}. We keep the nodes L_{TFI} and L_{TFO_TFI} as is (they are the same as the corresponding nodes in η_{FC}), since they are not affected by the change in the function represented by the nodes L_{FC^*}.

Assume that the cluster size $|L_{FC}|$ is 3. The generation of the nodes L_{FC^*} from the nodes L_{FC} is straightforward. Consider a candidate optimization R^* which moves minterm $m \in 2^{|P|}$ from output minterm $< 000 >$ to $< 011 >$.

In each of the 3 nodes of L_{FC}, m is in the offset, since the output for m in L_{FC} is $< 000 >$. To construct L_{FC^*}, we simply take m and move it to the onset of the second and third nodes (since the desired output minterm has a 1 value for the second and third cluster nodes). A similar transformation from L_{FC} to L_{FC^*} can be made for any candidate optimization R^*. This discussion can be generalized to a cube c, as well.

Our CCB optimization algorithm uses a SAT-based check to test if cluster R^* (nodes L_{FC^*}) represents a valid replacement of the cluster R (nodes L_{FC}). Our SAT-based check will test if the network η_{FC^*} with the optimized cluster R^*, is equivalent to the original net-

work η_{FC}. This is done by constructing a SAT instance S_C representing both the networks η_{FC} and η_{FC^*}. We add to the SAT instance S_C, miter clauses that represent the XOR of the outputs of η_{FC} and η_{FC^*}, (i.e. as the XOR gates shown in Figure 6.4), then we add a clause that represents the OR of all the miter outputs. The output of the OR gate is a function ($f(p)$) of the primary inputs (p) to the networks η_{FC} and η_{FC^*}. Finally, we add a clause to S_C that forces the the function $f(p)$ to be "true".

We use a SAT checker (in this work, we use MiniSat [76]) to check the SAT instance S_C. If S_C is satisfiable, it means that there exist an input combination on p that makes the output of cluster R and R^* different. This means that the networks η_{FC} and η_{FC^*} are not equivalent. In this case, the CCB optimization algorithm continues to check a different R^*. However, if the result of the SAT checker is unsatisfiable, it means that the networks η_{FC} and η_{FC^*} are equivalent. Hence, R^* (the nodes L_{FC^*}) represents a valid optimization of cluster R (the nodes L_{FC}). In this case, the nodes L_{FC} are replaced with the new nodes L_{FC^*} and the algorithm repeats the process on another cluster R.

Proof of Correctness: Consider a cluster L_{FC} which is replaced by cluster L_{FC^*} under optimization R^*. Then,

Theorem VI.3.1 *If L_{FC^*} is a valid replacement for L_{FC} in η_{FC}, then L_{FC^*} is a valid replacement for L_{FC} in η.*

Let L_{FC^*} be a valid replacement for L_{FC} in η_{FC}. In other words, the SAT instance S_C returns UNSAT, and hence the $|q|$ outputs of η_{FC^*} are logically equivalent to the $|q|$ outputs of η_{FC}, for all input minterms $m \in 2^{|p|}$. Now consider the $|X|$ primary inputs of

η. Let them enumerate all $2^{|X|}$ values, and let the resulting values observed on the p space be P. This is also called the *forward image* of X on the p cut. Since $P \subseteq 2^{|p|}$, the values of the output nodes of η_{FC^*} and η_{FC} would be identical in η, and hence the values of the output nodes of η would also be identical after the optimization.

Algorithm 2 Optimize a Cluster of Nodes

```
C = opt_choices(L_FC)
if (C is not empty) then
   L_TFI = get_tfi(L_FC, K)
   L_TFO = get_tfo(L_FC, K)
   L_TFO_TFI = get_tfi(L_TFO, 1)
   for each C_i in C do
      L_FC* = get_optimized_cluster(C_i)
      S_C = construct_sat_inst(L_FC,L_FC*,L_TFI, L_TFO, L_TFO_TFI)
      add_miters(S_C)
      S_R = MiniSat(S_C)
      if S_R == UNSAT then
         Replace L_FC by L_FC*
         break
      end if
   end for
end if
```

VI.4 Experiments

VI.4.1 Simulation Environment

Similar to Chapter V, the designs presented in this work are implemented in a 45nm process technology. In our experimental framework, the CMOS standard cell based digital circuits are synthesized and mapped using a 45nm Nangate FreePDK45 Open Cell Library [54, 55] using Synopsys Design Compiler [56]. The delay, power and area of the

Benchmark	CMOS	Flash	CMOS	Flash		CMOS	Flash		CMOS	Flash		Runtime	
	No. Stdcells	No. FCs	Delay (ns)	Delay Ratio		Power (mW)	Power Ratio		Cell Area $(\mu m)^2$	Cell Area Ratio		Flow [75]	CCB
				Flow [75]	CCB		Flow [75]	CCB		Flow [75]	CCB	Seconds	Ratio
b14	9892	1435	5.95	0.44×	0.38×	11.18	0.25×	0.24×	11523.65	0.67×	0.60×	17.75	1.17×
b15	13117	1943	4.91	0.80×	0.75×	7.43	0.51×	0.49×	15513.65	0.73×	0.63×	38.10	1.21×
b17	43848	6658	7.65	0.56×	0.88×	23.38	0.54×	0.53×	52188.14	0.72×	0.63×	675.38	1.42×
b20	20040	2901	6.15	0.53×	0.50×	26.05	0.21×	0.21×	23316.76	0.68×	0.63×	102.29	1.32×
b21	20511	2963	6.09	0.49×	0.50×	26.67	0.21×	0.21×	23884.41	0.69×	0.63×	109.85	1.59×
b22	30510	4362	6.80	0.50×	0.42×	37.58	0.22×	0.22×	35509.40	0.68×	0.63×	229.66	1.46×
s5378	1750	279	1.07	0.67×	0.67×	1.02	0.54×	0.54×	2090.23	0.65×	0.65×	1.19	1.08×
s6669	2693	329	4.90	0.43×	0.44×	2.63	0.26×	0.24×	3276.85	0.55×	0.52×	0.99	1.46×
s9234	2052	330	1.61	0.62×	0.55×	0.99	0.67×	0.60×	2373.78	0.69×	0.60×	0.97	1.47×
s13207	2646	460	1.84	0.47×	0.40×	0.89	0.99×	0.91×	3122.31	0.70×	0.64×	2.28	1.37×
s15850	3423	594	2.76	0.50×	0.43×	1.66	0.66×	0.63×	3945.05	0.70×	0.65×	5.47	1.14×
s35932	10371	1290	0.73	1.04×	0.30×	11.35	0.23×	0.22×	12579.67	0.51×	0.48×	15681.14	1.35×
s38417	10199	1593	1.50	0.89×	0.69×	6.36	0.43×	0.44×	11474.18	0.59×	0.59×	21.2	1.30×
s38584	12584	2077	2.14	0.53×	0.52×	5.63	0.66×	0.64×	14302.02	0.64×	0.63×	4558.66	1.23×
multiplier	44038	5821	19.35	0.46×	0.42×	98.49	0.11×	0.11×	53150.26	0.59×	0.56×	715.39	1.21×
voter	21166	2708	6.16	0.61×	0.55×	35.10	0.14×	0.15×	24943.35	0.57×	0.55×	38.88	1.05×
square	34052	5109	18.48	0.45×	0.41×	56.48	0.17×	0.17×	41201.01	0.64×	0.58×	1200.49	1.11×
arbiter	12674	1742	4.29	0.38×	0.40×	9.16	0.32×	0.31×	16514.88	0.55×	0.43×	27.84	1.30×
bar	5422	812	1.32	1.10×	1.07×	5.00	0.31×	0.29×	6640.16	0.58×	0.52×	17.90	1.23×
sin	9655	1549	17.84	0.49×	0.44×	16.41	0.17×	0.16×	11922.65	0.64×	0.56×	159.36	1.01×
Average	15532	2248	6.08	0.59×	0.53×	19.17	0.38×	0.36×	18473.62	0.63×	0.58×	1180.24	1.27×
Stdev	13286.46	1893.56	5.80	0.20×	0.19×	23.94	0.23×	0.22×	15879.09	0.06×	0.06×	3563.96	0.16×

Table 6.1: Delay, Power and Cell Area Ratios of Flash-based Digital Circuits (with and without Optimization) Relative to Their CMOS Counterparts ($K = 2$)

CMOS standard cell based digital circuits are extracted using Design Compiler. Custom scripts were created to generate the flash-based digital circuit. The model cards obtained from the model card regression procedure described in Section III.4.2 were used to model the flash transistors in the flash-based circuits. For CMOS devices, we used a 45nm PTM process [57]. The target programmed threshold voltages used in our designs are (VT_0 = -0.5 V) and (VT_1 = 0.5 V). The flash-based FCs were simulated in HSPICE and the correct logical operation of the flash-based digital circuit, realized as a network of interconnected FCs, was verified. The delay, power, energy and area of the flash-based designs were characterized using the tool-chain presented in Section V.4.2.

VI.4.2 Results and Analysis

We evaluated the flash-based digital circuit design approach by implementing a set of 20 of the largest benchmarks from ISCAS89 [66], ITC99 [67] and EPFL [68] benchmark suites, with and without our SAT-based optimization engine. We compare the delay, power and area results of the flash-based implementations to a CMOS standard-cell based implementation of the benchmarks. All the benchmarks are too large to run SIS *full_simplify* command on them. However, we run the command *simplify* on each of the designs before running them through the flash-based design flow or the CMOS standard-cell implementation flow.

Table 6.1 shows the benchmark name (Column 1), the number of standard cells used in the traditional CMOS standard-cell based approach (Column 2), the number of FCs used in the flash-based design approach (Column 3), the CMOS delay (Columns

4), the delay ratio of the flash-based design without optimization (Column 5) and with optimization (Column 6), the CMOS power (Column 7), the flash-based design power ratio without optimization (Columns 8) and with optimization (Column 9), the CMOS area (Column 10), and the flash-based design area ratio without optimization (Columns 11) and with optimization (Column 12). We also report the runtime for flash-based design flow without optimization (Column 13) and with optimization (Column 14). The delay, power and area ratios of the flash-based designs are relative to their CMOS standard-cell based counterparts. Table 6.1 shows the average and the standard deviation of the results of the benchmarks.

Table 6.1 shows that the flash-based design approach with optimization is $\sim$47% faster and consumes $\sim$64% lower power on average, compared to a traditional CMOS standard-cell based design approach. This is a significant improvement, and results in an energy improvement of $\sim 5\times$ over the standard-cell based approach. The CCB optimization obtained an improvement of $\sim$10% in speed and $\sim$5% in power dissipation compared to the unoptimized results. We also report the area ratio of the flash-based and CMOS standard cell implementations. The area reported for the CMOS standard cell based implementation is the sum of cell layout areas, while the area of our flash-based approach is the sum of FC layout areas. Design rules for flash were obtained from the ITRS 45nm flash technology node [59]. Digital circuits implemented using the flash-based implementation with our optimization engine use $\sim 0.58\times$ the physical area of a CMOS-based design, on average. This an improvement of $\sim$8% over the flash-based implementation without op-

timization. The results of the flash-based implementation with optimization are reported for $K = 2$ (refer to Section VI.3.3.2).

On average, enabling the optimization engine in the flash-based design flow increases the runtime by $\sim$27% compared to the flash-based design flow without optimization.

We also report the standard deviation of the results of the flash-based designs (in Table 6.1). As shown in the table, the standard deviation of the optimized flash-based designs, similar to the unoptimized flash-based designs, indicate that the delay, power and area results of the optimized flash-based designs can be predicted with small error.

We ran our optimization engine with $K = 3$, to show the effect of varying the parameter K on the quality of the results and the total runtime. As shown in Table 6.2, for $K = 3$, the flash-based designs implemented using our optimization approach show an improvement of 8% (delay), 7% (power), and 11% (area) over the approach shown in Chapter V (also [75]). In other words, using a value of $K = 3$, the results quality improved by 2% (power) and 3% (area) and a degradation by 2% (delay) compared to using a value of $K = 2$. Note that the optimization is area driven, which explains why the delay metric was not improved when the value K was increased. The runtime for $K = 2$ (from Table 6.1) was shown to be 1.29$\times$ compared to the unoptimized case. The corresponding runtime ratio for $K = 3$ is 1.87$\times$ (as shown in Table 6.2). The standard deviation of the results is shown in the bottom row of Table 6.2, and shows the same behavior as seen in Table 6.1.

We have shown in Section VI.3.3.1 how the candidate optimizations are selected.

Benchmark	CMOS	Flash	CMOS	Flash		CMOS	Flash		CMOS	Flash		Runtime	
	No. Stdcells	No. FCs	Delay (ns)	Delay Ratio		Power (mW)	Power Ratio		Cell Area $(\mu m)^2$	Cell Area Ratio		Flow [75]	CCB
				Flow [75]	CCB		Flow [75]	CCB		Flow [75]	CCB	Seconds	Ratio
b14	9892	1435	5.95	0.44×	0.36×	11.18	0.25×	0.24×	11523.65	0.67×	0.59×	17.75	2.49×
b15	13117	1943	4.91	0.80×	0.57×	7.43	0.51×	0.48×	15513.65	0.73×	0.61×	38.10	2.27×
b17	43848	6658	7.65	0.56×	0.49×	23.38	0.54×	0.52×	52188.14	0.72×	0.61×	675.38	1.84×
b20	20040	2901	6.15	0.53×	0.45×	26.05	0.21×	0.20×	23316.76	0.68×	0.59×	102.29	1.70×
b21	20511	2963	6.09	0.49×	0.47×	26.67	0.21×	0.20×	23884.41	0.69×	0.60×	109.85	2.55×
b22	30510	4362	6.80	0.50×	0.41×	37.58	0.22×	0.21×	35509.40	0.68×	0.59×	229.66	2.62×
s5378	1750	279	1.07	0.67×	0.67×	1.02	0.54×	0.54×	2090.23	0.65×	0.65×	1.19	1.15×
s6669	2693	329	4.90	0.43×	0.41×	2.63	0.26×	0.24×	3276.85	0.55×	0.51×	0.99	2.07×
s9234	2052	330	1.61	0.62×	0.59×	0.99	0.67×	0.59×	2373.78	0.69×	0.59×	0.97	1.74×
s13207	2646	460	1.84	0.47×	0.44×	0.89	0.99×	0.87×	3122.31	0.70×	0.60×	2.28	2.61×
s15850	3423	594	2.76	0.50×	0.46×	1.66	0.66×	0.60×	3945.05	0.70×	0.61×	5.47	1.15×
s35932	10371	1290	0.73	1.04×	1.04×	11.35	0.23×	0.21×	12579.67	0.51×	0.46×	15681.14	1.75×
s38417	10199	1593	1.50	0.89×	0.68×	6.36	0.43×	0.41×	11474.18	0.59×	0.54×	21.2	1.45×
s38584	12584	2077	2.14	0.53×	0.56×	5.63	0.66×	0.61×	14302.02	0.64×	0.59×	4558.66	1.60×
multiplier	44038	5821	19.35	0.46×	0.43×	98.49	0.11×	0.11×	53150.26	0.59×	0.53×	715.39	1.28×
voter	21166	2708	6.16	0.61×	0.56×	35.10	0.14×	0.13×	24943.35	0.57×	0.50×	38.88	1.77×
square	34052	5109	18.48	0.45×	0.36×	56.48	0.17×	0.16×	41201.01	0.64×	0.53×	1200.49	1.22×
arbiter	12674	1742	4.29	0.38×	0.39×	9.16	0.32×	0.31×	16514.88	0.55×	0.43×	27.84	2.28×
bar	5422	812	1.32	1.10×	1.09×	5.00	0.31×	0.29×	6640.16	0.58×	0.52×	17.90	2.69×
sin	9655	1549	17.84	0.49×	0.42×	16.41	0.17×	0.15×	11922.65	0.64×	0.53×	159.36	1.19×
Average	15532	2248	6.08	0.59×	0.54×	19.17	0.38×	0.35×	18473.62	0.63×	0.56×	1180.24	1.87×
Average	13286.46	1893.56	5.80	0.20×	0.20×	23.94	0.23×	0.21×	15879.09	0.06×	0.06×	3563.96	0.54×

Table 6.2: Delay, Power and Cell Area Ratios of Flash-based Digital Circuits (with and without Optimization) Relative to Their CMOS Counterparts ($K = 3$)

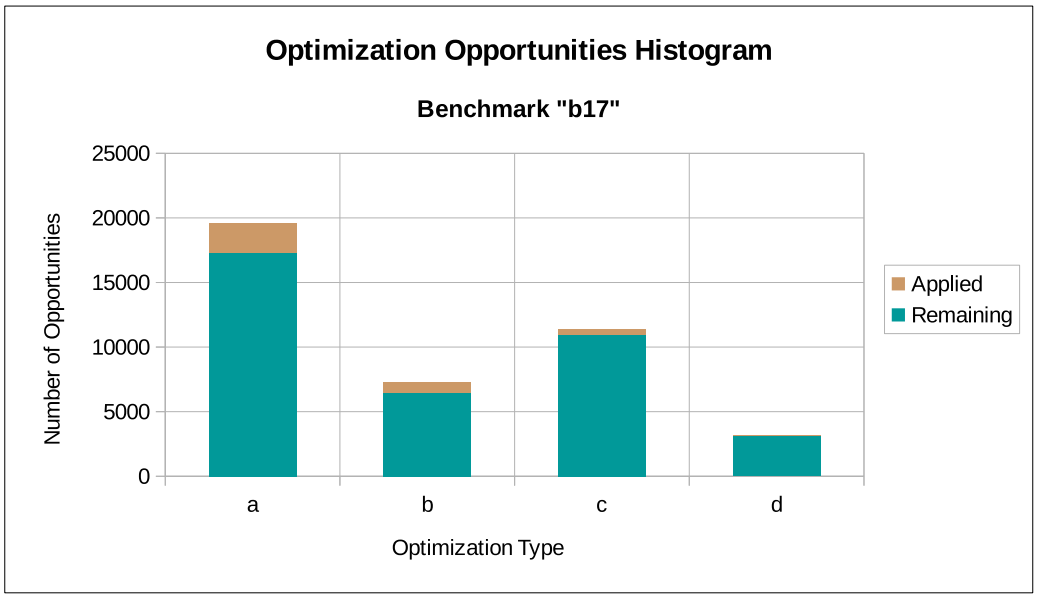


Figure 6.5: Optimization Opportunities Histogram for Benchmark ”b17”

For benchmark ”b17”, we tracked the number of candidate optimizations generated by our algorithm as well as the number of optimizations that have been applied to the design. Figure 6.5 shows the histogram of the optimization opportunities for benchmark ”b17”. The number of candidate optimizations generated for benchmark ”b17” were, 19605 of type A, 7243 of type B, 11378 of type C and 3216 of type D. Out of these candidate optimizations, our CCB algorithm applied 2302 of type A, 778 of type B, 461 of type C and 69 of type D. Figure 6.6 shows the histogram of the optimization opportunities for benchmark ”b14”. The number of candidate optimizations generated for benchmark ”b14” were, 3393 of type A, 1518 of type B, 2399 of type C and 768 of type D. Out of these candidate optimizations, our CCB algorithm applied 379 of type A, 216 of type B, 120 of type C and 25 of type D.

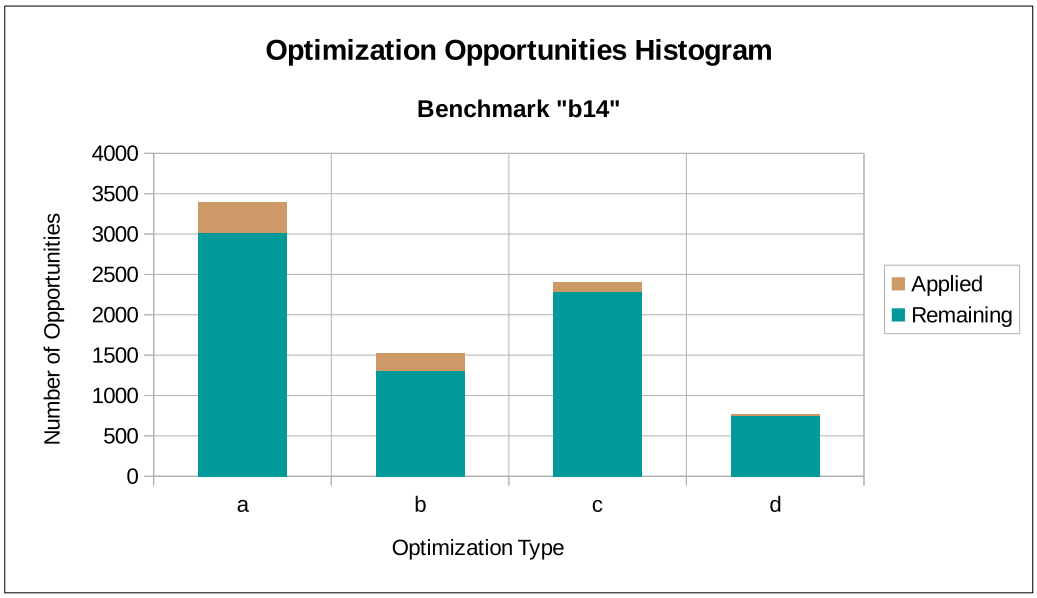


Figure 6.6: Optimization Opportunities Histogram for Benchmark ”b14”

From both these benchmarks, we note that type A and B optimizations are successful more often than type C or D optimizations. This is reasonable since type C and D have exactly one FLA (FLA_7 in particular) to which the cubes may be moved. On the other hand, type A and B optimization potentially have a larger number of FLAs to which the cube may be moved.

VI.5 Chapter Summary

The flash-based implementation of Chapter V did not consider don’t-care optimization. In this chapter, we presented a SAT-based optimization technique that implicitly uses the CODC of a multi-level logic network. Unlike other don’t-care optimization techniques presented in the past, our technique performs post-technology mapping optimiza-

tion which yields direct improvements in result quality as compared to pre-technology mapping optimization. Also, we optimize a cluster of nodes at once, instead of optimizing nodes one at a time. We characterized our implementation using 20 standard benchmarks, and shown that our optimization yields ∼47% lower delay, ∼64% lower power and ∼42% lower cell area compared to a CMOS standard cell implementation. This is a reduction of ∼10% in delay, ∼5% in power, ∼15% in energy and ∼8% in cell area compared to the flash-based design implemented without our optimization engine. We also reported that optimization engine incur a modest 27% runtime increase over the flash-based CAD flow, without optimization.

CHAPTER VII

PLA-LIKE, FLASH-BASED DIGITAL LOGIC CELL IMPLEMENTATION

This chapter focuses on the implementation of flash-based digital circuit at the cell-level using a variation of the *flash cluster* (FC) presented in Chapter IV. This chapter presents a cell structure that implements flash-based digital circuits in a manner that is similar to programmable logic arrays (PLAs). We call the flash-based cell presented in this chapter, a *PLA-like flash cluster* (PFC). Unlike the FC design approach, PFC design approach takes advantage of cube sharing (similar to a PLA), and is able to further reduce the physical area of the flash-based designs over the previously proposed approach. Similar to the FC, the PFC uses the ability to modify the threshold voltage of flash devices in order to implement combinational logic circuits. The PFC is fully compatible with the conventional sequential elements used in digital design. In this chapter, we compare the PFC design approach to both the CMOS standard cell design approach as well as the FC design approach.

VII.1 Background

Programmable logic arrays (PLAs) [3] are simple circuit structures that are used to implement digital circuits. A PLA can implement a combinational logic function $F_{m,n}$, where m is the number of inputs and n is the number of outputs. PLAs can be very

compact in area and are regular in structure, making them very appealing for digital circuit applications. A PLA consists of an AND-plane and an OR-plane as shown in Figure 7.1. The AND-plane implements the cubes of the function $F_{m,n}$. The input to the AND-plane is all the inputs of the PLA as well as their complements. Hence, the AND-plane has 2 $\times$ m vertical lines. The OR-plane combines the cubes constructed in the AND-plane to generate the outputs of the PLA. Hence, the OR-plane has n vertical lines. This structure allows the PLA to share cubes between outputs, which reduces the overall circuit area as well as its power dissipation.

Inputs			Outputs	
A	B	C	X	Y
0	-	1	1	0
1	1	1	1	1
-	0	-	0	1
0	1	-	1	0

Table 7.1: Espresso-MV Minimization Output of the Function $G_{3,2}$

Consider a logic function $G_{3,2}$, which has 2 outputs X and Y as described in Equations VII.1 and VII.2. Since $G_{3,2}$ is a function with multiple outputs, it can be minimized using Espresso-MV [29, 30] (which performs multi-valued logic minimization). Table 7.1 shows the output after performing multi-valued minimization on the function $G_{3,2}$. Note that the input literals in Table 7.1 are represented using ”0”, ”1” or ”-”, where a ”0” or a ”1” represent the input literal polarity, and a ”-” represents a don’t care literal. Also the output literals are represented using a ”1” or ”0”. A ”1” in the output of Espresso-MV

means that the corresponding input cube is in the on-set of that output. A ”0” in the output of Espresso-MV means that the minterm (or cube) does not control that output.

$$X = \overline{A}C + \overline{A}B + ABC \tag{VII.1}$$

$$Y = ABC + \overline{B} \tag{VII.2}$$

The minimized function $G_{3,2}$ shown in Table 7.1 can be written as shown in Equation VII.1 (for the first output ”X”) and Equation VII.2 (for the second output ”Y”).

Figure 7.1 shows the PLA that implements the function $G_{3,2}$ shown in Table 7.1.

VII.2 Previous Work

There has been no work in the past, in the area of PLA-like structures to implement digital circuits using flash transistors.

The work presented in Chapter IV (and also in [69]), implemented flash-based binary-valued, digital circuits using structures called flash clusters (FCs). The FC implements a logic function $F_{m,n}$, which has m inputs and n output minterms. The FC consists of an array of NAND flash-like pulldown stacks, each implementing a cube. The FC outputs are generated separately, without sharing input cubes across output minterms. This is done by separately running Espresso on the input minterms that map to each output minterm of $F_{m,n}$. This results in potential duplication of minterms or cubes in the FC, since minterms and cubes are not shared across output minterms. This generally yields a faster design, but

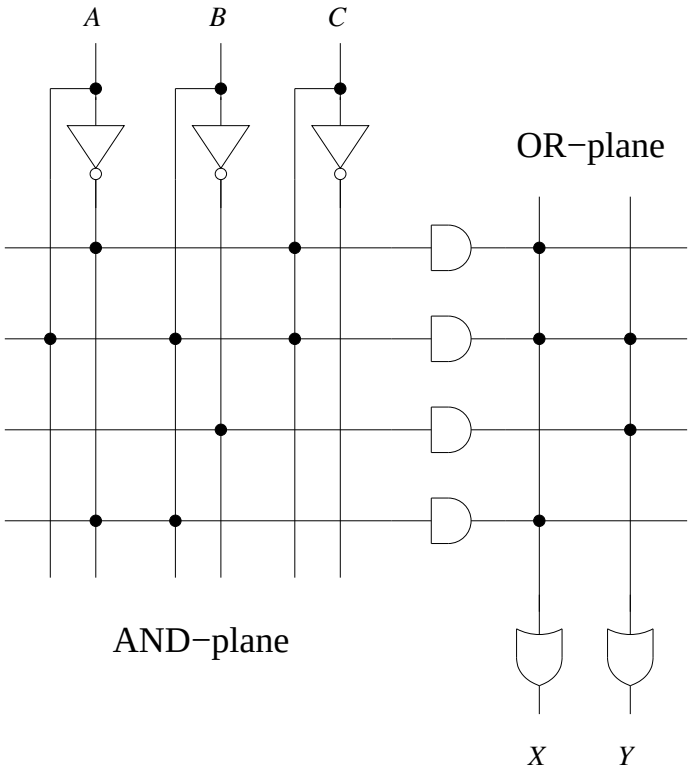


Figure 7.1: Example PLA Structure That Implements the Function $G_{3,2}$

at the cost of increased power and circuit area.

The key difference between the work in Chapter IV (also [69]) and the work presented in this chapter is in the way the function $F_{m,n}$ is minimized. In the FC, the input minterms of each output minterm of $F_{m,n}$ are grouped and minimized separately. An n output logic function $F_{m,n}$ thus requires 2^n such Espresso runs. The FC thus eliminates cube sharing between different output minterms, resulting in degraded area, and better power. In contrast, in the approach presented in this chapter, we perform the logic minimization on the entire function $F_{m,n}$ monolithically (using Espresso-MV). This causes a reduction in the number of cubes that represent the logic function $F_{m,n}$ (due to cube/minterm shar-

ing). This translates into reducing the area of the design, with a power and energy penalty. Thus our approach provides the designer another tool to explore the delay, power and area tradeoff along with the FC (in Chapter IV and in [69]).

VII.3 Approach

VII.3.1 Overview

The conversion process used to implement flash-based digital circuits for PFC-based designs is similar to that used to construct FC-based designs described in Chapter IV. To understand the top-level flow, the reader can refer to Section IV.3.1. For additional details about the entire CAD flow used to build and optimize flash-based digital circuits, the reader can refer to Chapter V and Chapter VI. This chapter focuses only on the flash-based PFC design, at the cell-level.

Each PFC implements an m input, n-output logic function ($F_{m,n}$). The circuit details of a PFC are described in Section VII.3.2. Based on experimental results, we choose the number of outputs ($n = 3$), and the number of inputs ($m = 6$) for our implementation of the PFC. Similar to the FC described in Chapter IV, the PFC consists of several *PLA-like flash output groups* (PFOGs), which in turn are made up of several *PLA-like flash logic bundles* (PFLBs). Details of the PFC structure will be discussed in Section VII.3.2.

VII.3.2 PLA-like Flash Cluster (PFC) Circuit Design

The PFC (Figure 7.2) is a dynamic structure that implements any Boolean logic function $F_{m,n}$, where m is the number of inputs and n is the number of outputs. The

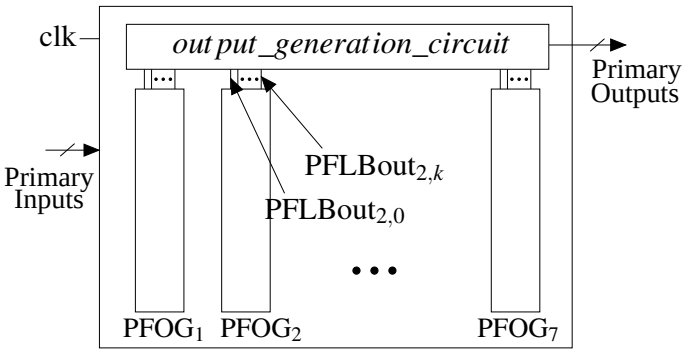


Figure 7.2: PLA-like Flash Cluster

function $F_{m,n}$ is realized using a PFC after programming the flash transistors in the PFC to implement the cubes representing $F_{m,n}$. The PFC is also equipped with the required logic for programming the threshold voltages of their floating gate devices (not shown in Figure 7.2).

Similar to PLAs, to implement a PFC, the function $F_{m,n}$ is first minimized using Espresso-MV [29]. Since the PFC is a precharged circuit, the outputs are precharged to VDD (logic "1") during the precharge cycle, and they get discharged to GND (logic "0") based on which input is applied to the circuit. Therefore, the cubes that should be implemented in the circuit, are the off-set cubes. Hence we use Espresso-MV to minimize the function $F_{m,n}$ against its off-set.

Internally, the PFC consists of multiple *PLA-like flash output groups* (PFOGs) and an *output generation circuit* as shown in Figure 7.2. Consider the minimized logic function of a PFC, $F_{m,n}$. After minimization, it consists of several cubes, each of which has an output label or state (Column 2, Table 7.2). In order to reduce the number of transistors in

the output generation circuitry, we group the cubes of $F_{m,n}$ by their output state or label as shown in Column 1 of Table 7.2. Each group is called a PLA-like flash-based output group (PFOG). If the cubes of $F_{m,n}$ were not grouped by PFOGs, each cube would need separate output generation transistors, thereby increasing area and power. As shown in Table 7.2, each PFOG controls one or more of the outputs of the function $F_{m,n}$. In particular, since the PFC is a dynamic circuit, each PFOG discharges one of more of the outputs of the function $F_{m,n}$. For example, as shown in Table 7.2, for a function with 3 outputs ($n = 3$), $PFOG_5$ causes the first and last outputs to discharge. However, $PFOG_5$ does nothing to the second output (i.e. it does not discharge the second output), similar to a PLA. Also $PFOG_0$ does not affect any of the outputs. Therefore, there are $2^n - 1$ PFOGs in total, in every PFC. We will show how the PFOGs ($PFOG_1, PFOG_2, \cdots, PFOG_7$) are constructed next.

$PFOG_i$	Output State	Effect on $F_{m,3}$ Outputs		
		First	Second	Third
$PFOG_0$	000	N/A	N/A	N/A
$PFOG_1$	001	N/A	N/A	Discharged
$PFOG_2$	010	N/A	Discharged	N/A
$PFOG_3$	011	N/A	Discharged	Discharged
$PFOG_4$	100	Discharged	N/A	N/A
$PFOG_5$	101	Discharged	N/A	Discharged
$PFOG_6$	110	Discharged	Discharged	N/A
$PFOG_7$	111	Discharged	Discharged	Discharged

Table 7.2: Association of $PFOG_i$ to Output State and Effect on $F_{m,3}$ Output When Minimizing Against Off-sets

Suppose that we are implementing the function $F_{m,n}$ where $m = 6$ and $n = 3$. Also assume that the outputs of the function $F_{m,n}$ are (f, g and h). After running Espresso-MV on the function $F_{m,n}$ we collect all the input cubes that correspond to each output states. Table 7.2 shows a list of output states. These states simply indicate which outputs are controlled by the corresponding input cubes Now, if the output state $< 010 >$ has 2 input cubes $< 011011 >$ and $< 110-11 >$, then $PFOG_2$ implements these two input cubes. Note that since we minimize a function $F_{m,n}$ against its off-set in Espresso-MV, the output state $< 010 >$ means that the corresponding input cubes discharge the second output of the function $F_{m,n}$ and does not affect the rest of the outputs (first and third) as illustrated in Table 7.2. Similarly, the output state $< 111 >$ ($PFOG_7$) means that the corresponding input cubes discharge all the outputs. The output state $< 000 >$ does not affect any of the outputs, and is never produced by Espresso-MV. It is listed in Table 7.2 completeness of the discussion.

We note that the output states only specify which output is pulled down, and not necessarily the final state of all the outputs of $F_{m,n}$. To illustrate this, assume that a function $G_{6,3}$ after minimization using Espresso-MV has two output states which are implemented in a PFC. The output state $< 001 >$ (which is implemented in $PFOG_1$) has one input cube $< 0-011- >$ and the output state $< 100 >$ (which is implemented in $PFOG_4$) has one input cube $< 00011- >$. Now assume that the input $< 000111 >$ is applied to the PFC that implements the function $G_{6,3}$. The final output of the PFC is $< 010 >$, since $PFOG_1$ pulls down the 1st output and $PFOG_4$ pulls down the 3rd output.

Next, we will discuss the structural details of an PFOG, and then the details of the output generation circuit.

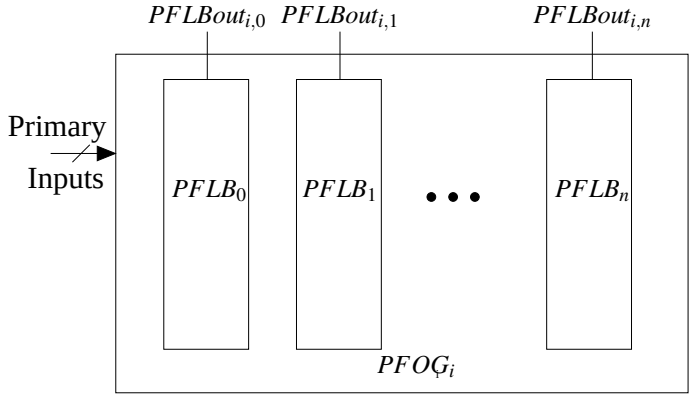


Figure 7.3: PLA-like Flash Output Group i (PFOG$_i$) Structure

VII.3.2.1 PLA-like Flash Output Group (PFOG)

The structure of the PFOG used in the PFC is similar to that used in the FC (refer to Section IV.3.2.1). Figure 7.3 shows a block diagram of the structure of $PFOG_i$. As shown in this figure, each PFOG consists of multiple *PLA-like flash logic bundles* (PFLBs). For the i^{th} PFOG, we refer to the number of cubes that this PFOG implements as its *cubes per output group* or CPG_i. The number of PFLBs that exist in $PFOG_i$ is $\lceil \frac{CPG_i}{CPB} \rceil$, where CPG_i is the number of cubes in $PFOG_i$, and CPB is the maximum number of cubes that can be implemented in any single PFLB. We swept the value of CPB, and found that CPB=3 yielded the best electrical characteristics. Notice that while CPG_i is determined by the logic function $F_{m,n}$, CPB is a design parameter that can be optimized to improve circuit delay, power, energy and physical area. The number of outputs of $PFOG_i$ is equal to the

number of PFLBs in $PFOG_i$, namely $\lceil \frac{CPG_i}{CPB} \rceil$. Unlike the FLA used in Chapter IV and in [69], more than one PFLB output ($PFLBout_{i,k}$) can pull down when an input is applied to the PFOG inputs, due to cube sharing.

VII.3.2.2 PLA-like Flash Logic Bundles (PFLB)

The structure of the PFLB used in the PFC is similar to that used in the FC shown in Section IV.3.2.2. Figure 7.4 shows the circuit details of an PFLB. An PFLB consists of a number of NAND flash-like pulldown stacks that share the same output. Each pulldown stack implements one cube of $F_{m,n}$. The maximum number of NAND flash-like pulldown stacks in each PFLB is CPB, where CPB is the maximum number of cubes that can be efficiently implemented in an PFLB. Since the number of cubes implemented by $PFOG_i$ is not necessarily a multiple of CPB, the last PFLB in $PFOG_i$ may have a smaller number of pulldown stacks than CPB.

Each one of the pulldown stacks has m flash transistors and 1 regular NMOS transistor (as shown in Figure 7.4), where m is the number of inputs of the function $F_{m,n}$. The flash transistors are programmed to implement cubes, while the regular transistors have a dual purpose. Since the flash-based implementation proposed in this work is based on dynamic logic, both a precharge and an evaluate transistor are needed. The shared regular PMOS transistor shown at the top of Figure 7.4 (M_{pch}), serves as the precharge transistor for all pulldown structures of the PFLB. When it turns on, it pulls up $PFLBout_{i,k}$ during the precharge (low) phase of the clock signal (clk). The lines VSP_0 to VSP_q are connected to ground during normal operation, to allow the NAND stacks to pull down when they eval-

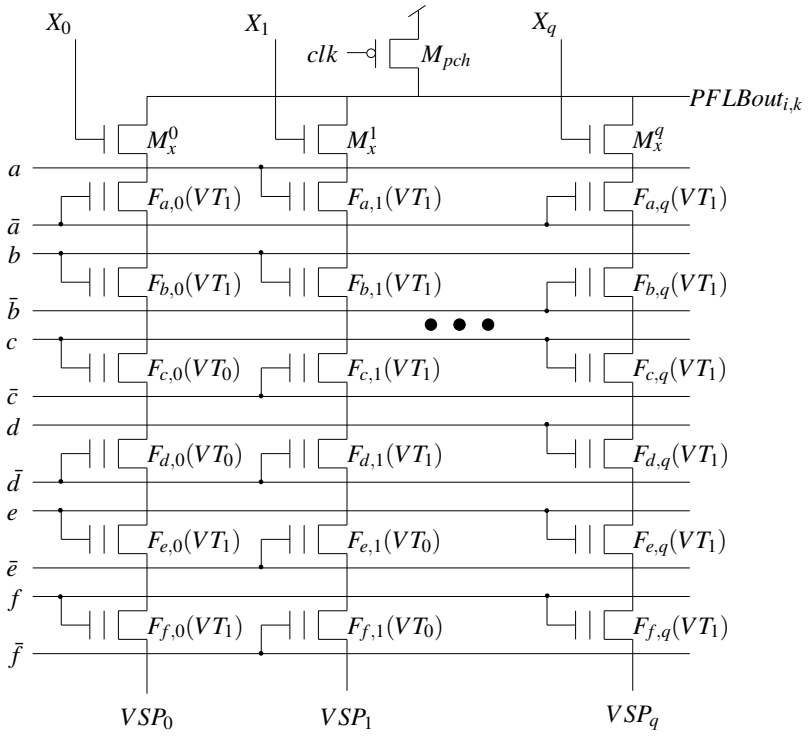


Figure 7.4: PLA-like Flash Logic Bundle i, k (PFLB$_{i,k}$)

uate. They also have a special purpose during programming, which will be discussed in Section VII.3.3. The regular NMOS transistors (M_x^i) shown at the top of each stack in Figure 7.4 serve a dual purpose. During chip operation they are used as the evaluate transistors which are off during the precharge (low) phase of *clk* and only turn on during the evaluate (high) phase of *clk*, to allow the pulldown stack to evaluate the output $PFLBout_{i,k}$. The NMOS transistors (M_x^i) are also utilized during the programming operation of the NAND flash stack. Programming will be discussed in detail in Section VII.3.3.

The flash transistors in each of the NAND flash-like stacks are programmed to realize a cube. Figure 7.5 shows a diagram of the V_T levels used in our PFC implementation.

On the left side of the figure, we show the voltage levels used to drive the flash transistors (namely VIH = VDD and VIL = GND). On the right side, we show the V_T levels. The erase threshold voltage (VT_0) is used for a flash transistor when it is intended to be always on (i.e. when the corresponding literal in the cube is not present). The program threshold voltage (VT_1) is used when the corresponding literal is present in the cube. For example, the left-most stack of Figure 7.4 implements the cube $\overline{a}bef$. Note that the transistors $F_{c,0}$ and $F_{d,0}$ are programmed to a threshold voltage VT_0 which is below GND (see Figure 7.5). The threshold voltage VT_0 is also referred to as the *erase* threshold. Therefore, these two transistors are *on* irrespective of the values of the signals c and d. Now, transistors $F_{a,0}$, $F_{b,0}$, $F_{e,0}$ and $F_{f,0}$ are programmed to a threshold voltage VT_1, which is between VDD and GND (see Figure 7.5). This means that these devices turn on only when their gate signal (respectively $\overline{a}$, b, e and f are greater than VT_1. As a consequence, the left-most stack of Figure 7.4 implements the cube $\overline{a}bef$.

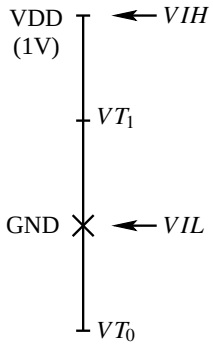


Figure 7.5: Flash Transistor Threshold Voltages Used in a PFC

Similarly, the second stack of Figure 7.4 implements the cube $ab\overline{c}\overline{d}$. The rightmost stack of Figure 7.4 implements the cube $\overline{a}\overline{b}cdef$. Note that all of its transistors are programmed to the VT_1 threshold, which is why it implements a minterm in the m-input space of $F_{m,n}$.

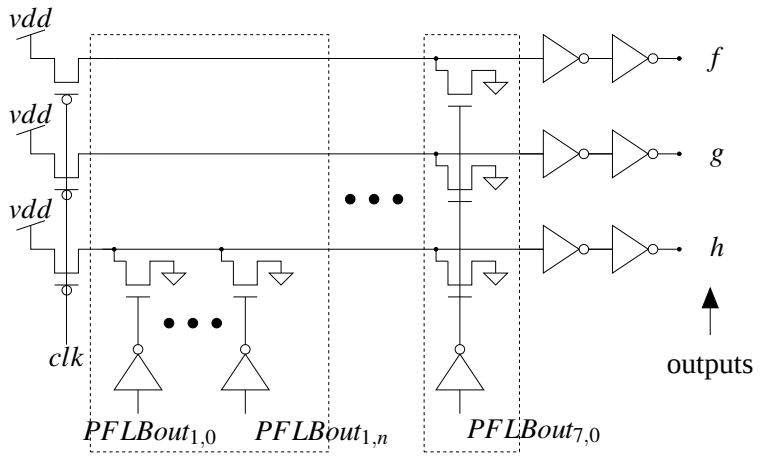


Figure 7.6: Flash Output Generation Circuit

VII.3.2.3 Output Logic

The output generation circuit in a PFC is different from the output generation circuit in an FC (Chapter IV and [69]) due to the fact that cube sharing is allowed between outputs in the PFC, and not allowed in the FC. This means that in the PFC, for some input minterm M_i, two or more PFOGs can pull down, which is a condition that was impossible in the FC design.

The output generation circuit used in the PFC is a dynamic structure. It consists of PMOS transistors precharging each of the output lines (depicted in Figure 7.6 by the

horizontal lines drawn across the figure). These output lines are then buffered in order to drive a load equivalent to 3 PFC input loads. For each output state in $F_{m,n}$ (see Table 7.2), if the value of the j^{th} output is 1, an NMOS transistor is inserted for the j^{th} output line. Otherwise, the output line will not be affected. Since the outputs of the PFOGs are active low, we insert an inverter for each $PFLBout_{i,k}$ before driving the gates of the pulldown NMOS devices in the output logic.

The inverted output of each one of the PFOGs in the PFC drives a group of NMOS transistors that evaluate the output lines of the output generation circuit. Note that due to cube sharing, some outputs may be discharged by multiple PFOGs. In other words, at the beginning of the evaluate cycle, all the output lines are precharged to VDD, then depending on the applied input minterm (M_i), these output lines may discharge due to more than one PFOGs pulling down.

As discussed at the end of Section VII.3.2, different outputs may be pulled down by different PFLBs, when an input minterm M_i is applied to the PFC.

VII.3.3 Programming the PFC

The programming of a PFC can be done in the same fashion as the programming of the FC (shown in Section IV.3.3).

VII.4 Experiments

In this section, we start by presenting the simulation environment used to evaluate our PFC-based digital design approach. We follow with a discussion about the imple-

mentation details of the PFC design. Finally, we present a discussion of the results of the FPC-based design.

VII.4.1 Simulation Environment

In this section, we compare the PFC design approach to both a CMOS standard cell-based design approach as well as the FC-based design approach presented in Chapter IV and [69]. All of the designs are implemented in a 45nm process technology. We use Synopsys Design Compiler [56] to synthesize and map the CMOS designs to the industry grade 45nm Nangate FreePDK45 Open Cell Library [54, 55]. The mapped designs were simulated using the Synopsys HSPICE [56] circuit simulation tool. We use the 45nm PTM [57] model card to model the CMOS transistors. We implemented the FC based approach for comparison purposes. The PFC-based designs are generated using our in-house tool chain. The flash transistors in the PFC-based designs are modeled using flash model cards generated using the same model card regression approach as described in Chapter IV and [69]. The FC and PFC-based designs are simulated in HSPICE and the correctness of their logical operation is verified through exhaustive simulations. The operating supply voltage for both the flash-based and CMOS standard cell-based designs is 1V. Flash-based designs use higher programming voltages (10V-20V) *only during programming*. Custom layouts for the PFC-based designs were fashioned using Cadence Virtuoso [65] using design rules obtained from the ITRS reports [59]. The physical area of flash-based designs are compared to the cell area of the CMOS-based designs. The PFC-based design approach is evaluated through 20 *randomly generated* circuit designs and compared to a

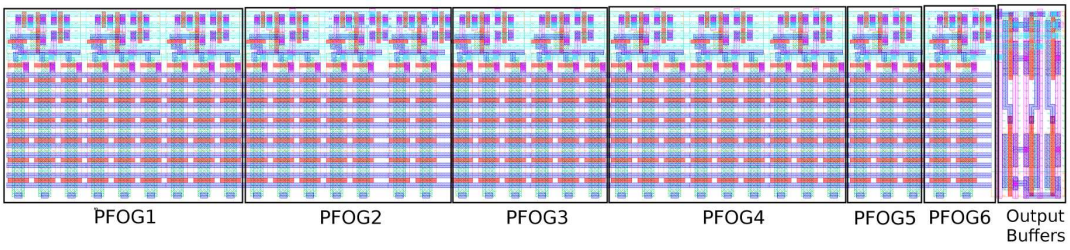


Figure 7.7: Example Layout View of a PFC (des00)

CMOS-based implementation of the same designs, and an FC-based approach as well.

VII.4.2 Flash-based Implementation Details

The logic functions implemented in the CMOS-based, FC-based and the PFC-based digital circuits have 6 inputs (m = 6) and 3 outputs (n = 3). These values were found to achieve the best delay, power, energy and physical area for the flash designs. The results we present are a comparative study over 20 randomly generated functions (des00 to des19) implemented in the CMOS standard cell-based approach, FC-based approach and our PFC-based approach. We used CPB = 3 for the FC and the PFC designs. Also, the threshold voltages used in our flash-based designs are (VT_0 = -0.5 V) and (VT_1 = 0.5 V).

VII.4.3 Results and Analysis

We report the delay (including precharge delay), power, energy and physical area ratios in Table 7.3. These results are obtained from implementing the 20 randomly generated logic functions using the PFC-based design approach and compared to the CMOS standard cell based approach. For the PFC-based results, the precharge delay is 39% of the total delay, on average. The delay reported in the table (D_{max} Ratio) is ratio of the maximum delay of any transition seen at any primary output of the circuit. Since the flash-

Circuit	D_{max} Ratio	P_{avg} Ratio	Eng Ratio	Cell Area Ratio
des00	0.81×	0.37×	0.30×	0.47×
des01	0.81×	0.35×	0.29×	0.46×
des02	0.87×	0.37×	0.32×	0.49×
des03	0.74×	0.40×	0.30×	0.44×
des04	0.89×	0.42×	0.38×	0.49×
des05	0.76×	0.36×	0.27×	0.44×
des06	0.96×	0.38×	0.36×	0.49×
des07	0.81×	0.39×	0.32×	0.46×
des08	0.87×	0.38×	0.33×	0.42×
des09	0.87×	0.36×	0.31×	0.43×
des10	0.93×	0.43×	0.40×	0.44×
des11	0.87×	0.42×	0.37×	0.42×
des12	0.85×	0.41×	0.35×	0.39×
des13	0.89×	0.43×	0.38×	0.49×
des14	0.80×	0.35×	0.28×	0.43×
des15	1.01×	0.40×	0.40×	0.51×
des16	0.88×	0.38×	0.33×	0.51×
des17	0.77×	0.40×	0.31×	0.49×
des18	0.77×	0.37×	0.29×	0.44×
des19	0.74×	0.41×	0.31×	0.44×
Average	0.85×	0.39×	0.33×	0.46×
Stdev	0.07×	0.03×	0.04×	0.03×

Table 7.3: Delay, Power, Energy and Cell Area Ratios of PFC-based Digital Circuits Relative to Their CMOS Standard Cell-based Counterparts

based implementation is dynamic, we accounted for the precharge delay in the reported delay shown in the table. As shown in the table, the delay of the flash-based digital circuits ranges from 0.74× to 1.01× of the CMOS standard cell-based digital circuit delay, with an average of 0.85×. The delay of the PFC is substantially similar to that of the FC presented in Chapter IV and [69]. The standard deviation of the results is shown in the bottom row of Table 7.3. The standard deviation in delay (8%), power (2%), energy

(3%) and physical area (3%) are relatively low, and demonstrate that the characteristics of digital design implemented using a PFC-based approach are quite predictable over a large number of designs.

Table 7.3 also reports the average power dissipation (0.39× of CMOS) and energy utilization (0.33× of CMOS) when implementing the digital circuits using our flash-based logic compared to CMOS standard cell-based implementation. The PFC has ∼11% higher power dissipation and energy consumption than those of the FC. This is because the FC does not allow cube sharing across outputs, unlike the PFC. Cube sharing also results in the evaluation of multiple pulldown stacks in the PFC, which increases the power dissipated in the evaluate and the precharge cycles of the clock, and hence, increases the average power dissipation and energy consumption.

We also report the area ratio of both implementations. The area reported for the CMOS standard cell-based implementation is the sum of physical cell areas, while the area of our flash-based approach is the layout area obtained from layout generation experiments. In this sense, the CMOS standard cell area is a lower bound of the physical area, while the PFC area is the true physical area. Design rules for flash were obtained from the ITRS 45nm flash technology node [59]. Digital circuits implemented in a PFC use 0.46× the physical area of a CMOS-based design, on average. This is ∼18% lower than the area ratio of the FC (in Chapter IV and [69]). This is expected because unlike the FC, our PFC design exploits cube sharing between outputs. In Figure 7.7, we show the representative layout of a cluster of the PFC for the design des00. Note that the layout of the PFC shown

in Figure 7.7 does not include neither of $PFOG_0$ nor $PFOG_7$. $PFOG_0$ is not implemented in the PFC of the design des00 since $PFOG_0$ is implemented by the precharge state. However, $PFOG_7$ is not implemented only because the design des00 does not contain any input cubes that control all the outputs (which are implemented in $PFOG_7$).

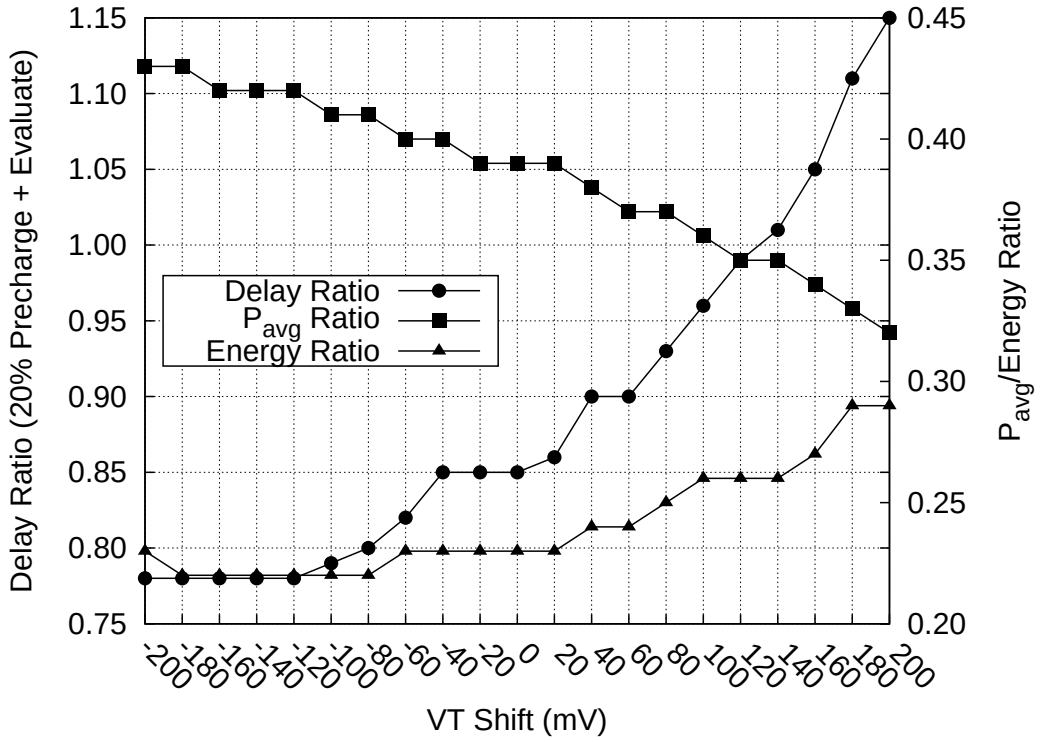


Figure 7.8: Delay, Power and Energy of the Flash-based Designs as V_T is Shifted.

Flash-based digital circuits have the ability of tuning their delay, power and energy characteristics. This is done by shifting the V_T of the flash transistors in the circuit. The ability to shift V_T offers the flash-based digital circuits huge advantages over the traditional CMOS standard-cell based circuits when it comes to speed binning at the factory,

aging mitigation and performing post-manufacturing ECOs. Figure 7.8 shows the average delay, power and energy of the flash-based digital designs as their V_T is modified around the nominal V_T value (which is indicated by a "V_T shift" value of 0 mV). The delay in Figure 7.8 is the sum of the evaluate and the precharge delays. Figure 7.8 shows that the PFC delay improves with a negative V_T shift, allowing the manufacturer to do speed adjustment in the factory, or aging mitigation in the field. The speed improvement is accomplished by an increase in power as expected.

VII.5 Chapter Summary

Flash transistors have some important properties that distinguish them from CMOS transistors, such as the ability to shift the threshold voltage of the flash devices, as well as their small input capacitance and compact area. In the past, these properties have been exploited to implement non-volatile memory. This chapter presented an approach to use flash transistors to implement digital circuits using a PLA-like circuit structure. We present the details of the circuit topology that we use in our PFC-based digital circuit approach. Our HSPICE simulations show that, averaged over 20 designs, our approach yields 0.85$\times$ the delay, 0.39$\times$ the power, 0.33$\times$ the energy utilization and 0.46$\times$ the physical area of the equivalent circuit implemented using CMOS standard cell-based design. The PFC design exhibits improved area ($\sim$18% smaller) than the FC design approach in Chapter IV and [69], at the cost of $\sim$11% increased power and energy consumption. The improvement in area is due to the fact that unlike the FC, the PFC design can share input cubes across

outputs, in a PLA-like fashion.

CHAPTER VIII
MULTI-VALUED, FLASH-BASED DIGITAL LOGIC CELL IMPLEMENTATION

In this chapter, we present a circuit implementation that uses flash transistors to implement multi-valued digital circuits. The flash transistors used in our implementation only need two threshold voltages. As a result they have high V_{gs} values which improves delays significantly, and have higher write endurance as well. We evaluate our design methodology through circuit simulations, and compare our results to a CMOS standard cell based approach as well as to the implementation of ternary-valued logic using flash transistors presented in Chapter III and [64]. Also, the proposed approach scales elegantly to multi-valued logic using more than three values as well. The circuit topology we utilize is a cluster of unprogrammed flash transistors arranged in a NAND flash-like configuration (we call these MVFCs), which are programmed in the factory to implement the desired logic function.

VIII.1 Background

There are two approaches to implement multi-valued digital circuits. The first approach, which was introduced in Chapter III and [64], uses flash transistors with three V_T levels to implement ternary-valued logic functions. In Chapter III, the structure that implements ternary-valued logic is called a ternary logic cluster (TLC). One could implement

four-valued (or more) flash-based digital circuits by using four (or more) V_T levels. However, increasing the number of V_T levels would result in a very expensive delay penalty, due to the fact that the V_{gs} values of the flash devices would be lowered.

The second approach, which is presented in this chapter, is based on the idea of implementing multi-valued logic using a one-hot implementation. In this implementation, instead of using multiple V_T levels to implement multi-valued logic, flash transistors will only be programmed to one of two V_T levels, the erase V_T level (we call it VT_0) and the program V_T level (we call it VT_1). This has many benefits, such as – a) using two V_T levels instead of three (or more) results in higher V_{gs} values, which improves the design speed, b) using *single-level cells* (SLCs) results in improved write endurance as well as tolerance to read and write disturbs, compared to the *multi-level cells* (MLCs) used in [64] and c) the scheme of this chapter generalizes to multi-valued logic with 4 or more values elegantly, with no V_{gs} reduction and speed penalties as exhibited by the TLCs (presented in Chapter III and [64]).

VIII.2 Previous Work

In Chapter III and [64], we used a structure called *ternary logic clusters* (TLCs), which use multiple threshold voltage levels to implement ternary-valued logic functions. Although this approach has benefits, the use of using multiple threshold voltages has some drawbacks. One of the issues that arise from using multiple threshold voltages is decreased write endurance, since programming a flash transistor becomes harder and requires

a longer process of V_T adjustment, thus degrading the write endurance of the device. Another issue is read and write disturbs, since the threshold voltage levels are closer to each other and hence a read or write disturb could change the threshold voltage of a victim cell sufficiently, thereby altering the proper operation of the circuit. Finally, using multiple threshold voltages between VDD and GND reduces the V_{gs} values applied to the flash transistors which lowers the currents flowing through the flash device when its on, resulting in increased delays. Also, the TLC structure used in Chapter III and [64] consists of NAND flash-like pulldown stacks. These stacks require 2 flash transistors in series per ternary-valued variable. So a 4-variable TLC would require 8 series transistors. The use of long series stack results in an additional increase in the delay of the design. In our work, we address these issues by using two threshold voltage levels only (one for program and one for erase), and by using pulldown stacks that only have 1 flash transistor per input variable. These changes in the design result in an increase in the speed of the multi-valued flash-based designs by more than $3\times$ compared to the speeds reported in Chapter III and [64]. However, this improvement in speed is accomplished by an increase in power and energy. Compared to CMOS standard cells, however, the MVFCs are faster (by 23%) and consume less energy (by 26%) and power (by 5%) as well.

In Chapter IV and in [69], we presented a flash-based design approach to implement binary logic circuits. In this implementation, we used a *flash cluster* (FC). The FC consists of an array of NAND flash-like pulldown stacks. Each stack has 6 flash transistors (one transistors per variable). The work of this chapter achieves lowered delays compared to

the FC-based designs approach, since we use only 4 flash transistors in series. The use of a shorter pulldown stack also increases the power of the MVFC design by 42%, compared to the FC.

VIII.3 Approach

VIII.3.1 Overview

The focus of this chapter is on the circuit details of the multi-valued flash cluster (MVFC). However, before describing the circuit details of the MVFC, we will briefly outline how a digital circuit block can be mapped into a flash-based block using a network of MVFCs. Our MVFC structure is capable of implementing multi-valued logic with an arbitrary number of logic values. However, in this chapter, we focus on ternary-valued logic in order to compare the performance of our approach to that of the TLCs in Chapter III and [64]. In this chapter, each MVFC implements a ternary logic function $H_{r,s}$, where r is the number of inputs and s is the number of outputs. Our simulations have shown that choosing $r = 4$ and $s = 2$ yields the best results. After describing the mapping procedure, we describe the MVFC structure, the configuration of the MVFC cell and the procedure used for MVFC programming.

VIII.3.2 Flash-based Design Conversion

The flash-based design conversion of multi-valued flash clusters is similar to that of the TLC described in Chapter III and [64]. However, the MVFC approach supports k-valued logic in general. Multi-valued flash clusters implement s-output (multi-valued)

functions with up to r multi-valued inputs ($H_{r,s}$). Therefore, to implement an entire digital circuit block using MVFC cells, we need to represent the digital circuit block using functions of the type $H_{r,s}$. This is done starting with a technology-independent logic netlist, and grouping the logic nodes into clusters. These clusters of logic are constructed so as to meet the criteria of the number of inputs and number of outputs that can be implemented by an MVFC. Note that the technology-independent logic netlist represents binary logic, and hence, we convert the binary logic functions into multi-valued logic functions as described above. The process of converting binary logic functions into ternary-valued logic functions is described next (since we focus on ternary-valued MVFCs in this chapter) and is followed by a discussion on mapping of a technology-independent logic netlist into an MVFC-based design.

VIII.3.3 Multi-valued Logic Flash-based Design Conversion

In this section, we show how we convert a binary logic function into a multi-valued logic function (ternary-valued in our implementation). We perform this conversion in two steps. The first step is to implement the entire design using an interconnected network of MVFCs, and the second step is to identify the MVFC cells that implement the logic at the I/O interfaces of the digital design and appropriately configure them as shown later in this section.

In this chapter we need to convert the binary functions in the logic cluster into ternary-valued logic functions to be implemented using our MVFC cell. In our implementation, we convert each binary logic function with u binary inputs (which has

2^u binary minterms) into a v-input ternary-valued logic function (which has 3^v ternary-valued minterms). This is done by mapping the 2^u binary minterms into 3^v ternary-valued minterms. Since, there is no value of $u > 0$ and $v > 0$ that results in a solution for the equation $2^u = 3^v$, we have to choose u and v such that $2^u < 3^v$ (to be able to map all the binary minterms). However, this choice will result in *unused* ternary minterms. It was found that choosing $u = 3$ and $v = 2$ results in the least number of unused ternary-valued minterms. As a result, when implementing a digital design using MVFC cells, we cluster binary logic nodes such that the numbers of inputs and outputs that are multiples of 3. This yields a ternary-valued function with inputs and outputs that are multiples of 2. This issue of unused minterms can be entirely avoided if we implement multi-valued logic with k values, where $k = 2^w$. In such a case, mapping a binary function into a multi-valued function is straightforward and can be done by encoding w binary bits in each 2^w-valued multi-valued digit. As mentioned earlier, the reason we are implementing ternary-valued logic is to compare the multi-valued implementation approach of this chapter to that presented in Chapter III and in [64], which also discusses a ternary-valued logic design approach.

From a block-level perspective, the mapping of a binary logic design into a ternary-valued logic design has been discussed earlier in Section III.3.2. Each MVFC implements a ternary-valued logic function $H_{r,s}$, where $r = \frac{2}{3}m$ and $s = \frac{2}{3}n$. Our simulations have shown that choosing $m = 6$, $n = 3$ (which results in $r = 4$ and $s = 2$) yields the best results.

Similar to TLC-based designs (presented in Chapter III), our MVFC-based design approach can natively interface with binary logic *without* the need for specialized circuits

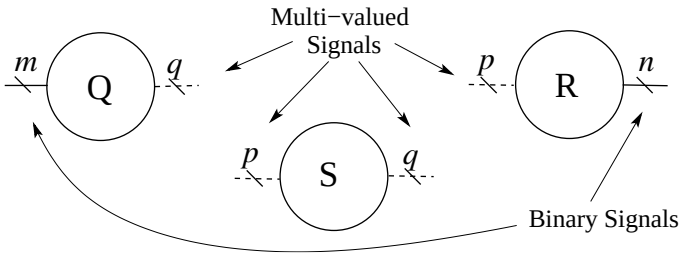


Figure 8.1: MVFC Types in Our Implementation.

dedicated to encode the binary signals into multi-valued signals. This reduces design overhead, complexity and area utilization. Figure 8.1 shows the types of MVFCs used in our implementation. In the figure, we represent binary signals with a solid line, and multi-valued (ternary-valued in our implementation) with dashed lines. The MVFCs used at the input interfaces of a logic block have binary input and multi-valued output (type Q) as shown in Figure 8.1. We use MVFCs that have ternary-valued inputs and binary outputs (type R) at the output interfaces of our multi-valued flash-based blocks. The remaining MVFCs used in our design approach have ternary-valued inputs and outputs (type S) and are used internally. Note that an MVFC can have a mix of binary and ternary-valued inputs (or outputs) in order to implement clusters that have some of their inputs (or outputs) being binary and the remaining inputs (or outputs) being ternary-valued.

VIII.3.4 Multi-valued Flash Cluster Circuit Design

The flash-based design approach presented in this chapter maps multi-valued logic functions (ternary-valued in our implementation) into multi-valued flash clusters (MVFCs). Figure 8.2 shows the structure of an MVFC. MVFCs are dynamic circuit struc-

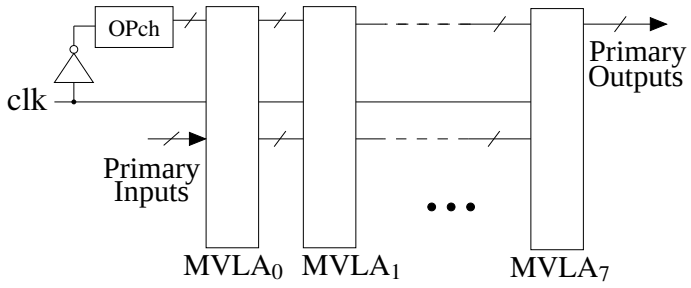


Figure 8.2: Multi-valued Flash Cluster (MVFC)

tures that implement the cubes of a multi-valued logic function $H_{r,s}$, where r is the number of inputs and s is the number of outputs. Recall that the multi-valued logic function $H_{r,s}$ is equivalent to the binary logic function $F_{m,n}$, where $r = \frac{2}{3}m$ and $s = \frac{2}{3}n$. Each MVFC consists of an array of NAND flash-like pulldown stacks (each stack implements a cube of the multi-valued logic function $H_{r,s}$). These pulldown stacks are grouped based on their output. We call these groups *multi-valued logic arrays* (MVLAs). Since the MVFC is a dynamic circuit, the outputs of the MVFC are pre-discharged to GND using the *oPch* circuit. The oPch circuit block consists of pulldown NMOS transistors driven by the clock signal (*clk*). The primary inputs of the MVFC drive all the MVLAs. Also, the clock signal (*clk*) also drives precharge and evaluate transistors in the MVLAs. In our implementation, the number of primary outputs of the MVFC is 2 (i.e. s = 2), and the number of primary inputs is 4 (i.e. r = 4). The MVFCs also include the programming logic that is required to program the flash transistors to implement the desired function $H_{r,s}$.

As mentioned earlier, each MVFC consists of a group of MVLAs, where $MVLA_k$ generates the k^{th} output minterm of the function $H_{r,s}$. For example, let the 3 ternary-valued

input cubes (represented using the multi-valued positional notation) $< 010|111|100|001 >$, $< 001|100|010|001 >$ and $< 100|010|010|100 >$ generate the output $< 010|100 >$, which is the 5^{th} output in decimal notation. In this case, $MVLA_5$ consists of 3 pulldown stacks, where each stack implements one of the input cubes mentioned above. The number of cubes implemented in any $MVLA_k$ is called *cubes per array* (CPA_k). The values of CPA_k are determined by the logic function $H_{r,s}$. In our MVFC-based design approach, only one of the MVLAs pulls down when an input minterm is applied to the MVFC, since input minterms are not shared among different MVLAs. Finally, the outputs 2^s through $3^n - 1$ are not used, and hence, we do not implement $MVLA_{(2^s)}$ through $MVLA_{(3^n-1)}$. In our MVFC-based design approach, we only implement 2^s MVLAs (shown as $MVLA_0$, $MVLA_1$, $\cdots$, $MVLA_7$ in Figure 8.2, for $s = 2$. Note that the MVLC implements $MVLA_0$ (the pre-discharge state) unlike the FC and the PFC (which do not implement the precharge state). This is because the output generated by $MVLA_0$ is $< 001|001 >$ (for a 2-output ternary-valued logic function). Since, in the pre-discharge state, we discharge all the outputs (which represent the illegal output state of $< 000|000 >$), we need to pull up the least significant digits of the outputs to represent the correct output state of $< 001|001 >$.

VIII.3.5 Multi-valued Logic Array

The delay of the MVLA increases as the number of cubes in the MVLA increases. This is a result of the increase in output diffusion capacitance incurred from using a wire-OR evaluation style of the MVLA. The delay of the MVLA is improved by splitting the pulldown stacks implemented in each MVLA into bundles of pulldown stacks, where each

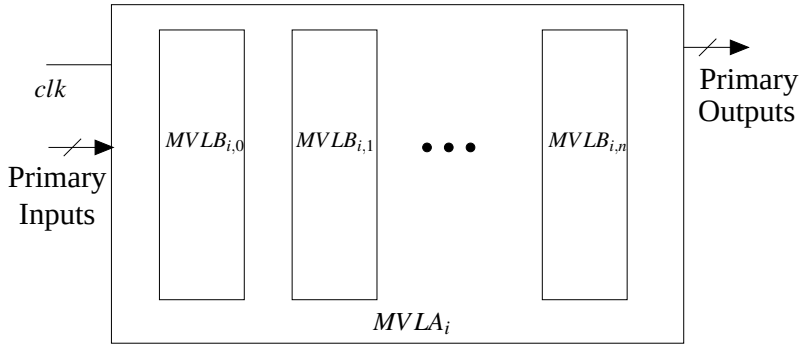


Figure 8.3: i^{th} Multi-valued Logic Array (MVLA$_i$) Structure.

bundle has a limited number of pulldown stacks (shown in Figure 8.3). These bundles are called *multi-valued logic bundles* (MVLBs). The circuit details of MVLBs are covered in Section VIII.3.6. The maximum number of pulldown stacks per MVLB is called *cubes per bundle* (*CPB*). In our implementation, it was found through parametric sweeps that *CPB* = 3 yields the best results. *CPB* is a global parameter that is fixed for all MVFC-based designs. If $MVLA_i$ has CPA_i cubes in total, then the total number of MVLBs in $MVLA_i$ is $\lceil \frac{CPA_i}{CPB} \rceil$. Note that all the MVLBs in each MVLA generate the same output, however, each one of them has its own output generation logic.

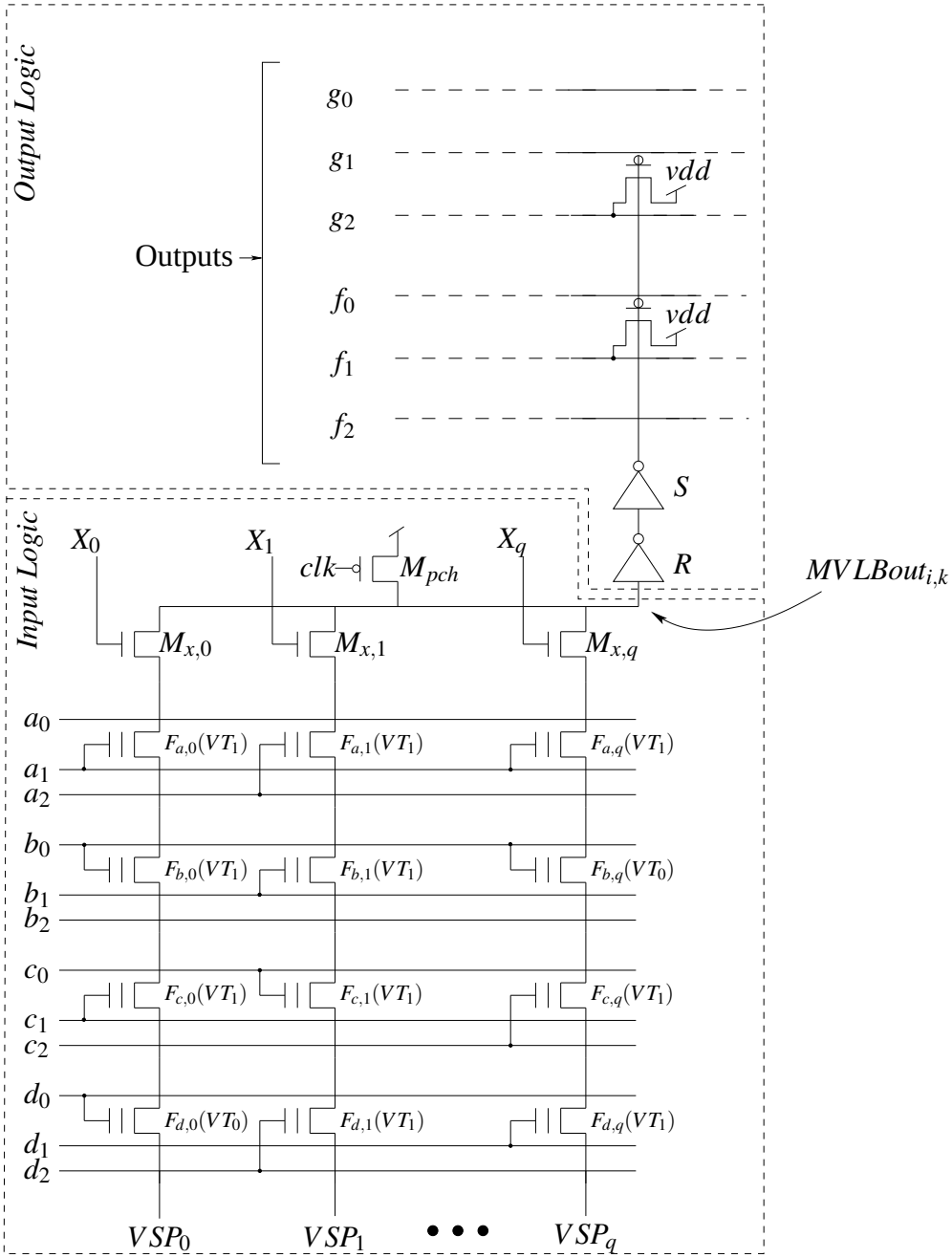


Figure 8.4: k^{th} Multi-valued Logic Bundle in MVLA$_i$ (MVLB$_{i,k}$)

VIII.3.6 Multi-valued Logic Bundles

In Figure 8.4 we show the circuit details of an MVLB. The MVLB can be split into two portions – a) the input logic, and b) the output logic.

The input logic (shown inside the bottom dashed area in Figure 8.4) consists of the NAND flash-like pulldown stacks. These stacks share the same output in a wire-OR configuration. Each one of the pulldown stacks implements a cube of the function $H_{r,s}$. The maximum number of cubes any MVLB can implement is *CPB*, which is an electrically determined parameter that is used to improve the characteristics (power, delay and area) of the MVFC-based designs. As shown in Figure 8.4, each pulldown stack in the MVLB has r flash transistors and 1 regular NMOS transistor (as shown in Figure 8.4). The multi-valued inputs of the function $H_{r,s}$ are implemented using one-hot literal representation. For example, since we are implementing ternary-valued logic, we represent the inputs a, b, c and d using their literals a_0, a_1 and a_2 for the input a, b_0, b_1 and b_2 for the input b, and similarly for the inputs c and d. For each input, only one of its literals will be high based on the input value. We will explain the configuration of the pull down stack using an example in Section VIII.3.7. The flash transistors are programmed to implement cubes, while the regular NMOS transistor is used for evaluate and program purposes. The shared regular PMOS transistor shown at the top of the *input logic* block of Figure 8.4 (M_{pch}) serves as the precharge transistor for all pulldown structures of the MVLB. When it turns on, it pulls up $MVLBout_{i,k}$ during the precharge (low) phase of the clock signal (clk). The lines VSP_0 to VSP_q are connected to ground during operation to allow the NAND stacks to pull down

when they evaluate, but also have a special purpose during programming, which will be discussed in Section VIII.3.8. The regular NMOS transistors ($M_{x,i}$) shown at the top of each stack in Figure 8.4 serve two purposes. In regular operation they are used as evaluate transistors, and are off during the precharge (low) phase of clk and only turn on during the evaluate (high) phase of clk. In this way, they allow the pulldown stack to evaluate the output $MVLBout_{i,k}$. The NMOS transistors ($M_{x,i}$) are also used during the programming of the NAND flash stack. Programming will be discussed in Section VIII.3.8.

The output logic (shown inside the top dashed area in Figure 8.4) generates the final output of the MVFC. Each MVLB has a dedicated output logic that activates when one of the pulldown stacks of the MVLB pulls down (when the applied input minterm is contained in one of the cubes implemented by this MVLB). When the output logic is activated, it pulls up the corresponding pre-discharged output lines, generating the final output. The horizontal lines shown in the top part of Figure 8.4 represent one of the final one-hot output lines in the MVFC. These output lines run across the entire MVFC, as shown in Figure 8.2. As mentioned in Section VIII.3.4, the output lines are pre-discharged using NMOS transistors in the oPch circuit shown in Figure 8.2. The output logic circuit has PMOS transistors that charge the output lines based on which output this MVLB drives. For example, if the current MVLB is part of $MVLA_5$, then the ternary-valued output generated by this MVLB is $< f, g > = < 010|100 >$. As a result, the output logic will connect the drains of two PMOS transistors to the lines f_1 and g_2, respectively (given that f is the most significant output digit). This is illustrated in Figure 8.4. The gates of these

two PMOS transistors are driven by the buffered output of the NAND flash-like pulldown stacks, as shown in Figure 8.4. These PMOS transistors are sized to drive a load equal to 3 MVFC inputs.

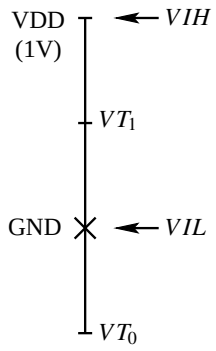


Figure 8.5: Threshold Voltages Used in Multi-valued Flash-based Designs

VIII.3.7 MVLB Configuration

Unlike the TLC-based approach (presented in Chapter III and in [64]) which uses 3 V_T levels, the MVLC-based design approach uses two V_T levels only. Figure 8.5 shows (on the left side), the input voltages applied to the gate of the flash devices in the MVLC. On the right side, the figure shows the V_T levels used to configure the flash transistors in the MVLC. The MVLC is configured by first determining which input cubes of the function $H_{r,s}$ are implemented by each MVLA. $MVLA_k$ implements all cubes that produce the output vector whose decimal encoding is k. Then each MVLA is split into a group of MVLBs based on the number of cubes implemented by this MVLA. Each input cube of the function $H_{r,s}$ is configured in the MVLB (shown in Figure 8.4) by choosing the V_T

of the flash transistors in the stack corresponding to the input cube. Table 8.1 shows the configuration of each flash transistor for each literal. The flash transistors in the MVLC can only be configured to implement the literals $< 001 >$, $< 010 >$, $< 100 >$ and $< 111 >$. The remaining literals shown in Table 8.1 cannot be represented by the MVFC design approach. For example, the left-most stack of Figure 8.4 implements the ternary-valued cube $< abcd > = < 010|001|010|111 >$. Note that the transistor $F_{d,0}$ is driven by the literal d_0 and is programmed to a threshold voltage VT_0 which is below GND (see Figure 8.5). The threshold voltage VT_0 is also referred to as the *erase* threshold voltage. Therefore, this transistor is *on* irrespective of the value of the literal d_0. In other words, this transistor is *on* irrespective of the value of the input d, and hence implements the multi-valued literal of $< 111 >$ for d. In fact, the flash transistors that are programmed to VT_0 are always driven by the 0^{th} literal of their input, only because they cannot be left floating. Now, the transistors $F_{a,0}$, $F_{b,0}$ and $F_{c,0}$ are programmed to a threshold voltage VT_1, which is between VDD and GND (see Figure 8.5). This means that these devices turn on only when their gate signal is high. The transistors $F_{a,0}$, $F_{b,0}$ and $F_{c,0}$ are driven by the signals a_1, b_0 and c_1 respectively. As a consequence, the left-most stack of Figure 8.4 implements the cube $< abcd > = < 010|001|010|111 >$.

The second stack of Figure 8.4 implements the cube $< 100|010|001|100 >$. Note that all of its transistors are programmed to the VT_1 threshold, which is why it implements a minterm in the r-input space of $H_{r,s}$. Finally, the rightmost stack of Figure 8.4 implements the cube $< 010|111|100|010 >$.

Literal	V_T of F_x	Gate Input
001	VT_1	x_0
010	VT_1	x_1
100	VT_1	x_2
011	Not supported	N/A
110	Not supported	N/A
101	Not supported	N/A
111	VT_1	x_0
000	Invalid	N/A

Table 8.1: Flash Transistor Configuration F_x for Each Minterm/Cube Literal.

VIII.3.8 Programming the Multi-valued Flash Cluster

Figure 8.4 shows the MVLB circuit structure, including both the input logic and the output logic parts. The input logic is the only part that has flash transistors and requires programming. The output logic part does not contain flash transistors, and hence, does not require programming.

In an MVFC, all the flash transistors share a common bulk. As a result, the flash transistors in the MVFC are all erased together. This is performed by driving the bulk node of the MVFC to a high voltage, and floating the source and drain terminals of each flash transistor. The source and drain terminals of the flash transistors in each stack are floated by turning off all the $M_{x,i}$ transistors and floating all the VSP_i signals. The gates of all transistors are driven to GND. This results in the erasure of all the flash transistors in the MVFC, and resets their threshold voltage to VT_0.

For programming, assume that $F_{b,0}$ and $F_{b,1}$ need to be programmed. In this case,

the b_0 and b_1 lines are driven to a programming voltage for a sufficiently long duration. The transistors $M_{x,0}$ and $M_{x,1}$ are turned on by driving X_0 and X_1 high. All other X_i are driven low. Also, the lines VSP_0 and VSP_1 are driven low and the remaining VSP_i lines are floated. This disables programming of all but the 1^{st} and 2^{nd} NAND stacks of the MVLB. All other inputs (i.e. a_0, a_1, a_2, b_2, c_0, c_1, c_2, d_0, d_1 and d_2) are driven high to a pass voltage, and the common bulk is held to GND. The duration of the programming pulse is determined based on the final desired V_T. This results in a programming of $F_{b,0}$ and $F_{b,1}$ to the desired V_T (VT_1), while the threshold voltage levels of all other transistors in the MVLB are unaltered from the erase threshold voltage (VT_0).

VIII.3.9 Logic Minimization of the MVFC

We use Espresso-MV [29] to minimize the logic function ($H_{r,s}$) before implementing it in the MVFC (recall $H_{r,s}$ has r inputs and s outputs). The input minterms $\{m_j\}$ of each output minterm j are minimized separately using Espresso-MV. This guarantees that the outputs produced from Espresso-MV are one hot and are compatible with our MVFC output circuit (which only generates one hot outputs). We also ensure that Espresso-MV does not generate any unsupported input literal as shown in Table 8.1. If unsupported literals are produced, they are split. For example a literal $< 101 >$ will be split into 2 cubes, one with literal $< 100 >$ and the other with literal $< 001 >$. The resulting minimized cover is used to configure the MVFC.

VIII.4 Experiments

In this chapter, we compare our MVLC-based design approach to the CMOS standard cell-based and the TLC-based [64] design approaches. In this section, we first present the simulation environment used in evaluating our proposed design approach. Then, we present the details of our experiments and discuss the results.

VIII.4.1 Simulation Environment

In our experiments, we evaluate the flash-based designs using flash design flows (MVFC and TLC), and CMOS-based designs using a CMOS standard cell design flow. In all three design flows, we implement the same benchmark circuits using the flash-based and the CMOS-based design approaches and compare the results obtained by each design approach. We implement all design approaches in 45nm process technology.

For the CMOS-based design approach, we use Synopsys Design Compiler [56] to synthesize and map the benchmark circuits. We use the 45nm Nangate FreePDK45 Open Cell Library [54, 55] to implement the CMOS-based designs. The mapped designs are then simulated using HSPICE circuit simulator from Synopsys [56] using a 45nm PTM model card [57]. Layout areas were computed as the sum of cell areas.

For the flash-based design approach, we used custom scripts to generate the circuits. We also used HSPICE to simulate the flash-based circuits. The CMOS devices in the flash-based designs were modeled using a 45nm PTM model card, and the flash devices used a model card that we generated using model card regression from device measurements obtained for a fabricated 45nm flash device as described in Chapter III and [64]. For

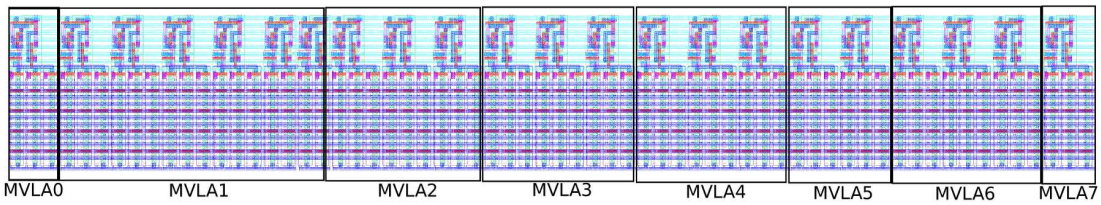


Figure 8.6: Layout View of an MVFC Implementing Benchmark ”des00”

flash-based design approaches, we verified the correct logical operation through exhaustive simulation. We generated custom layouts for the multi-valued digital circuit using Cadence Virtuoso. The layout of our MVFCs used design rules for flash devices that were compiled from the ITRS [59]. The layout area for the flash-based designs are more accurate than those for the CMOS standard cell-based designs (which are lower bounds of the true area).

We generated 20 random designs to evaluate our multi-valued digital circuit design approach. These designs implement binary logic functions of 6 inputs and 3 outputs, and were converted into multi-valued designs (ternary-valued in this chapter) as described previously in Section VIII.3.2. The logic functions implemented in the MVFCs in this chapter have 4 inputs (p = 4) and 2 outputs (q = 2). The maximum number of cubes per bundle (CPB) we used in this chapter is 3. The threshold voltages used for the flash transistors in our MVLC design approach were VT_0 = -0.5V and VT_1 = 0.5V. The flash-based designs were each verified to be correct by conducting exhaustive circuit simulations.

Design	D_{max} Ratio	P_{avg} Ratio	*Eng* Ratio	Cell Area Ratio
des00	0.73×	0.92×	0.68×	0.99×
des01	0.71×	0.87×	0.62×	0.86×
des02	0.77×	0.95×	0.74×	0.93×
des03	0.69×	1.01×	0.70×	0.89×
des04	0.79×	1.03×	0.82×	0.97×
des05	0.71×	0.89×	0.64×	0.94×
des06	0.88×	0.89×	0.79×	0.96×
des07	0.77×	0.98×	0.76×	0.97×
des08	0.78×	0.93×	0.73×	0.96×
des09	0.80×	0.84×	0.68×	0.93×
des10	0.83×	1.05×	0.88×	0.92×
des11	0.77×	1.04×	0.80×	0.90×
des12	0.79×	0.97×	0.77×	0.90×
des13	0.83×	1.04×	0.87×	1.02×
des14	0.73×	0.90×	0.66×	0.93×
des15	0.91×	1.05×	0.96×	1.02×
des16	0.80×	0.90×	0.72×	1.02×
des17	0.69×	0.95×	0.66×	1.02×
des18	0.75×	0.91×	0.69×	1.02×
des19	0.70×	0.97×	0.68×	1.01×
Average	0.77×	0.95×	0.74×	0.96×
Stdev	0.06×	0.07×	0.09×	0.05×

Table 8.2: Delay, Power, Energy and Area Ratios of Multi-valued (Ternary) Logic Circuits Relative to CMOS Standard Cell-based Circuits

VIII.4.2 Results and Analysis

Table 8.2 shows the delay (including precharge delay), power, energy and physical area ratios of 20 randomly generated logic functions implemented using our MVFC-based approach compared to the CMOS-based based implementation. The delay reported in the table (D_{max} Ratio) is maximum delay of any transition seen at any primary output of the circuit. Since the flash-based implementation is dynamic, we accounted for the precharge

delay in the reported delay shown in the table. As shown in the table, the delay of the flash-based digital circuits ranges from 0.69× to 0.91× of the CMOS standard cell-based digital circuit delay, with an average of 0.77×. Table 8.2 also shows power dissipation (of 0.95×) when implementing the digital circuits using our multi-valued logic compared to CMOS standard cell based implementation. We also show the energy utilization of our multi-valued implementation compared to the CMOS standard cell based implementation. On average, the energy utilization of the MVFC-based ternary-valued digital circuits is about 0.74× of the CMOS standard cell based implementation. Also, on average, digital circuits implemented using an MVFC-based design approach use 0.96× the physical area of a CMOS-based design. Figure 8.6, shows the representative layout of an MVFC implementing the benchmark "des00". Note that the MVFC implementing the benchmark "des00" implements $MVLA_0$ (the pre-discharge state). As mentioned in Section VIII.3.4, the pre-discharge state must be implemented by $MVLA_0$ in order to generate a legal output state during the evaluate phase of the clock. In general, it is rare for a new circuit design approach to beat the established standard cell-based approach on all metrics (delay, area, power and energy), and so these results are significant.

The MVFC shows tremendous improvements in the delay compared the TLC (presented in Chapter III and [64]). The delay of the MVFC is ∼0.32× the delay of the TLC. This delay improvements in the MVFC comes with an increase in its power dissipation and energy. In average, the MVFC has ∼7.9× the power dissipation and ∼2.5× the energy consumption of a TLC. The MVFC has slightly larger physical area (1% larger) compared

to the TLC. The TLC can be used for power or energy constrained designs which can tolerate much larger delays. The MVFC can be used for delay sensitive designs, and improves on standard cell designs in terms of delay, area, power and energy.

Table 8.2 also shows the standard deviation of the results. The standard deviation indicates that the relative results of the flash-based designs vary by 6% in delay, 7% in power, 9% in energy and 5% in physical area. The noise in these metrics is smaller than the improvements, specially for delay and energy.

Flash-based digital circuits have the ability of tuning their delay, power and energy characteristics. This is done through shifting the V_T of the flash transistors in the circuit. The ability to shift V_T with precision offers the flash-based digital circuits huge advantages over the traditional CMOS standard cell-based circuits when it comes to speed binning at the factory, aging mitigation and performing post-manufacturing ECOs. Figure 8.7 shows the average delay, power and energy of the flash-based digital designs as their V_T is swept. The delay in Figure 8.7 is the sum of the evaluate and the precharge delays.

We note that by reducing the V_T from the nominal values (indicated by a "V_T shift" of 0 mV in Figure 8.7), the current delay can be reduced, at the cost of power. The converse is true as well. This can be used to do aging mitigation in the field, or speed/power binning in the factory.

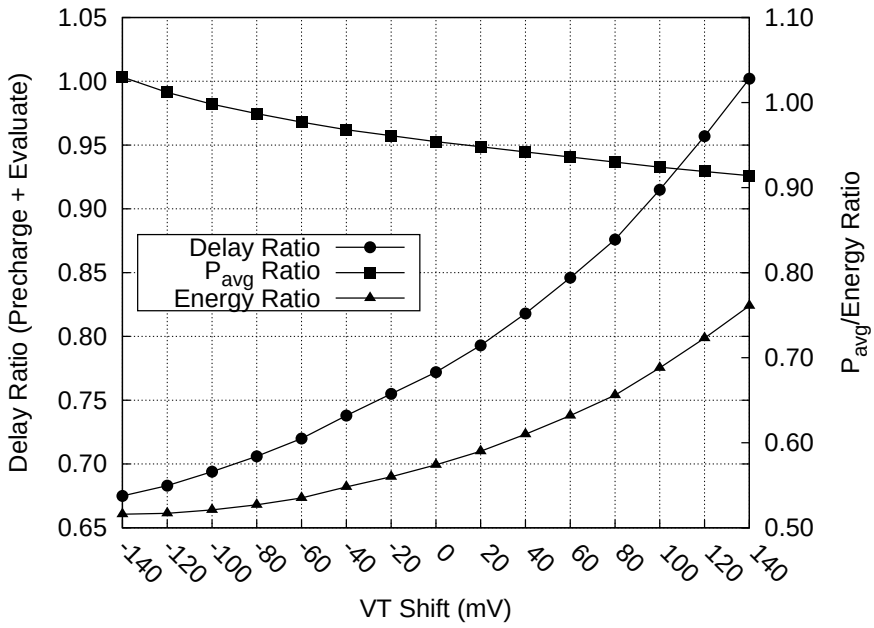


Figure 8.7: Delay, Power and Energy of the MVFC-based Designs as the V_T is Shifted.

VIII.5 Chapter Summary

This chapter presented the use of flash transistors to implement multi-valued digital circuits. We presented the details of the circuit topology that we use in our multi-valued, flash-based digital circuit approach. Our HSPICE simulations show that, averaged over 20 designs, our approach yields improved delay ($\sim$23% lower), power ($\sim$5% lower), energy ($\sim$26% lower) and area ($\sim$4% lower) characteristics compared to a traditional CMOS standard cell based approach, when averaged over 20 designs. Also we reported that the MVFC has $\sim$0.32$\times$ the delay of the TLC. The delay improvement is accompanied by an increase in power ($\sim$7.9$\times$) and energy ($\sim$2.5$\times$). Our approach targets high performance multi-valued, flash-based applications, while the TLC may be used for power/energy crit-

ical applications that can tolerate higher delays.

CHAPTER IX

FLASH-BASED FIELD PROGRAMMABLE GATE ARRAY (FPGA)[1]

In this chapter, we present a study of flash-based FPGA designs (both static and dynamic), and present the tradeoff of delay, power dissipation and energy consumption for the various designs. Our work differs from previously proposed flash-based FPGAs, since we embed the flash transistors (which store the configuration bits) directly within the logic and interconnect fabrics. We also present a detailed description of how the programming of the configuration bits is accomplished. Our proposed static flash-based LUT structure yields a faster operation, lower dynamic power dissipation, lower energy consumption and lower static power dissipation compared to a traditional SRAM-based LUT. We also show that, for high performance applications, a dynamic flash-based LUT can achieve further performance improvements with higher energy consumption compared to an SRAM-based LUT. We also show that the flash-based interconnect structure, implemented using our approach, provides lower delay and lower overall power consumption compared to the traditional interconnect structure used in SRAM-based FPGAs.

IX.1 Background

FPGAs consist of a number of *configurable logic block* (CLBs). Each CLB implements a small portion of the entire logic programmed on the FPGA. Each CLB is com-

[1]Part of the data reported in this chapter is reprinted with permission from "Exploring Static and Dynamic Flash-based FPGA Design Topologies" by Monther Abusultan and Sunil P. Khatri, IEEE 34th International Conference on Computer Design (ICCD) 2016, pp. 416-419, Copyright 2016 by IEEE.

prised of one or more *look-up tables* (LUTs). Each LUT can implement any logic function of up to *n* inputs (where *n* varies from 4 to 6 depending on the FPGA device). The design of the LUT is a key determinant of the speed, area and power of the FPGA. Therefore, efficient design of the LUT is a critically important issue. In addition, the FPGA uses a network of programmable switches to connect the signals between the LUTs, as well as to connect the LUT I/O signals to the FPGA I/O signals. This dense interconnect fabric occupies a large portion of the FPGA's real state due to the large number of programmable switches.

Most FPGAs use 5 transistors (5T) *static random access memory* (SRAM) cells to hold the configuration bits to implement the LUT logic function as well as the interconnect configuration. SRAM cells are volatile, however, and so there is a significant interest in the use of non-volatile memory devices to store the FPGA logic and interconnect configuration bits. One of the most popular non-volatile memory technologies used in industry is flash memory.

Figure 9.1, Figure 9.2 and Figure 9.3 show three styles of non-volatile FPGA designs that can retain the configuration bits after a power cycle event. Figure 9.1 shows a CMOS SRAM-based FPGA that has an off-chip *programmable read-only memory* (PROM) chip to store the configuration bits and load them into the SRAM cells after the chip is powered on (similar to conventional FPGAs manufactured by Xilinx [77]). This results in prolonged boot up delays. Figure 9.2 uses on-chip flash memory to store the configuration bits [78, 79]. Once the chip is powered on, the configuration bits are transferred from the

flash memory to the CMOS SRAM cells that hold the FPGA configuration during operation. Although the design in Figure 9.2 has both CMOS and flash transistors on the same chip, it only uses the flash memory to hold the configuration bits while the chip is powered off. This type of FPGAs also has a prolonged boot up delay similar to that in Figure 9.1. Figure 9.3 shows an FPGA that uses flash memory to hold the configuration bits. The CMOS SRAM cells are replaced with flash memory cells which store the FPGA configuration bits, both during operation as well as while the FPGA is powered down. This is the approach used in [80, 81] and the work presented in this chapter. Our work differs from [80, 81] since we *embed* the flash transistors in the interconnect and logic fabrics directly.

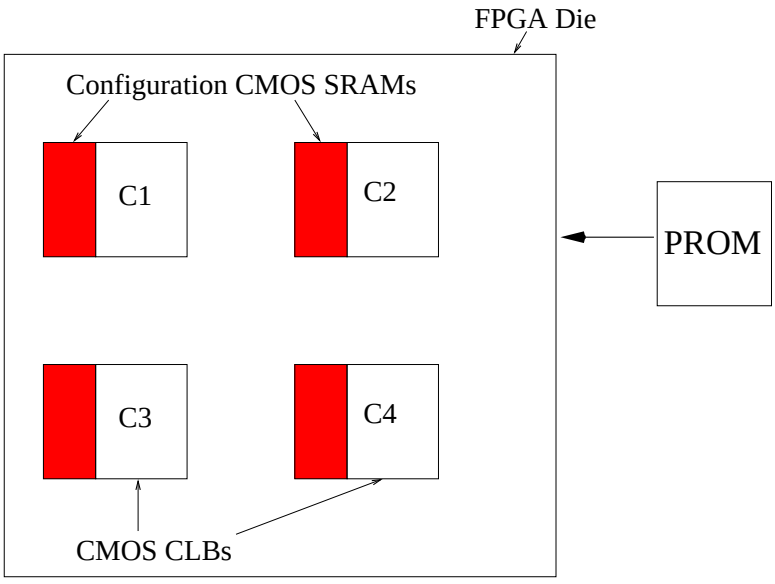


Figure 9.1: SRAM-based FPGA with Off-chip PROM

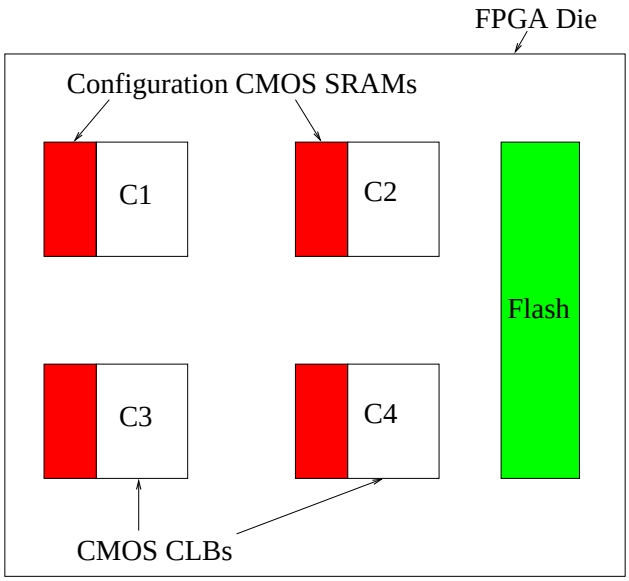


Figure 9.2: SRAM-based FPGA with On-chip PROM

The use of flash transistors to hold the configuration bits in our approach results in a smaller FPGA die area due to the absence of 5T-SRAM cells. The use of both CMOS SRAM cells *and* flash memory to hold the configuration bits (as shown in Figure 9.2) results in a larger FPGA die to hold the same amount of programmable fabric compared to the designs shown in Figures 9.1 and 9.3. The FPGA in Figure 9.1 uses more die area than the FPGA in Figure 9.3, and in addition, it requires additional off-chip PROM memory to hold the configuration bits during the power off condition. Additionally, in our approach, the use of flash transistors as the FPGA's main configuration memory allows for temporarily power gating of parts of the FPGA (which are not being used by the application) and also enables the application to instantly power up and power down any portions of the FPGA.

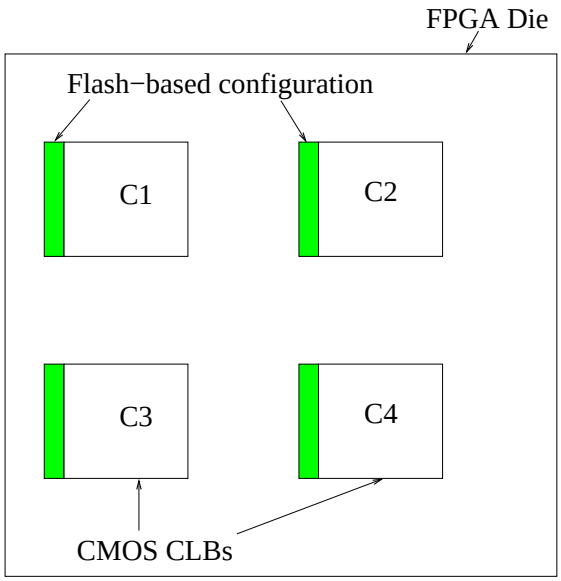


Figure 9.3: Flash-based FPGA

Loading the configuration bits into flash-based FPGAs is similar to that of SRAM-based FPGAs. In SRAM-based FPGAs, configuration bits are grouped in k-bit words which are loaded serially into a configuration register using a JTAG boundary scan interface [82]. A representative value of k is 32. In SRAM-based FPGAs, configuration cells (logic and interconnect) are grouped in frames laid vertically in the FPGA. Each frame consists of a number of words (101 words in the 7 Series Xilinx FPGAs). Each frame is programmed by loading the configuration bits in parallel from the configuration register into each word of the frame. In the flash-based FPGA, configuration data is loaded serially into a configuration register similar to SRAM-based FPGAs. Programming the flash transistors in parallel can be achieved for both the LUTs and the interconnect configuration bits. For LUT configuration, we program one flash transistor from each LUT at a time

(the details are described in Section IX.3). We use a 32-bit configuration register to program 32 flash transistors in parallel for 32 different LUTs. For interconnect configuration, we program the flash transistors in each row in parallel. Flash-based FPGA (LUT and interconnect) programming is discussed in details in Section IX.3.

In this chapter, we explore the use of flash transistors to implement non-volatile FPGAs. We remove all the CMOS SRAM cells in a traditional FPGA structure and redesign the FPGA to only use flash transistors to store the configuration bits as shown in Figure 9.3. We present and compare different LUT and interconnect structures that can be implemented in a non-volatile FPGA. We also present the tradeoff between the delay, dynamic power dissipation, energy consumption and static power dissipation characteristics (obtained from the circuit level simulations) for each of the proposed structures. We also present a flash-based interconnect fabric, and compare its electrical characteristics with traditional CMOS SRAM-based FPGA interconnect.

In general, the differences between an SRAM-based FPGA and our proposed flash-based FPGA are:

- Area– since the flash-based FPGA only uses 1 flash transistor per configuration bit versus 5 transistors in the SRAM-based FPGA (5T SRAM cells) per configuration bit.
- Power– flash transistors in general have low on-currents compared to regular NMOS transistors. Also, flash transistors have lower input capacitance (discussed in Section IX.4).

- Delay– embedding the flash transistors in the logic and interconnect fabrics shortens the signal path from the configuration cells to the output of the LUT (or the input of the interconnect switch), resulting in improved speeds.
- Programming time– programming flash transistors is a much slower process than programming CMOS SRAM cells. This could become an issue during the application development phase, during which the FPGA will be loaded frequently with configuration bits. However, once the FPGA is deployed, flash-based FPGAs are at an advantage, since they do not require reloading of the configuration bits whenever the FPGA is booted up after a power cycle (compared to the $\sim$1.8 seconds delay in SRAM-based FPGAs in the example discussed earlier). This makes flash-based FPGAs the preferred option for the production and deployment phase of an application. Also, using flash-based FPGAs in production keeps the FPGA safe from reaching the limit on the number of program cycles of the flash-based configuration bits (which is in the order of 10K-100K [40, 41]).

IX.2 Previous Work

Work in the area of flash-based FPGA implementation has been reported in [81, 80]. The authors of [81] present a flash-based FPGA implementation that uses a 2T composite flash device similar to [83], which is a flash device that consists of two floating gate transistors that share their floating and control gates. One transistor is used for programming and the other is used for operation. The work in [81] eliminates the need for conven-

tional SRAM cells and only use flash devices to store the configuration bits, resulting in a low-cost, low-power FPGA design. Unlike the work presented in [81], our work uses 1T flash devices which leads to further area reduction. Also we propose multiple flash-based FPGA designs and compare them amongst each other as well as with SRAM-based FPGA designs (unlike [81] which only presents their flash-based FPGA design with no performance and power comparisons with current CMOS SRAM-based FPGAs). Unlike [81], we show the circuit level details of the LUT and interconnect implementation used in our flash-based FPGA.

The authors of [80] show the implementation details of a non-volatile programmable switch for use in flash-based FPGAs. The work in [80] is implemented using a 2T flash transistor similar to [81, 83] and they show area improvement when using flash-based versus SRAM-based configuration memory. They also demonstrate a static power reduction due to the fact that they employ a power gating scheme which is mostly effective when the FPGA has low utilization. In contrast to [80] we use 1T flash devices and present a complete implementation of the flash-based FPGA, including the logic and interconnect fabrics. We also use an NMOS-only MUX implementation (instead of the CMOS MUX implementation used in [80]) because the NMOS-only MUX implementation has been proven to be faster, with a lower power consumption in the literature due to its reduced diffusion area [4, 34, 84, 6]. Furthermore, in contrast to both [81, 80], our proposed flash-based FPGA designs use flash transistors that are embedded in the logic and interconnect circuitry (instead of placing the configuration bits in a separate flash-based memory cell

as in [81, 80]) which leads to improvements in delay, power and area. Unlike [81, 80], we present a detailed description of how the programming of the flash configuration bits is accomplished.

IX.3 Approach

IX.3.1 Overview

In this section, we discuss the design details of our candidate flash-based FPGA structures. We will first discuss the general FPGA topology used in the different designs. Then we will present our proposed flash-based lookup table (LUT) structures. Finally, we will discuss the details of the interconnect switch boxes used in our proposed flash-based FPGA. For each candidate design that we present in this chapter, we will describe the circuit level details of the design as well as a detailed walk-through of how it is programmed.

Figure 9.4 shows the basic representation of a logic island in an FPGA with the surrounding interconnect elements used to transfer signals to/from the LUTs. Each group of LUTs in our FPGA architecture is grouped into a larger logic unit called a *configurable logic block* (CLB). CLBs contain flip-flops at the output of each LUT as well as the required circuitry to select the registered or the non-registered output of the LUTs. Figure 9.4 also shows a *connection box* (CB) which is a matrix of switches that connects each I/O in the CLB with the *switch box* (SB), also shown in Figure 9.4. The SB is used to connect signals from the CB to the FPGA interconnect fabric. In this chapter, the SB will be used to evaluate the proposed flash-based interconnect structures. The SB structure used in

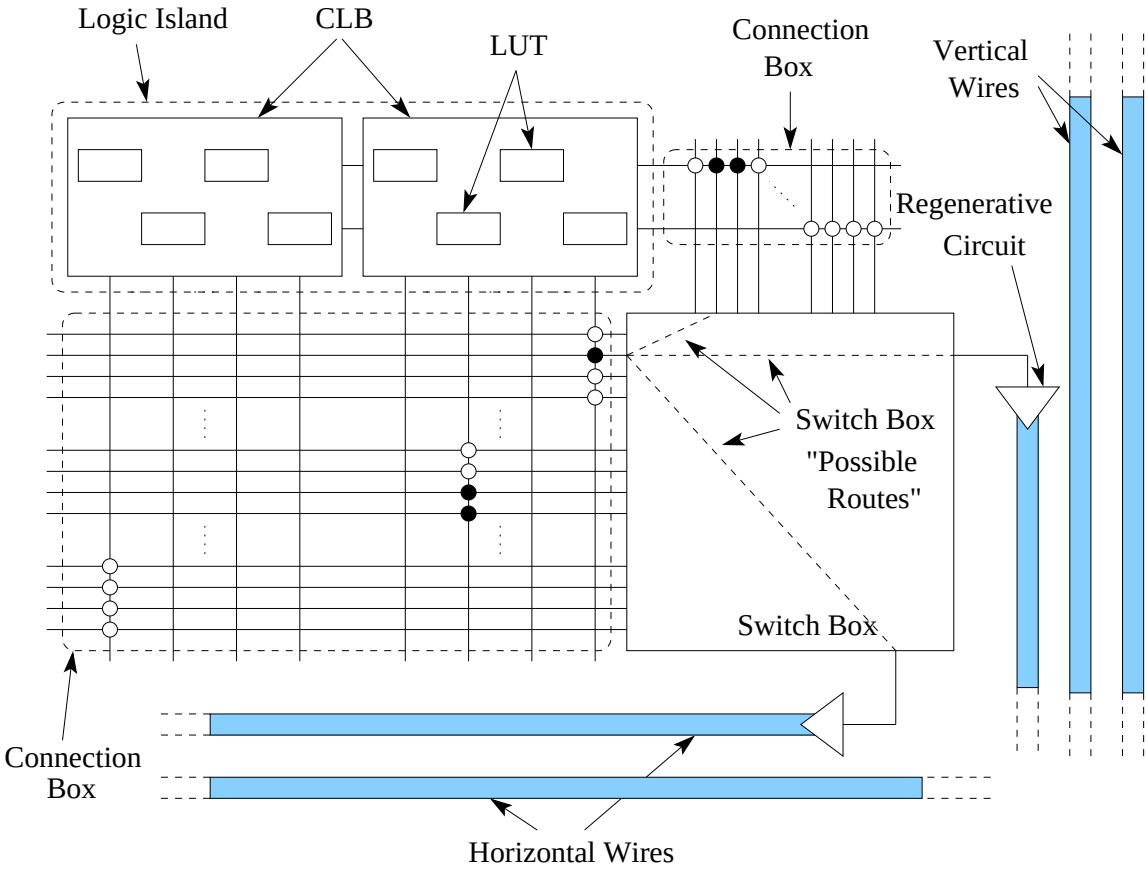


Figure 9.4: FPGA Logic and Interconnect Fabric Elements

this chapter can implement any of the commonly used interconnect topologies described in [85, 86, 87], and can be used to implement the CB as well.

Next, we will briefly discuss the conventional LUT structures which we have used to evaluate our proposed LUT structures against.

IX.3.2 Conventional SRAM-based LUT Structure

Figure 9.5 shows the circuit details of the conventional LUT used in SRAM-based FPGAs (SReg-LUT). In this chapter, we evaluate 6-input LUTs since they are the most

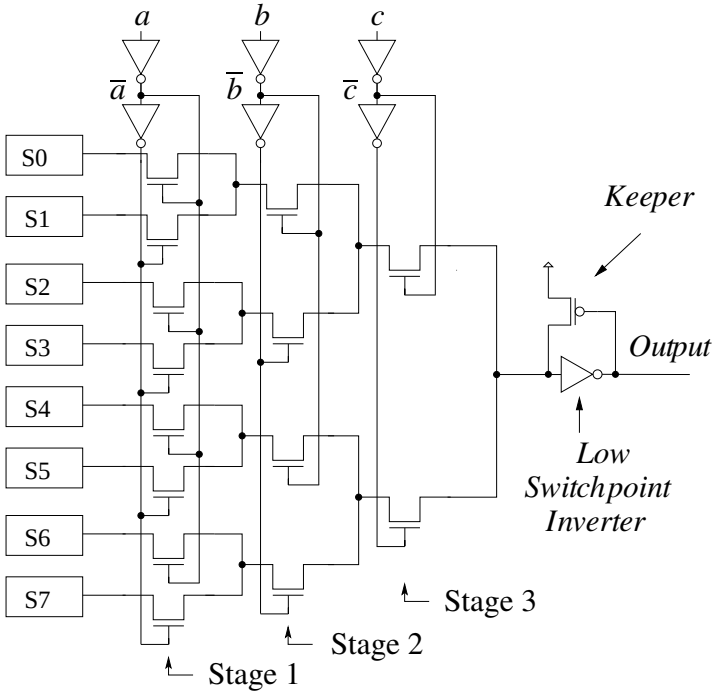


Figure 9.5: Conventional 3-input LUT Structure

commonly used in current FPGAs, however we only show a 3-input LUT in Figure 9.5, for simplicity. The blocks marked S0-S7 represent the logic configuration memory. They are realized using 5T SRAM cells in the SRAM-based LUT. When the input (a, b and c) is applied to the LUT circuit, one of the values of the SRAM cells S0-S7 propagates through the NMOS MUX tree and drives the output. We use a low switch-point inverter and a long channel PMOS keeper to regenerate the signal at the output, since it is degraded after passing through the NMOS MUX stages. Figure 9.6 shows the circuit details of four 5T SRAM cells used in an example 2-input LUT. The top SRAM cell in Figure 9.6 consists of two cross-coupled inverters, P_0 (minimum size inverter) and Q_0 (long channel inverter), and an NMOS pass gate (M_0) which is used to load the new configuration bit into the

SRAM cell. This SReg-LUT structure is used as a reference in the evaluation of our flash-based candidate LUT structures.

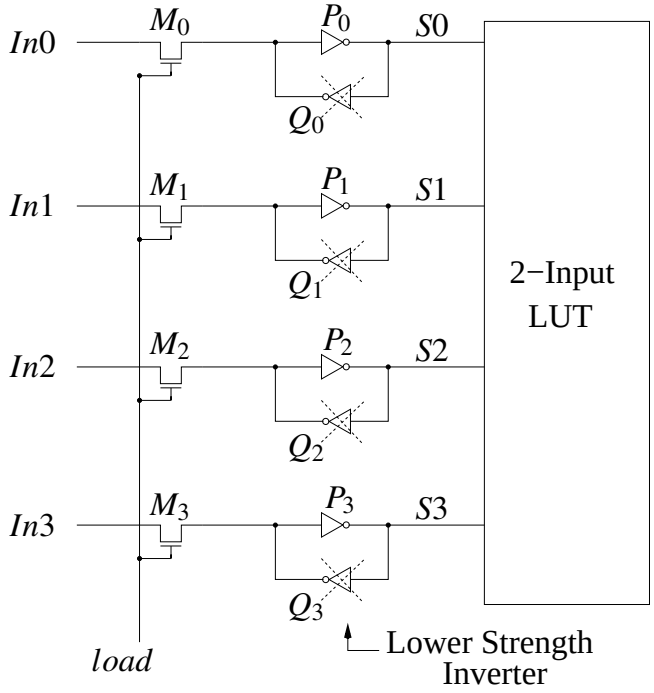


Figure 9.6: 5T SRAM Cells

IX.3.3 Conventional Flash-based LUT Structure

IX.3.3.1 LUT Structure

The conventional flash-based LUT structure (FReg-LUT) is very similar to the SReg-LUT structure shown in Figure 9.5. The main difference is in the implementation of the memory cells S0-S7. Figure 9.7 shows the flash memory cells (FMCs) used to hold the configuration bits of an example 2-input LUT. Each cell consists of two flash transistors Fp*i* and Fq*i*. Fp*i* and Fq*i* are driven by VAP and VBP respectively. During

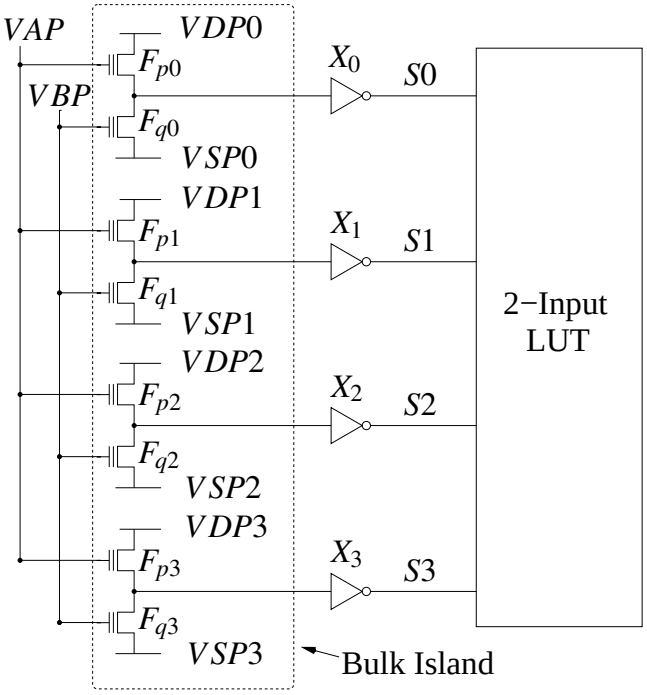


Figure 9.7: Flash Memory Cells (FMCs)

operation, the lines VAP and VBP are driven high (VDD) to turn on the FMCs, and the lines VDP*i* and VSP*i* are driven to VDD and GND respectively. The transistors Fp*i* and Fq*i* are programmed such that only one of them turns on when the corresponding VAP*i* and VBP*i* are driven high, thereby passing either the VDP*i* (VDD) or the VSP*i* (GND) voltage to the inverter X*i* which generates the configuration signal S*i*. The inverter X*i* is required to regenerate the degraded signals due to the use of only N-type flash transistors. Note that the configuration bits have to be logically flipped to account for the inversion caused by the inverter X*i*. In this chapter, the FReg-LUT is used as a reference design in the comparative analysis with our flash-based LUTs.

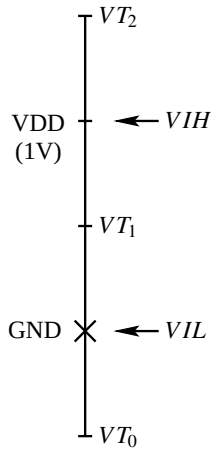


Figure 9.8: Threshold Voltages Levels Used in Flash-based Designs

IX.3.3.2 Programming the FReg-LUT

Programming flash memory in general involves two steps – a) erasing an entire block of cells, and b) programming individual transistors to a desired threshold voltage value as shown in Figure 9.8. Figure 9.8 shows the voltage levels used to drive the flash transistors (VDD and GND) in our implementation of flash-based FPGAs. The V_T levels used to program the flash transistors are shown to the right side of Figure 9.8. In all of the flash-based implementations, we use 3 V_T levels. The low (erase) threshold voltage (VT_0) is below GND, the medium threshold voltage (VT_1) is between VDD and GND, and the high threshold voltage VT_2 is above VDD. All the flash transistors in the flash memory cells shown in Figure 9.7 are erased by driving the lines VAP and VBP to GND, floating all the VDP*i* and VSP*i* lines, and driving the shared bulk island with a high voltage (10V-20V). The erase process will result in reseting all the threshold voltages of the flash transistors

shown in Figure 9.7 to the erase threshold VT_0 in Figure 9.8. In other words, all the flash transistors will be in an "always passing" state. After erasing the flash transistors, we only need to program one flash transistor in each FMC based on the value of the corresponding configuration bit. If we want to store a 1 in the FMC, we have to program Fqi to VT_2 and leave Fpi in the erase state (VT_0). If we want to store a 0 in the FMC, we have to program Fpi to VT_2, and leave Fqi in the erase state. Notice that due to the inversion at the output of the FMC, the final output of the FMC (Si) will be inverted, which has to be taken into consideration when generating the logic configuration bit string of the LUT. Fqi is programmed by driving VBP to a high voltage (10V-20V), VAP to a passing voltage (2V), VDPi and VSPi to GND and holding the bulk at GND. The programming duration determines the final V_T value of Fqi (VT_2 in our case). Note that the lines VAP and VBP are shared between LUTs in the same column of the FPGA, and the lines VDP and VSP are shared between LUTs in the same row of the FPGA (as shown in Figure 9.9). This allows us to address all FMCs in the FPGA. We float the VDP and VSP signals of the flash transistors in the same column which do not need to be programmed. We drive the VAP and VBP signals of flash transistors in the same row to GND, if these transistors do not need to be programmed. In this way, at most $2k$ flash transistors can be programmed at a time, where k is the number of LUTs per column.

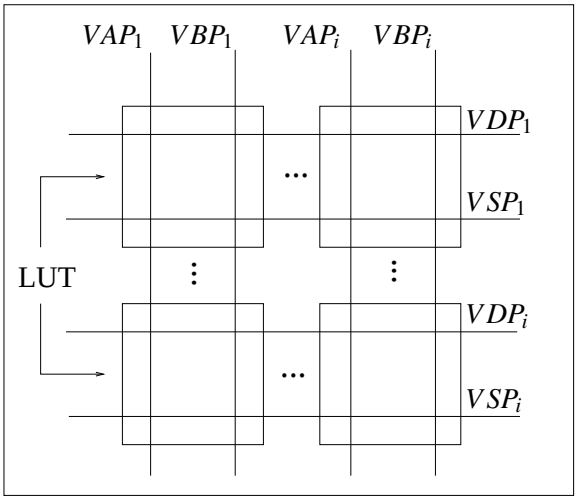


Figure 9.9: Programming the Configuration Flash Memory Cells

IX.3.4 Proposed Static Flash-based LUT (SF-LUT)

IX.3.4.1 LUT Structure

Figure 9.11 shows our proposed static flash-based LUT structure. The LUT consists of a modified MUX tree, a low switch point inverter (P) and a long channel keeper PMOS similar to Figure 9.5. The main difference between the proposed LUT structure and the conventional LUT structure of Figure 9.5 is the configuration scheme. The memory cells used to hold the configuration bits of our proposed LUT are merged in the LUT's MUX tree, which results in reduced area, delay and power. We will explain how the SF-LUT is configured in Section IX.3.4.2. During operation, the lines VSP and VDP are driven to GND and VDD respectively, the lines $A/VAP1$ and $\overline{A}/VAP2$ are driven to A and $\overline{A}$ respectively, and the lines B/VBP_0 and $\overline{B}/VBP_{1:4}$ are driven to B and $\overline{B}$ respectively. Note that each of the four lines $\overline{B}/VBP_{1:4}$ drive F_{qi}, F_{si}, F_{ui} and F_{wi} respectively. We separate

the inputs of these flash transistors for programmability purposes (see Section IX.3.4.3). The pull-down NMOS (M_{prg}) is also used for programming the SF-LUT, which will also be discussed in Section IX.3.4.3. The notation X/Y is used throughout this chapter for a signal that is driven by X in normal operation, and by Y during programming.

IX.3.4.2 SF-LUT Configuration

Figure 9.10 shows how the function $F(A,B)$ is constructed in an SRAM-based FPGA (left) and how we construct the same function using SF-LUT (right). Constructing the LUT using a 4-input MUX that selects between GND, A, $\overline{A}$ and VDD (Figure 9.10, right) instead of using a regular 2 stage 2-input MUX (Figure 9.10, left) allows the design to save area by omitting the need for separate programmable memory cells to store the configuration bits [81]. In the SRAM-based LUT, we program the SRAM cells S0-S3 using the truth table of the function $F(A,B)$. In the SF-LUT structure, we program the threshold voltages of the flash transistors (F_p, F_q, F_r, F_s, F_t, F_u, F_v and F_w) by following the programming V_T's shown in Table 9.1. For example, if want to program the circuit in Figure 9.10 to implement the function $F(A,B)$ = A (corresponds to line 11 in Table 9.1), then we will program F_p, F_q, F_t, F_u, F_v and F_w to VT_2 (always off) and program F_r and F_s to VT_0 (always on). As a consequence, regardless of the value of B, the output $F(A,B)$ will always be A. Similarly, if we desire to program the SF-LUT with the function $F(A,B) = A \oplus B$ (corresponds to line 15 in Table 9.1), then we program F_p, F_q, F_v and F_w to VT_2 (always off), program F_r and F_s to VT_0 and VT_1 respectively (this path will pass A only when B = 0), and program F_t and F_u to VT_1 and VT_0 respectively to pass $\overline{A}$ only when B = 1. Ta-

ble 9.1 is used after the place and route step in the design flow to generate the SF-LUT's configuration bitstream.

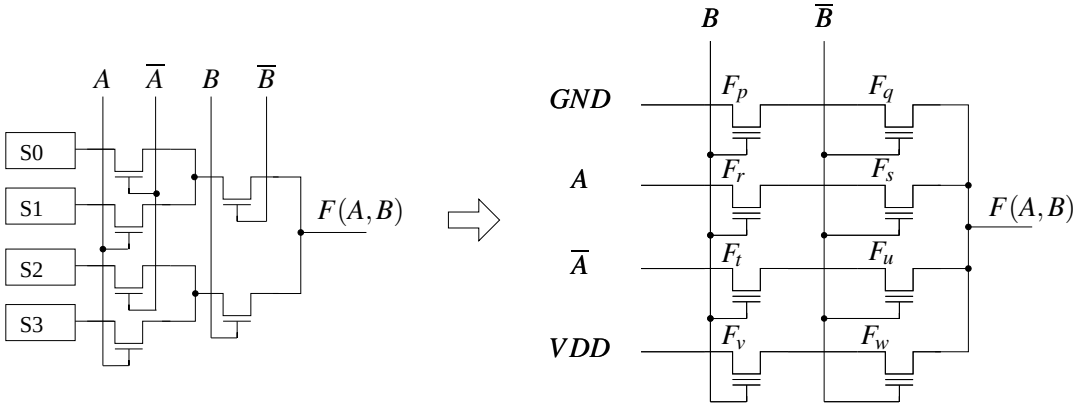


Figure 9.10: MUX Tree Implementation in SF-LUT

Table 9.1 is generalized by performing the cofactor of $F(A,B)$ against B and $\overline{B}$. If $F_B = A$ and $F_{\overline{B}} \neq A$, then F_r is programmed to VT_1 and F_s is programmed to VT_0, in order to drive the output with A, when the input is B. All other flash devices are programmed to VT_2, to block the GND, $\overline{A}$ and VDD signals from reaching the output. If $F_B = F_{\overline{B}} = A$, then F_r and F_s are programmed to VT_0, and all other transistors are programmed to VT_2, to block GND, $\overline{A}$ or VDD from reaching the output. The entries of the other rows in Table 9.1 can be determined in a similar manner.

IX.3.4.3 Programming the SF-LUT

The SF-LUT is programmed by first erasing its flash transistors, then programming their V_T's to desired levels. The erasing process (see Figure 9.11) is accomplished by

No.	F(A,B)		Threshold Voltage (V_T)							
	B = 1	B = 0	F_p	F_q	F_r	F_s	F_t	F_u	F_v	F_w
1	0	0	VT_0	VT_0	VT_2	VT_2	VT_2	VT_2	VT_2	VT_2
2	0	1	VT_1	VT_0	VT_2	VT_2	VT_2	VT_2	VT_0	VT_1
3	0	A	VT_1	VT_0	VT_0	VT_1	VT_2	VT_2	VT_2	VT_2
4	0	$\overline{A}$	VT_1	VT_0	VT_2	VT_2	VT_0	VT_1	VT_2	VT_2
5	1	0	VT_0	VT_1	VT_2	VT_2	VT_2	VT_2	VT_1	VT_0
6	1	1	VT_2	VT_2	VT_2	VT_2	VT_2	VT_2	VT_0	VT_0
7	1	A	VT_2	VT_2	VT_0	VT_1	VT_2	VT_2	VT_1	VT_0
8	1	$\overline{A}$	VT_2	VT_2	VT_2	VT_2	VT_0	VT_1	VT_1	VT_0
9	A	0	VT_0	VT_1	VT_1	VT_0	VT_2	VT_2	VT_2	VT_2
10	A	1	VT_2	VT_2	VT_1	VT_0	VT_2	VT_2	VT_0	VT_1
11	A	A	VT_2	VT_2	VT_0	VT_0	VT_2	VT_2	VT_2	VT_2
12	A	$\overline{A}$	VT_2	VT_2	VT_1	VT_0	VT_0	VT_1	VT_2	VT_2
13	$\overline{A}$	0	VT_0	VT_1	VT_2	VT_2	VT_1	VT_0	VT_2	VT_2
14	$\overline{A}$	1	VT_2	VT_2	VT_2	VT_2	VT_1	VT_0	VT_0	VT_1
15	$\overline{A}$	A	VT_2	VT_2	VT_0	VT_1	VT_1	VT_0	VT_2	VT_2
16	$\overline{A}$	$\overline{A}$	VT_2	VT_2	VT_2	VT_2	VT_0	VT_0	VT_2	VT_2

Table 9.1: SF-LUT Programming V_T's Configuration

floating each of VSP, $A/VAP1$, $\overline{A}/VAP2$ and VDP, drive prg to GND, drive B/VBP_0 and $\overline{B}/VBP_{1:4}$ to GND, and apply a high voltage (10V-20V) to the bulk. After the erasing process is complete, the V_T's of the flash transistors will be reset to VT_0. Unlike the erase process, the flash transistors in an SF-LUT are programmed individually. Assume that we want to program F_{p1} and F_{q1} in Figure 9.11, both to VT_2. We activate the path from F_{p1} and F_{q1} to M_{prg} (near the output of the LUT), in order to GND the source and drain terminals of F_{p1} and F_{q1}. We do this by driving prg to VDD (to turn M_{prg} on), C to GND (which turns M_{C1} on, and turns M_{C2} off). Note that this step may require programming the FPGA interconnect before programming the LUTs. To program F_{p1}, we float VSP,

$A/VAP1$, $\overline{A}/VAP2$, VDP and $\overline{B}/VBP_{2:4}$, drive $\overline{B}/VBP_1$ to a passing voltage (2V), drive B/VBP_0 to a programming voltage (10V-20V), drive the corresponding bulk to GND, and floating all other bulk islands. We can program F_{q1} by repeating the previous steps, except that we drive $\overline{B}/VBP_1$ to the program voltage and driving B/VBP_0 to GND.

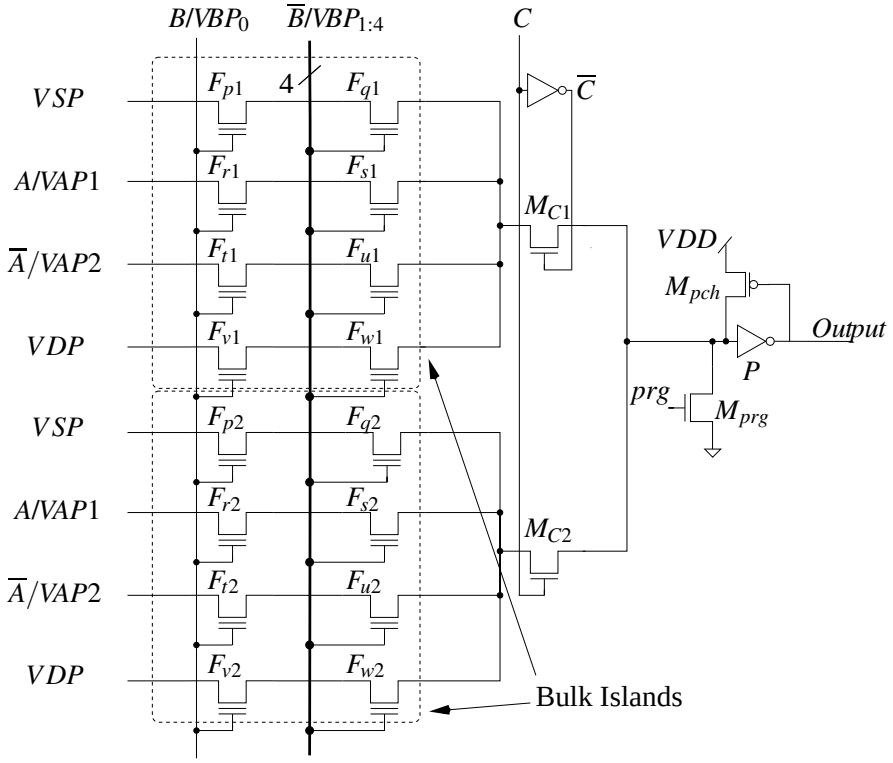


Figure 9.11: Proposed Static Flash-based LUT Structure (SF-LUT)

IX.3.5 Proposed Dynamic Flash-based LUT (DF-LUT)

IX.3.5.1 LUT Structure

As the name suggests, the DF-LUT is implemented using dynamic logic. There are a few advantages to implementing a flash-based LUT using dynamic logic. Our design uses

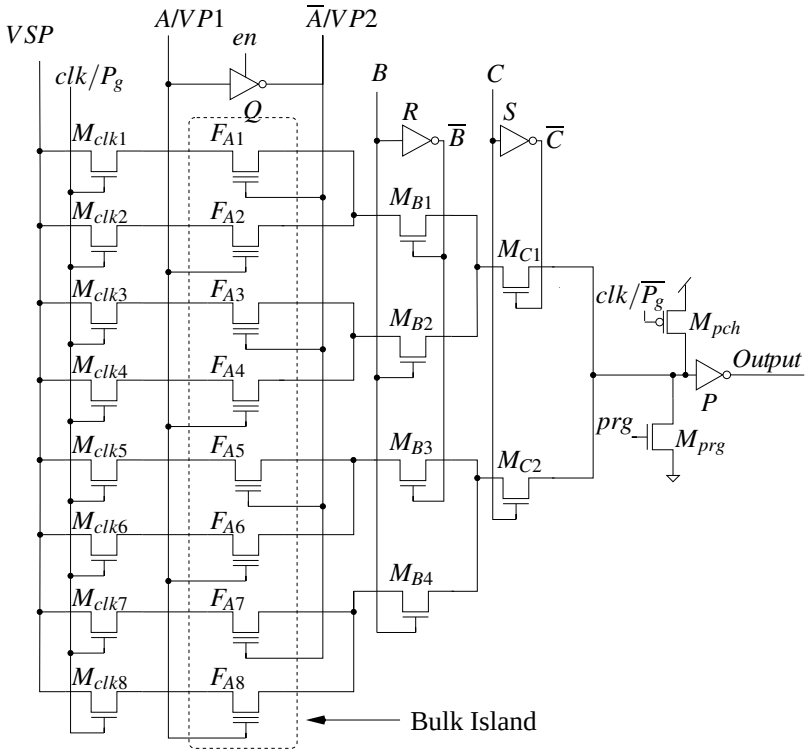


Figure 9.12: LUT Structure Used in DF-LUT

NMOS passgates (following common FPGA design practice [4, 34, 84, 6]), which are best at passing a logic 0. However, when they are used to pass a logic 1, they result in a drop in the output voltage by V_T, which slows down the LUT operation and requires the use of leaky lower switch point inverters to regenerate the output signals. Hence, an advantage to precharging the output to VDD (logic 1) is that we no longer have to pass a logic 1 through the NMOS passgates. Also, in FPGAs, we already have a clock distribution network, which can be utilized to implement the dynamic LUT structure. Figure 9.12 shows our proposed dynamic flash-based LUT (DF-LUT). We use a conventional MUX tree structure (similar to Figure 9.5). We replace the output low switch-point inverter and

PMOS keeper in Figure 9.5 with a regular minimum size inverter (P) and a precharge PMOS device (M_{pch}) driven by the clock signal ($clk/\overline{P_g}$). Also, to prevent creating a short-circuit between VDD and GND during the precharge cycle, we add evaluate NMOS transistors to each branch of the MUX tree driven by clk/P_g (these are shown on the left of Figure 9.12). This is done in order to pass the signal VSP (which is driven to GND during operation) during the evaluation phase (i.e. $clk/P_g = 1$) of normal operation. Note that while it is possible to only use one evaluate NMOS transistor at the output of the tree to prevent a short current between VDD and GND, the evaluate NMOS devices of Figure 9.12 additionally allow us to float the source nodes of the flash transistors during programming. Also, as shown in Figure 9.12 we use an NMOS pull-down (M_{prg}) at the output for programming purposes and a tri-state inverter (Q) to generate $\overline{A}$ from A during normal operation ($P_g = 0$) and to control the lines $A/VP1$ and $\overline{A}/VP2$ independently during programming ($P_g = 1$).

IX.3.5.2 DF-LUT Configuration

Configuring the DF-LUT is simpler than configuring the SF-LUT. For example, assume that the output of the LUT is logic 0 when the inputs A, B and C are all 0's, and the output is logic 1 for all other input combinations. Then we want program F_{A2} through F_{A8} to VT_1 (to turn on when they are driven to VDD) and program F_{A1} to VT_2 (to never turn on during operation, and hence cause the output to stay at logic 0 when the inputs $A = 0$, $B = 0$ and $C = 0$ are applied to the LUT). In general, we can reduce flash programming time (which is a slow process), by leaving F_{Ai} and $F_{A(i+1)}$ (assuming i is odd) at their erase

threshold voltage (VT_0) whenever the above configuration requires them both to be at VT_1. The reason behind this is that when a pair of branches (F_{Ai} and $F_{A(i+1)}$, for odd i) in the first stage of the DF-LUT both cause the output to be logic 1, then allowing these two branches to stay on at all the time (by leaving them at VT_0) will produce the same effect as well.

IX.3.5.3 Programming the DF-LUT

The flash transistors in the DF-LUT can be erased by driving the bulk to a high voltage (10V-20V), clk/P_g low, $clk/\overline{P_g}$ high, *prg* low and *en* low. This (respectively) turns off the evaluate transistors (M_{clk1}-M_{clk8}), the precharge transistor (M_{pch}), the program transistor M_{prg} and the inverter (Q). Then we drive the lines $A/VP1$ and $\overline{A}/VP2$ to GND. This resets all the V_T's of the flash devices F_{A1} through F_{A8} to VT_0. To program any of the flash transistors' threshold voltage to VT_1 or VT_2, we first select the path from the flash transistor (that we want to program) to M_{prg} (near the output) by applying the corresponding input combination (B and C in a 3-input LUT). Then, we drive *prg* to VDD (to turn on M_{prg}) and ground the selected path. We also drive clk/P_g and $clk/\overline{P_g}$ to GND and VDD respectively, to turn off the evaluate transistors (M_{clk1}-M_{clk8}) and the precharge transistor (M_{pch}). We finally, turn off the inverter (Q) by driving *en* low and drive $A/VP1$ to a program voltage and $\overline{A}/VP2$ to a passing voltage (2V) (or vice versa, depending on which of the two flash transistors in the selected pair we want to program). We repeat this process for each flash transistor we want to program in the DF-LUT.

IX.3.6 Configuration Time

One major aspect of difference between SRAM-based and flash-based FPGAs is the configuration time. Flash transistors erase and write delays are much higher than SRAM-cell write delays. Xilinx reports that the configuration time for a Kintex 7 SRAM-based FPGA is 1.83s (for an uncompressed bitstream file size of 91,584,896 bits), which is about 6.4μs per 32-bit word [82, 88]. To estimate the configuration time of a flash-based FPGA, we use the block erase and page program times for an SLC NAND flash memory manufactured by Micron [89]. Micron reports that for the MT29F2G08A SLC NAND flash memory, the block erase time is 700μs and the page write time is 200μs [89]. Therefore, we need 57s to configure a flash-based FPGA of the same size as the Kintex 7 FPGA. Our configuration time estimation for flash-based FPGAs aligns with Microsemi IGLOO2 flash-based FPGA programming times [90]. Note that the erase process will be done initially in parallel for the entire FPGA, thereby resulting in a minimal block erase time overhead (the number of blocks erased in parallel is limited only by the current required to perform the block erase process).

Since flash-based FPGA programming is slower, we propose that flash-based FPGAs be used in late stages of a project. This would allow the design to benefit from the rapid power-up of the flash-based FPGA, while ensuring that the number of program cycles for the flash transistors remains low (the maximum number of program cycles is in the 10K-100K range [40, 41]).

IX.3.7 Conventional SRAM-based Programmable Switch Structure

The interconnect fabric in the FPGA consists of variable length wires and programmable switches (in the SBs and CBs) that connect these wires together to form a path from a source to a destination in the FPGA, as shown in Figure 9.4. The source/destination can be either FPGA I/O port, LUT output or any other embedded processing elements in the FPGA. In SRAM-based FPGAs, these programmable switches are NMOS passgate transistors driven by a 5T SRAM cell similar to those in Figure 9.6. Due to the large area of 5T SRAM cells, each source in an SB can connect to one of three destinations based on the implemented topology, as discussed in Section IX.3.1.

IX.3.8 Conventional Flash-based Programmable Switch Structure

One way to implement (non-volatile) flash-based programmable switches is by replacing each SRAM cell with an FMC similar to that shown in Figure 9.7. The drawback in using such FMCs to implement the programmable switches is that the FMC will cause additional leakage when the programmable switches are programmed to turn be *on*. The leakage in this type of flash-based programmable switch is due to the V_T drop when the pullup transistor F_{pi} (shown in Figure 9.7) is passing VDD, which will cause the inverter X_i to be driven by a non-rail value, hence resulting in a high leakage current.

IX.3.9 Proposed Flash-in-path (FIP) Programmable Switch

IX.3.9.1 FIP Structure and Configuration

Our proposed Flash-in-path (FIP) programmable switch replaces the NMOS pass-gates (in an SB or CB) with flash transistors, eliminating the need for additional memory elements. This results in substantial area reduction, allowing additional interconnect configurability in the FPGA. Figure 9.13 shows an array of 4×4 FIP programmable switches connecting the wires $X1$, $X2$, $X3$ and $X4$ to the wires $Y1$, $Y2$, $Y3$ and $Y4$ respectively and driven by the lines $Z1$ through $Z4$ (which are driven to VDD during normal operation). Note that the programmable switches are grouped in vertical bulk islands (as shown in Figure 9.13) for programming purposes. Since the FIP switches are directly embedded in the path of signals, configuring the interconnect is simply done by programming their threshold voltage to VT_0 (to connect the path) or to VT_2 (to disconnect the path).

IX.3.9.2 Programming the FIP Array

The array of FIP programmable switches are erased by driving the bulk islands (B1 through B4) to a high voltage (10V-20V), floating the lines $X1$ through $X4$ and $Y1$ through $Y4$, and driving the lines $Z1$ through $Z4$ to GND. The erase process will reset the threshold voltages of all the flash transistors to VT_0 (erase threshold). While the erase process erases all the flash transistors in the FIP array, we program FIP array one column at a time. Assume that we want to program the top-most flash transistor inside the bulk island B1. This can be accomplished by driving the bulk island B1 to GND and floating the other bulk islands (B2, B3 and B4), driving the lines $X1$ through $X4$ and $Y1$ through $Y4$ to GND, and

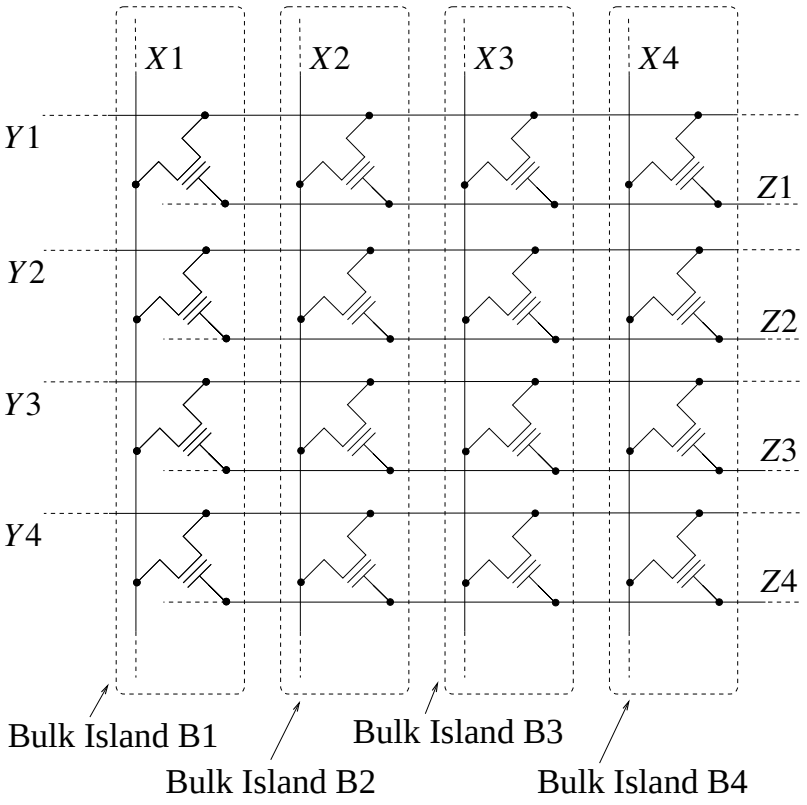


Figure 9.13: Array of Flash-in-path (FIP) Switches

driving the line $Z1$ to a programming voltage (10-20V) to program the threshold voltage for the top-most transistor of bulk island B1 to VT_2 (disconnected). The other lines ($Z2$ through $Z4$) are driven at a passing voltage (2V) to leave their threshold voltage at VT_0 (connected). As a consequence of this, signal $X1$ will be connected to signals $Y2$, $Y3$ and $Y4$.

In the next section, we will discuss our experimental results for each of the structures discussed earlier.

IX.4 Experiments

In this section, we first present the simulation environment used in evaluating our proposed flash-based FPGA LUT and interconnect switch design approaches. Then we discuss their circuit implementation details. Finally, we present the results of our experiments followed by a discussion of these results.

IX.4.1 Simulation Environment

The designs presented in this chapter are implemented in a 45nm process technology. The designs were simulated using the Synopsys HSPICE [56] circuit simulation tool and the 45nm PTM [57] card. The nominal supply voltage for the 45nm PTM card is 1V. We used custom scripts to generate the flash-based LUT and interconnect switch designs. For CMOS devices, we used the 45nm PTM model cards [57], while for flash devices, we used the same model cards described in Section III.4.2.

IX.4.2 LUT Implementation Details

In this section, we present the implementation details of the SRAM-based and flash-based LUT designs. The MUX tree in all of our candidate LUTs are constructed using minimum size NMOS transistors (W = 90nm and L = 45nm). The flash transistors in all of our flash-based LUTs also have W = 90nm and L = 45nm. The input drivers to our candidate LUT structures are appropriately sized to achieve optimal driving strength using the concept of logical effort [3]. For the SReg-LUT, FReg-LUT and SF-LUT, the output inverter is designed to have a low switch point, to regenerate the degraded signal coming

through the NMOS passgates. We found that a lower switch point inverter with the sizes Wn = 240nm, Ln = 45nm, Wp = 90nm and Lp = 45nm provides the best delay results for the SReg-LUT and FReg-LUT. For the SF-LUT, the optimal sizes of the low switch point inverter are Wn = 100nm, Ln = 45nm, Wp = 90nm and Lp = 45nm. The keeper device sizes are Wp = 90nm and Lp = 115nm for the SReg-LUT and FReg-LUT, and Wp = 90nm and Lp = 135nm for the SF-LUT. The DF-LUT uses a minimum size inverter at the output (Wn = 90nm, Ln = 45nm, Wp = 140nm and Lp = 45nm). For the interconnect programmable switch experiments, we used minimum size NMOS devices and minimum size flash transistors to implement the designs. The threshold voltages used in all of our flash-based designs are (VT_0 = -0.5 V), (VT_1 = 0.5 V) and (VT_2 = 1.5V).

IX.4.3 Results and Analysis

LUT Type	Delay (ps)	P_{Dyn} (uW)	E_{Dyn} (fJ)	P_{Static} (uW)
SReg-LUT	132.10	33.22	4.39	9.30
FReg-LUT	1.00×	1.01×	1.01×	0.99×
SF-LUT	0.90×	0.88×	0.79×	0.71×
DF-LUT	0.68×	2.02×	1.37×	0.22×

Table 9.2: Delay, Dynamic Power (P_{Dyn}), Dynamic Energy (E_{Dyn}) and Static Power (P_{Static}) Ratios of the LUTs

Table 9.2 shows the delay, dynamic power dissipation (P_{Dyn}), energy consumption (E_{Dyn}) and static power dissipation (P_{Static}) of each of the proposed flash-based LUT designs compared to a conventional SRAM-based LUT (SReg-LUT) implementation, whose

values are presented as absolute numbers, while other LUTs' values are shown relative to the values of the SReg-LUT. The FReg-LUT has a similar delay, dynamic power, dynamic energy and static power compared to the SReg-LUT design. Our proposed static flash-based LUT structure shows improvements over both the SReg-LUT and the FReg-LUT in terms of performance, power and energy. The SF-LUT shows 10% faster performance, 12% lower dynamic power dissipation, 21% lower dynamic energy and 29% lower static power dissipation compared to the SReg-LUT. The DF-LUT exhibits 32% faster operation and 78% lower static power dissipation, with a penalty of 37% in dynamic energy consumption compared to the SReg-LUT. For high performance applications, a dynamic flash-based implementation is the optimal choice.

The key reason for these improvements is that in the SF-LUT and DF-LUT, the programming devices are embedded in the logic of the LUT itself.

Programmable Switch Type	Delay (ps)	P_{Dyn} (nW)	P_{Static} (nW)	P_{Total} (nW)
SRAM-based	13.39	53.82	7.99	13.95
FMC-based	0.89×	0.58×	0.43×	0.51×
FIP-based	0.11×	0.71×	0.02×	0.29×

Table 9.3: Delay, Dynamic Power (P_{Dyn}), Static Power (P_{Static}) and Total Power (P_{Total}) Ratios of the Programmable Switches

Table 9.3 shows the delay, dynamic power dissipation (P_{Dyn}), static power dissipation (P_{Static}) and total power dissipation (P_{Total}) of each of the proposed flash-based programmable switch designs compared to a conventional SRAM-based programmable

switch implementation. The FMC-based programmable switch shows 11% faster operation, 42% lower dynamic power, 57% lower static power and 49% lower total power dissipation compared to the SRAM-based programmable switch. The FIP-based programmable switch, however, shows very promising results. It has 89% lower delay, 29% lower dynamic power dissipation, 98% lower static power dissipation and 71% lower overall power dissipation compared to the SRAM-based programmable switch.

The reason for these improvements is that the FIP-based switch, when turned on, has a high gate drive (of VDD + VT_0). The power dissipation is lower in both the flash-based programmable switch designs compared to the SRAM-based programmable switch due to their reduced number of devices.

Since the V_T levels of the flash transistors are adjustable with a fine granularity, we can adjust the performance characteristics of the flash-based LUTs by adjusting their V_T values. We performed a sweep of the V_T values for the flash-based LUT designs, to show the effect of varying V_T on the delay, power dissipation and energy consumption of the flash-based LUT. Figure 9.14 shows the normalized delay, dynamic power dissipation, dynamic energy consumption and static power dissipation of the SF-LUT as the V_T values of VT_0, VT_1 and VT_2 are varied around their nominal values. The delay curve in Figure 9.14 suggests that up to $\sim$14% faster operation can be achieved by lowering the V_T values of the flash transistors. This improved performance comes with a penalty of about $\sim$18% higher dynamic power dissipation and about $\sim$11% higher static power dissipation. The energy stays substantially flat, which is a desired feature for battery powered as well as

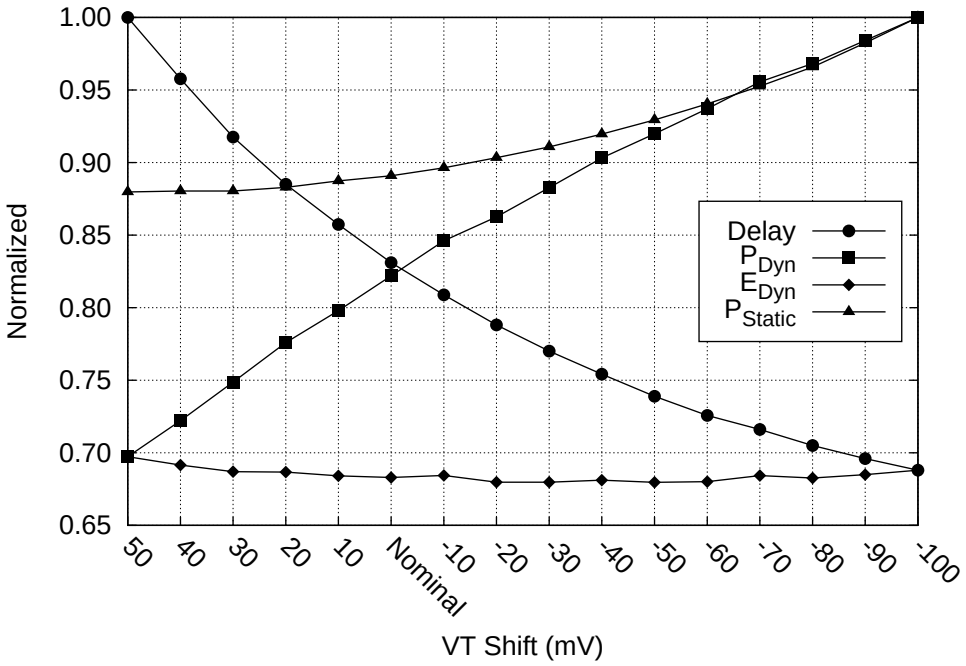


Figure 9.14: Normalized Delay, Dynamic Power, Dynamic Energy and Static Power of SF-LUT as the V_T is Shifted.

tethered computing platforms.

One of the issues of using flash transistors is the issue read and write disturbs. In our flash-based FPGA, we suppress the issue of read and write disturbs by:

- We use three threshold voltage levels (VT_0, VT_1 and VT_2), such that the difference between any two adjacent threshold voltage levels is the same as that of an SLC cell, which has exponentially more immunity to read and write disturbs [43].
- Our structures are limited to 1 or 2 flash transistors in series, reducing the susceptibility to disturbs.
- In our implementation, we limit the operating supply voltage to 1V, resulting in

reduced electric fields, which in turn reduces read disturbs effects on the flash transistors..

- In our flash-based FPGA, we do not use passing voltages (which are higher than regular read voltages), since we read all the flash devices in series at once.

IX.5 Chapter Summary

FPGAs serve as the platform of choice for low and medium volume digital designs, as well as designs that require in-field modifications. However, the prolonged boot time for SRAM-based FPGAs (due to the need to load the configuration bits into the SRAM cells) has motivated the design and manufacturing of (non-volatile) flash-based FPGAs. In this chapter, we presented the design and implementation of flash-based FPGA LUT and interconnect fabrics. Our proposed static flash-based LUT structure provides 10% faster operation, 12% lower dynamic power dissipation, 21% lower energy consumption and 29% lower static power dissipation compared to a traditional SRAM-based LUT. Our dynamic flash-based LUT can achieve further performance improvements (32% lower delay) with a higher energy consumption (37% higher) compared to an SRAM-based LUT. We have also shown that our flash-based interconnect structure yields 89% lower delay and 71% lower overall power consumption compared to the traditional interconnect structure used in SRAM-based FPGAs.

CHAPTER X

THESIS SUMMARY AND CONCLUSIONS

Flash transistors are the workhorse technology for non-volatile data storage applications today. However, there has been no previous research in the use of flash technology to implement digital logic. In this thesis, we presented four different design approaches to implement flash-based designs at the cell-level. These different designs approaches are ternary-valued design approach using a TLC, a binary design approach using an FC, a PLA-like design approach using a PFC, and a multi-valued design approach using an MVFC. We summarize and compare the results obtained from these flash-based design approaches in Section X.1.

Additionally, we have presented a design flow, with optimization, to implement a flash-based design at the block-level. We adapted this design flow to use FCs to implement flash-based designs at the block level. This flow can be modified to use TLCs, PFCs or MVFCs as well. We presented the algorithmic details to implement complete flash-based digital circuit blocks, in the form of an interconnected network of FCs. We presented techniques to perform logic clustering, on-the-fly physical synthesis of all the FCs of a design, and the automatic characterization of the delay, power and area of the resulting circuit. We also presented a method to perform SAT-based optimization of the flash-based digital block. This optimization technique uses the CODC of a multi-level logic network. Unlike other don't-care optimization techniques presented in the past, our technique per-

forms post-technology mapped optimization, which yields direct improvements in result quality as compared to pre-technology mapped optimization. Also, we optimize a cluster of nodes at once, instead of optimizing nodes one at a time.

We also presented novel flash-based FPGA designs to implement FPGAs. FPGAs serve as the platform of choice for low and medium volume digital designs, as well as designs that require in-field modifications. However, the prolonged boot time for SRAM-based FPGAs (due to the need to load the configuration bits into the SRAM cells) has motivated the design and manufacturing of (non-volatile) flash-based FPGAs. In this thesis, we presented the design and implementation of both dynamic and static flash-based FPGA LUT structures as well as an ultra low power flash-based interconnect fabric.

X.1 Choosing the Right Flash-based Design Approach

Approach	Chapter	Delay Ratio	Power Ratio	Energy Ratio	Cell Area Ratio
TLC	III	2.43×	0.12×	0.30×	0.95×
FC	IV	0.84×	0.35×	0.30×	0.54×
PFC	VII	0.85×	0.39×	0.33×	0.46×
MVFC	VIII	0.77×	0.95×	0.74×	0.96×

Table 10.1: A Comparison Between Flash-based Design Approaches to Implement Digital Circuits

Table 10.1 shows a list of the different flash-based design approaches presented in this thesis. The table also shows the delay, power and cell area ratios of these design approaches compared to a CMOS standard cell design approach. The delay, power and

area results shown in the table are the average results obtained from characterizing 20 randomly generated designs which were implemented using a CMOS standard cell design approach (reference), as well as the ternary-valued flash-based (Chapter III), binary flash-based (Chapter IV), PLA-like flash-based (Chapter VII)) and multi-valued flash-based (Chapter VIII) design approaches. The set of 20 designs used to perform our comparison of flash-based designs to a CMOS standard cell counterpart is the same set used for all the various flash-based design approaches presented in this thesis (TLC-based, FC-based, PFC-based and MVFC-based flash-based designs).

Table 10.1 shows that TLC-based designs have the lowest power dissipation across the board ($0.12\times$), which comes at the expense of increased delays ($2.43\times$). Therefore, TLC-based designs target extreme low power applications that have relaxed delay requirements. Also, TLC-based designs have similar area footprint compared to CMOS. FC-based designs show lower delays ($0.84\times$) and lower power ($0.35\times$) compared with CMOS as shown in Table 10.1. This makes the FC-based designs appealing candidates for power-aware, high performance applications due to their superior delays and low power. We also note that the delay of FC-based designs is much lower than the delay of the TLC-based designs, while the power dissipation in FC-based designs is higher than TLC-based designs. However, the energy consumption in both designs are similar. Additionally, the cell area of the FC-based designs is much lower than that of the TLC-based designs, which makes the FC-based designs, in general, more appealing than the TLC-based designs to implement digital circuits. Table 10.1 also shows that the delay of PFC-based designs is

0.85× the delay of CMOS, while the power dissipation in PFC-based designs is 0.39× the power dissipation in CMOS, which results in the energy of PFC-based designs being 0.33× the energy in CMOS. The delay of the PFC is substantially similar to the delay of the FC. However, the power and energy are ∼11% higher in the PFC compared to the FC. The main advantage for PFC-based designs is in the small cell area (0.46× the cell area of CMOS), which is the smallest cell area across all the flash-based designs (18% smaller than the FC). PFC-designs are appealing for high performance applications that have tight area constraints. Finally, Table 10.1 shows the delay, power and area of the MVFC-based. MVFC-based designs show superior delay results (0.77× the delay of CMOS), which is the lowest across the board. However, the power dissipation and energy consumption of MVFC-based designs are the highest compared to the other flash-based designs. This makes the MVFC-based designs a viable candidate for high performance applications that have very tight delay constraints with more relaxed power and energy constraints. The area of MVFC-based designs is higher than FC-based and PFC-based designs, which adds an additional metric to consider when choosing the candidate flash-based design approach for high performance applications.

In general, TLC-based design approach is appealing for extreme low power applications. FC-based design approach is appealing for high performance application which have balanced delay and power requirements. PFC-based design approach is appealing for high performance applications which have tight area constraints. MVFC-based design approach is appealing for high performance applications with very tight delay requirements.

CHAPTER XI

FUTURE WORK

The work presented in this thesis is the first work to discuss the use of floating gate transistors to implement digital circuits. We addressed the circuit structure as well as the CAD flow to implement flash-based digital circuits. In this section, we discuss some open ideas that can be implemented in the future to further improve the flash-based digital design approach presented in this thesis.

XI.1 Flash Technology Scaling

In this thesis, we explored the flash-based design approach at the 45nm technology node for both of the CMOS standard cell and the flash-based implementations. The main reason for our choice of technology node was the availability of an industry CMOS standard cell library (Nangate standard cell library [55]) and the corresponding process design kit (FreePDK [54]) as well as the electrical characteristics for 45nm flash fabrication process [46, 45, 58]. Although this work is done at the 45nm technology node, flash-based designs have been and can be implemented using technology nodes with finer feature sizes. Floating transistor technology has substantially been following CMOS technology node scaling trends. Currently, memory devices are being fabricated at sub-20nm minimum feature dimensions, similar to CMOS technology nodes [60, 61, 62, 63, 91]. The need for smaller and more compact non-volatile memory has been the main driving

factor to the advancement of floating gate transistors technology.

As the flash technology is scaled, endurance and data retention issues have been on the rise [92]. As flash technology nodes are advancing and the feature size of floating gate transistors are shrinking, the number of program and erase (P/E) cycles (i.e. endurance) have been dropping. The issue of endurance limits the number of times the cell can be erased and programmed. This issue is very challenging for the implementation of flash memories that use triple-level cell (TLC) and multi-level cell (MLC) memories. This issue of endurance is more amplified in MLC and TLC flash cells, since these types of flash cells use large number of V_T levels (4 V_T levels in MLC and 8 V_T levels in TLC) compared to only 2 V_T levels in SLC cells (used in our work). The larger the number of V_T levels in a cell makes the programming of the cell harder (i.e. requires more program cycles) to accurately set the V_T level of that cell. The accuracy of the V_T level value is important since error margins become smaller as more V_T levels are packed between VDD and GND. However, in our flash-based design approach (more specifically in FC, PFC and MVFC) we use SLC cells which use 2 V_T levels (program V_T which is between VDD and GND and erase V_T below GND). This makes our error margins substantially high and the programming of the V_T levels in our work faster. Additionally, unlike in memory, which is continuously erased and programmed to store new data, our flash-based design only requires a few programming cycles at the factory for configuring the circuit and for binning purposes, and possibly in the field for aging mitigation and performance customization. This makes are flash-based design approach immune to endurance issues

arising due to the scaling of floating gate transistors. The other issue that appears due to the technology scaling of flash transistors is data retention, which can be addressed in a similar fashion as aging of ICs is handled. As the V_T levels of the flash-based design drift, cause the circuit to slow down or consume higher power than the specifications, the flash circuit is reprogrammed and the V_T levels are refreshed to their specified values to achieve the desired performance specifications. This way, we overcome both data retention as well as aging issues.

XI.2 3D NAND Technology

Recently, a new 3D NAND flash technology has been developed [93, 94, 95, 96, 97, 98]. This technology allows the fabrication of vertically stacked flash transistors in a die to build an ultra-dense NAND flash memory. Samsung has also produced non-volatile memory using 3D NAND flash technology (called V-NAND [99]). This innovation in NAND flash device technology can be exploited to enhance our flash-based digital circuit design approaches, and enables the implementation of monolithic 3D digital circuits that expand in the vertical direction. In the flash-based design implementations presented in this thesis, we use NAND flash-like stacks for length 4, 6 and 8 flash transistors per stack. Implementing these stacks using 3D NAND technology would achieve substantial area reductions and allow the flash-based designs to deliver even better area improvements over CMOS.

XI.3 Static Flash-based Implementation Using P-type Flash

In this thesis we presented dynamic flash-based digital circuits. The dynamic implementation has many advantages such as compact area, high performance and the absence of P-type flash devices. In [100] the authors consider the use of P-type flash transistors to implement NAND flash memory. They show that better endurance can be obtained for non-volatile memory implemented using P-type flash memory, compared to non-volatile memory implemented using N-type flash memory. However, as they mention, P-type flash devices need careful doping concentration control. The work in [100] paves the path for using P-type flash transistors in implementing flash-based designs, potentially enabling static flash-based digital designs. Static implementations have several advantages over dynamic implementation since they do not require the use of a clock and typically have lower power dissipation than dynamic implementations. Static flash-based implementations can be obtained by implementing a circuit that uses a similar pulldown structure as presented in this thesis. However, the pullup structure can be constructed as the dual of the pulldown structure, using the P-type flash transistors. The circuit topology would resemble traditional static CMOS logic circuits. Also, careful design consideration is required to allow the programming of both the pulldown (N-type) and pullup (P-type) structures independently. The circuit structure would need to be evaluated to determine the number of inputs and outputs of the flash-based logic cell. Also, the choice of standard cells versus generic cells (that can implement any function) will need to be made.

XI.4 Preset V_T Levels to Meet Application Needs

In this work, we demonstrated the ability to change the delay, power and energy characteristics of the flash-based digital circuits by shifting the V_T level values of the flash transistors. Shifting the V_T level to higher values results in a decrease of the power dissipation and an increase of the delay of the flash-based digital circuit. Conversely, shifting the V_T level to lower values results in a decrease in the circuit delay and an increase in the power dissipation of the flash-based digital circuit. This feature can be utilized to serve the needs of the application implemented using the flash-based digital circuit. For example, chip manufacturers reduce the cost of fabricating digital ICs by reusing the same design for different applications. If the design is intended for high performance applications, the fabricated chip can be configured to operate at a high frequency. However, if the design is intended for low power applications, then the chip is configured to operate at a low frequency to conserve power. In fact, different portions of the same IC can utilize different V_T values if their performance requirements differ.

Flash-based digital circuits add an additional layer of configuration to allow maximum flexibility in tuning the fabricated chip to meet the application requirements. This additional layer of configuration can be achieved by tuning the V_T levels of the flash transistors to meet certain application requirements. This layer of configuration can be used to perform chip binning at the factory as well as to allow the manufacturer or the end-user to tune the performance of the chip in the field.

The manufacturer can choose the V_T levels of the flash transistors that places the

chip in a fast bin or in a low power bin. This can only be done at the factory and does not require the chip to have any special firmware or memory to hold the configuration since the flash transistors are non-volatile. However, if more flexibility is desired the chip can be designed such that it stores multiple V_T level settings on the chip and selectively program the flash transistors of the chip to a V_T level that matches the application needs. This capability is easily achieved by developing a firmware that chooses which V_T level to program the flash transistors with, potentially at boot time. Since programming the chip is a slow process, this programming process is typically done at the factory. However, if further flexibility is desired, the end-user can be allowed access to the firmware (with limitations) in order to take advantage of this configuration capability. The user access to the firmware may be limited for reliability reasons. For example, the firmware should only allow the user to program the V_T levels of the flash transistors in the chip to tested values that are known to be safe and meet system constraints such as thermal and electrical limits. This can be done by preloading tested V_T levels onto an on-chip PROM (or fuse array). These V_T levels need to be tested at the factory, and have to be known to operate the chip without any issues.

XI.5 Replacing Always-on Flash Transistors with Metal Wires

The TLC, FLB and MVLB designs shown in Sections III.3.3.2, IV.3.2.2 and VIII.3.6 respectively, consist of NAND flash-like pulldown stacks each consist of 8, 6 and 4 flash transistors respectively. As mentioned earlier, the flash transistors in these pulldown stacks

are programmed to VT_0 when they are required to be always "on", since the VT_0 level is lower than GND. In other words, if the function of the pulldown stack does not depend on a certain input, we program the flash transistor driven by this input to VT_0. Figure 11.1 shows an example pulldown stack that has 4 flash transistors. In the figure, the flash transistors F_a and F_c are programmed to VT_1 indicating the inputs a_1 and c_1 control the function of the pulldown stack. Conversely, the flash transistors F_b and F_d are programmed to VT_0 indicating that the inputs b_0, b_1, d_0 or d_1 do not affect the function of the pulldown stack. In other words, the state of the flash transistors F_a and F_c depends on the value of their inputs and the state of the flash transistors F_b and F_d are always "on". Although the flash transistors F_b and F_d do not have any functional effect on the pulldown stack shown in Figure 11.1, they have an electrical effect by limiting the current passing through the stack. In effect, they increase the evaluation time of the pulldown stack.

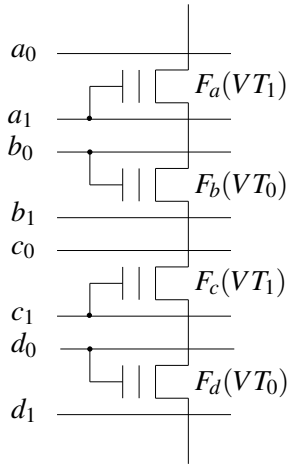


Figure 11.1: NAND Flash-like Pulldown Stack Used in Our Flash-based Designs

Recall that the flash transistors that are programmed to VT_0 are always turned "on". Therefore, we can improve the evaluation time of the pulldown stacks that have flash transistors programmed to VT_0 by removing these transistors and replacing them with metal wires as shown in Figure 11.2. Since the flash transistors F_b and F_d in Figure 11.1 are programmed to VT_0, we removed these two flash transistors and replaced them with a metal wire as shown in Figure 11.1. Note that the flash transistors F_a and F_c are left unchanged in Figure 11.1. This change results in making the pulldown stack in Figure 11.2 shorter than the original pulldown stack in Figure 11.1, which results in improving the evaluation time of the pulldown stack. Note that to keep the layout of flash-based design pitch-matched, when we remove the flash transistors F_b and F_d, we leave the flash transistors F_a and F_c in place, simply connecting the source of the flash transistor F_a and the drain of the flash transistor F_c with a metal wire. This pitch-matching is very important since we route the inputs of the flash transistors horizontally and need to place the flash transistors near their inputs.

XI.6 Customized Espresso-MV for Multi-valued Flash-based Design

In Chapter VIII, we have shown how to implement multi-valued flash-based digital circuits using the MVFC structure. In this implementation method, we use Espresso-MV to minimize the implemented logic function as shown in Section VIII.3.7. The output of Espresso-MV minimization is shown in Table 8.1. Note that out of all the possible literals of the ternary-valued inputs or outputs (001, 010, 100, 011, 101, 110 and 111) the

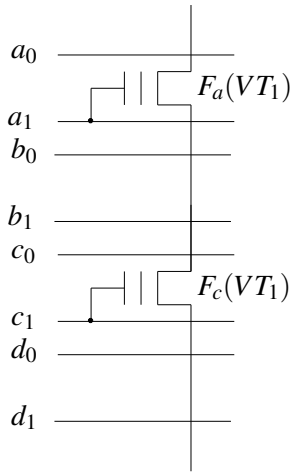


Figure 11.2: Proposed NAND Flash-like Pulldown Stack

MVFC structure can only implement (001, 010, 100 and 111). If the literal was either 011, 101 or 110, then we have to split the literal into two outputs of type 001, 010 or 100. Therefore, it will be pointless to have Espresso-MV produce the literals (011, 101 and 110) since they cannot be implemented by the MVFC structure. Instead, we can guide the minimization towards producing the output 111. This can be easily done by reimplementing the Espresso-MV code so that it does not generate the illegal literals.

XI.7 Delay Driven Optimization

In Chapter VI we discussed a SAT-based optimization technique that aims towards the reduction of the area utilization of the flash-based design (area-driven optimization). We also reported the results of the optimization for K = 2 in Table 6.1 and for K = 3 in Table 6.2. The optimization results show that when the parameter K is increased, the area of the flash-based design decreased, however the delay of the flash-based design increased.

This is because the optimization we run on the flash-based design is area-driven. However, for applications that have high performance requirements, a delay-driven optimization is required. We can enhance our optimization technique discussed in Chapter VI by introducing delay-driven optimization, granting the user the choice optimization goal (delay or area).

In Section VI.3.3.1 we showed how we initially create a list of candidate optimizations to perform on the flash-based design. We order the list based on the area reduction achieved by each of the optimization candidates. Although these candidate optimizations result in an area reduction of the design, some of them cause the delay of the design to increase, especially if the optimization is performed on the critical path of the design. We can perform delay-driven optimization by considering the effects of the optimization at the cell-level and the block-level.

In Section VI.3.3.1, our optimization is done by moving cubes from FLBs that have fewer cubes (1 or 2 cubes) to form FLBs that have a larger number of cubes (2 or 3 cubes). This results in an area improvement at the cell-level since FLBs that implement a larger number of cubes have a lower area utilization per cube. This area-driven optimization is performed at the cell-level, and the area reduction obtained by performing this optimization directly translates into an area reduction at the block-level. We can perform delay-driven optimization by constructing the list of optimization candidates in a different manner, such that the delay of the flash-based design at the cell-level is minimized. The delay of the cell increases monotonically with the total number of cubes implemented in each

cell as well as the size of the largest FLB in the cell. Therefore, to perform delay-driven optimization, we prune the optimization candidate list to only perform cube elimination (by moving cubes into the precharge state,since this decreases the total number of cubes implemented in the cell). We guarantee that the size of the largest FLB is never increased, by only selecting the optimization candidates that move cubes from one FLB to another in a manner that does not increase the size of any FLB in the cell beyond the size of the largest FLB in that cell. For example, if the largest FLB size in a cell is 2, then we will only allow moving a cube from any FLB to another FLB of size 1, since this move would result in the formation of an FLB of size 2 (which is the same size of the largest FLB in the cell). We would forbid moving cubes into FLBs of size 2, since it would form an FLB of size 3, thus increasing the delay of the cell (since an FLB of size 3 is larger than the largest FLB in the original cell before the optimization). A proposed optimization candidate list for delay-driven optimization would be as follows.

A Optimization in which cubes of type FLB-A are moved to FLA_7.

B Optimization in which cubes of type FLB-B are moved to FLA_7.

C Optimization where a cube of type FLB-C is moved to FLA_7.

D Optimizations where a cube of FLB-C is moved to another cube of FLB-C if the largest FLB is of type FLB-B or FLB-A.

E Optimization in which a cube of type FLB-C is moved to a cube of type FLB-B if the largest FLB is of type FLB-A.

In addition to the optimization done at the cell-level as discussed in the previous

paragraph, we can perform delay-driven optimization at the block-level by extracting the longest paths of the circuit (including the critical path) and performing delay-driven optimization on the cells on these paths. For the rest of the design, we could perform area-driven optimization, which will not affect the maximum delay of the design but will result in more area reduction than performing delay-driven optimization on the entire design. For these cells on the critical path, one can also target the largest FLBs in a cell, and try to move its cubes so that the size of the largest FLB in the cell is reduced.

The approaches presented in this thesis for cell design (FC, PFC, TLC and MVFC) and FPGA design (SF-LUT, DF-LUT and the FIP programmable switch) were all accompanied by a discussion on programming. Our goal was to demonstrate feasibility of the programming task. Alternate circuit topologies and programming algorithms can be conceived to reduce the total programming time. The simplest among techniques would employ multiple programming hardware units, which program multiple sections of flash transistors in parallel.

REFERENCES

[1] "Simon, W. G., Transistor Count and Moore's Law 2011 (Accessed on 1 December 2016)." `https://commons.wikimedia.org/wiki/User:Wgsimon`.

[2] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, pp. 82–85, Jan 1998.

[3] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. USA: Addison-Wesley Publishing Company, 4th ed., 2010.

[4] M. Abusultan and S. Khatri, "Look-up table design for deep sub-threshold through full-supply operation," in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, May 2014.

[5] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pp. 29–40, Feb 2002.

[6] M. Abusultan and S. P. Khatri, "Delay, power and energy tradeoffs in deep voltage-scaled FPGAs," in *Proc. of the 25th Great Lakes Symposium on VLSI*, 2015.

[7] G. Patounakis, Y. W. Li, and K. L. Shepard, "A fully integrated on-chip DC-DC conversion and power management system," *IEEE Journal of Solid-State Circuits*, vol. 39, pp. 443–451, March 2004.

[8] A. Elshennawy and S. P. Khatri, "An asynchronous network-on-chip router with low standby power," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pp. 394–399, Oct 2014.

[9] R. Gonzalez, B. M. Gordon, and M. A. Horowitz, "Supply and threshold voltage scaling for low power CMOS," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 1210–1216, Aug 1997.

[10] M. Onabajo and J. Silva-Martinez, *Analog Circuit Design for Process Variation-Resilient Systems-on-a-Chip*. Springer US, 2012.

[11] V. Kheterpal, V. Rovner, T. G. Hersan, D. Motiani, Y. Takegawa, A. J. Strojwas, and L. Pileggi, "Design methodology for IC manufacturability based on regular logic-bricks," in *Proceedings. 42nd Design Automation Conference, 2005.*, pp. 353–358, June 2005.

[12] K. Y. Tong, V. Kheterpal, V. Rovner, L. Pileggi, and H. Schmit, "Regular logic fabrics for a via patterned gate array (VPGA)," in *Proceedings of the IEEE 2003 Custom Integrated Circuits Conference, 2003.*, pp. 53–56, Sept 2003.

[13] L. Pileggi, H. Schmit, A. J. Strojwas, P. Gopalakrishnan, V. Kheterpal, A. Koorapaty, C. Patel, V. Rovner, and K. Y. Tong, "Exploring regular fabrics to optimize the performance-cost trade-off," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pp. 782–787, June 2003.

[14] Y. Ran and M. Marek-Sadowska, "On designing via-configurable cell blocks for regular fabrics," in *Proceedings. 41st Design Automation Conference, 2004.*, pp. 198–203, July 2004.

[15] D. G. Chinnery and K. Keutzer, "Closing the gap between ASIC and custom: An ASIC perspective," in *Proceedings of the 42nd Annual Design Automation Conference*, DAC '05, pp. 275–280, ACM, 2005.

[16] B. H. Calhoun, Y. Cao, X. Li, K. Mai, L. T. Pileggi, R. A. Rutenbar, and K. L. Shepard, "Digital circuit design challenges and opportunities in the era of nanoscale CMOS," *Proceedings of the IEEE*, vol. 96, pp. 343–365, Feb 2008.

[17] L. V. den Hove, A. M. Goethals, K. Ronse, M. V. Bavel, and G. Vandenberghe, "Lithography for sub-90nm applications," in *Digest. International Electron Devices Meeting,*, pp. 3–8, Dec 2002.

[18] W. Ye, B. Yu, D. Z. O. Pan, Y.-C. Ban, and L. Liebmann, "Standard cell layout regularity and pin access optimization considering middle-of-line," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pp. 289–294, ACM, 2015.

[19] L. Capodlieci, P. Gulpta, A. B. Kahng, D. Sylvester, and J. Yang, "Toward a methodology for manufacturability-driven design rule exploration," in *Proceedings. 41st Design Automation Conference, 2004.*, pp. 311–316, July 2004.

[20] K.-C. Wu and Y.-W. Tsai, "Structured ASIC, evolution or revolution?," in *Proceedings of the 2004 International Symposium on Physical Design*, ISPD '04, pp. 103–106, ACM, 2004.

[21] S. P. Khatri, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric," in *Proceedings of the 2000 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '00, pp. 412–419, 2000.

[22] S. P. Khatri, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, *Cross-talk Noise Immune VLSI Design using Regular Layout Fabrics*. Norwell, MA, USA: Kluwer Academic Publishers, 1984.

[23] S. Gopalani, R. Garg, S. P. Khatri, and M. Cheng, "A lithography-friendly structured ASIC design approach," in *Proceedings of the 18th ACM Great Lakes Symposium on VLSI*, GLSVLSI '08, pp. 315–320, 2008.

[24] S. Gopalani, R. Garg, S. P. Khatri, and M. Cheng, "DFM-aware structured ASIC design," in *Proceedings of the 2009 12th International Symposium on Integrated Circuits*, pp. 29–32, Dec 2009.

[25] K. Gulati, N. Jayakumar, and S. P. Khatri, "A ptl based highly testable structured asic design approach," in *Proceedings of the 2009 12th International Symposium on Integrated Circuits*, pp. 33–36, Dec 2009.

[26] S.-M. S. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits Analysis and Design*. New York, NY, USA: McGraw-Hill, Inc., 3 ed., 2003.

[27] "Circuit design techniques for a gigahertz integer microprocessor," in *Proceedings of the International Conference on Computer Design*, ICCD '98, (Washington, DC, USA), pp. 11–, IEEE Computer Society, 1998.

[28] K. Gulati, N. Jayakumar, and S. P. Khatri, "A structured ASIC design approach using pass transistor logic," in *2007 IEEE International Symposium on Circuits and Systems*, pp. 1787–1790, May 2007.

[29] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984.

[30] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, pp. 727–750, September 1987.

[31] S. Trimberger, ed., *Field-Programmable Gate Array Technology*. Netherlands: Kluwer Academic Publishers Group, 1994.

[32] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-programmable gate arrays*. Norwell, MA, USA: Kluwer Academic Publishers, 1992.

[33] M. Abusultan and S. P. Khatri, "Exploring static and dynamic flash-based FPGA design topologies," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pp. 416–419, Oct 2016.

[34] M. Abusultan and S. P. Khatri, "A comparison of FinFET based FPGA LUT designs," in *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI*, 2014.

[35] M. Abusultan, S. Harkness, B. J. LaMeres, and Y. Huang, "FPGA implementation of a bartlett direction of arrival algorithm for a 5.8GHz circular antenna array," in *2010 IEEE Aerospace Conference*, pp. 1–10, March 2010.

[36] B. J. LaMeres, R. J. Weber, Y. Huang, M. Abusultan, and S. Harkness, "Design and test of FPGA-based direction-of-arrival algorithms for adaptive array antennas," in *2011 Aerospace Conference*, pp. 1–8, March 2011.

[37] A. Sangiovanni-Vincentelli, "The tides of EDA." Keynote Talk, Design Automation Conference, June 2003.

[38] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, FPGA '06, pp. 21–30, ACM, 2006.

[39] R. Fowler and L. Nordheim, "Electron emission in intense electric fields," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 119, pp. 173–181, May 1928.

[40] D. Jung, Y.-H. Chae, H. Jo, J.-S. Kim, and J. Lee, "A group-based wear-leveling algorithm for large-capacity flash memory storage systems," in *Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '07, pp. 160–164, ACM, 2007.

[41] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, (Berkeley, CA, USA), pp. 9–9, USENIX Association, 2010.

[42] S. Aritome, *NAND Flash Memory Technologies*. Wiley-IEEE Press, 2015.

[43] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, "Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 438–449, June 2015.

[44] K. Takeuchi, "Novel co-design of nand flash memory and NAND flash controller circuits for sub-30nm low-power high-speed solid-state drives (SSD)," *Solid-State Circuits, IEEE Journal of*, vol. 44, pp. 1227–1234, April 2009.

[45] S. G. Jung, K. W. Lee, K. S. Kim, S. W. Shin, S. S. Lee, J. C. Om, G. H. Bae, and J. H. Lee, "Modeling of Vth shift in NAND flash-memory cell device considering crosstalk and short-channel effects," *IEEE Transactions on Electron Devices*, vol. 55, pp. 1020–1026, April 2008.

[46] H. An, K. Kim, S. Jung, H. Yang, K. Kim, and Y. Song, "The threshold voltage fluctuation of one memory cell for the scaling-down NOR flash," in *2nd IEEE International Conference on Network Infrastructure and Digital Content*, Sept 2010.

[47] B. Park, J. Song, E. Cho, S. Hong, J. Kim, Y. Choi, Y. Kim, S. Lee, C. Lee, D. Kang, D. Lee, B. Kim, Y. Choi, W. Lee, J. Choi, K. Suh, and T. Jung, "32nm 3-bit 32Gb NAND flash memory with DPT (double patterning technology) process for mass production," in *IEEE Symposium on VLSI Technology*, pp. 125–126, June 2010.

[48] E. Choi and S. Park, "Device considerations for high density and highly reliable 3D NAND flash cell in near future," in *IEEE International Electron Devices Meeting (IEDM)*, (San Francisco, CA), pp. 9.4.1 – 9.4.4, Dec 2012.

[49] H. Shim, S. Lee, B. Kim, N. Lee, D. Kim, H. Kim, B. Ahn, Y. Hwang, H. Lee, J. Kim, Y. Lee, H. Lee, J. Lee, S. Chang, J. Yang, S. Paark, S. Aritome, S. Lee, K. Ahn, G. Bae, and Y. Yang, "Highly reliable 26nm 64Gb MLC E2NAND (embedded-ECC and enhanced-efficiency) flash memory with MSP (memory signal processing) controller," in *IEEE Symposium on VLSI Technology*, pp. 216–217, June 2011.

[50] K. Takeuchi, T. Tanaka, and H. Nakamura, "A double-level-Vth select gate array architecture for multilevel NAND flash memories," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 602–609, Apr 1996.

[51] R. Huang, F. Zhou, Y. Li, Y. Cai, X. Shan, X. Zhang, and Y. Wang, "Novel silicon-based flash cell structures for low power and high density memory applications," in *2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*, pp. 709–712, Oct 2006.

[52] H. Nobukata, S. Takagi, K. Hiraga, T. Ohgishi, M. Miyashita, K. Kamimura, S. Hiramatsu, K. Sakai, T. Ishida, H. Arakawa, M. Itoh, I. Naiki, and M. Noda, "A 144 mb 8-level nand flash memory with optimized pulse width programming," in *1999 Symposium on VLSI Circuits. Digest of Papers (IEEE Cat. No.99CH36326)*, pp. 39–40, June 1999.

[53] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, (New York, NY, USA), pp. 14–23, ACM, 2009.

[54] J. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. Davis, P. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *IEEE International Conference on Microelectronic Systems Education (MSE)*, pp. 173–174, June 2007.

[55] "NanGate Library Optimization website." `http://www.nangate.com/`.

[56] "Synopsys website." `http://www.synopsys.com/`.

[57] "PTM website." `http://ptm.asu.edu/`.

[58] K. Zaitsu, K. Tatsumura, M. Matsumoto, M. Oda, S. Fujita, and S. Yasuda, "Flash-based nonvolatile programmable switch for low-power and high-speed FPGA by adjacent integration of MONOS/logic and novel programming scheme," in *VLSI Technology (VLSI-Technology): Digest of Technical Papers, 2014 Symposium on*, pp. 1–2, June 2014.

[59] "ITRS website." `http://www.itrs.net/`.

[60] S. Lee *et al.*, "A 128Gb 2b/cell NAND flash memory in 14nm technology with t_{PROG}=640us and 800MB/s I/O rate," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 138–139, Jan 2016.

[61] D. Lee *et al.*, "A 64Gb 533Mb/s DDR interface MLC NAND flash in sub-20nm technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 430–432, Feb 2012.

[62] K.-T. Park *et al.*, "A 7MB/s 64Gb 3-bit/cell DDR NAND flash memory in 20nm-node technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pp. 212–213, Feb 2011.

[63] J.-R. Hwang *et al.*, "20nm gate bulk-FinFET SONOS flash," in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pp. 154–157, Dec 2005.

[64] M. Abusultan and S. P. Khatri, "A ternary-valued, floating gate transistor-based circuit design approach," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016.

[65] "Cadence Design Systems website." `http://www.cadence.com/`.

[66] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Circuits and Systems, 1989., IEEE International Symposium on*, pp. 1929–1934 vol.3, May 1989.

[67] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design Test of Computers*, vol. 17, pp. 44–53, Jul 2000.

[68] "The EPFL Combinational Benchmark Suite Webpage." `http://lsi.epfl.ch/benchmarks`.

[69] M. Abusultan and S. Khatri, "Implementing low power digital circuits using flash devices," in *Computer Design (ICCD), 2016 34nd IEEE International Conference on*, October 2016.

[70] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," tech. rep., EECS Department, University of California, Berkeley, 1992.

[71] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, pp. 677–691, Aug. 1986.

[72] N. Saluja and S. P. Khatri, "A robust algorithm for approximate compatible observability don't care (CODC) computation," in *Design Automation Conference, 2004. Proceedings. 41st*, pp. 422–427, July 2004.

[73] A. Mishchenko and R. K. Brayton, "SAT-based complete don't-care computation for network optimization," in *Design, Automation and Test in Europe*, pp. 412–417 Vol. 1, March 2005.

[74] S. P. Khatri, S. Sinha, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SPFD-based wire removal in standard-cell and network-of-PLA circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, July 2004.

[75] M. Abusultan and S. P. Khatri, "A flash-based digital circuit design flow," in *International Conference On Computer Aided Design (ICCAD)*, November 2016.

[76] N. Eén and N. Sörensson, "An extensible SAT-solver," in *International conference on theory and applications of satisfiability testing*, pp. 502–518, Springer, 2003.

[77] "Xilinx website." `http://www.xilinx.com/`.

[78] H. Kojima, T. Ema, T. Anezaki, J. Ariyoshi, H. Ogawa, K. Yoshizawa, S. Mehta, S. Fong, S. Logie, R. Smoak, and D. Rutledge, "Embedded flash on 90nm logic

technology & beyond for FPGAs," in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, Dec 2007.

[79] S. Fong, J. Ariyoshi, and T. Ema, "Embedded flash on a low-power 65-nm logic technology," *Electron Device Letters, IEEE*, Sept 2012.

[80] K. Zaitsu *et al.*, "Nonvolatile programmable switch with adjacently integrated flash memory and CMOS logic for low-power and high-speed FPGA," *Electron Devices, IEEE Transactions on*, Dec 2015.

[81] J. Greene, S. Kaptanoglu, W. Feng, V. Hecht, J. Landry, F. Li, A. Krouglyanskiy, M. Morosan, and V. Pevzner, "A 65nm flash-based FPGA fabric optimized for low cost and power," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2011.

[82] "7 Series FPGAs Configuration User Guide (UG470)." Xilinx Corporation.

[83] J. Wang *et al.*, "Total ionizing dose effects on flash-based field programmable gate array," *Nuclear Science, IEEE Transactions on*, Dec 2004.

[84] P. Chow, S. O. Seo, J. Rose, K. Chung, G. Paez-Monzon, and I. Rahardja, "The design of a SRAM-based field-programmable gate array-Part II: Circuit design and layout," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, pp. 321–330, Sept. 1999.

[85] G. G. Lemieux and S. D. Brown, "A detailed routing algorithm for allocating wire segments in field-programmable gate arrays," in *Proc. Physical Design Workshop, Lake Arrowhead, CA*, 1993.

[86] Y.-W. Chang, D. F. Wong, and C. K. Wong, "Universal switch modules for FPGA design," *ACM Trans. Des. Autom. Electron. Syst.*, Jan. 1996.

[87] S. J. E. Wilton, *Architectures and Algorithms for Field-programmable Gate Arrays with Embedded Memory*. PhD thesis, 1997.

[88] "Application Note, Using SPI Flash with 7 Series FPGAs." Xilinx Corporation.

[89] "M69S SLC NAND flash memory datasheet." Micron Corporation.

[90] "IGLOO2 FPGA and SmartFusion2 SoC FPGA datasheet." Microsemi Corporation.

[91] K. W. Lee, S. K. Choi, S. J. Chung, H. L. Lee, S. M. Yi, B. I. Han, B. I. Lee, D. H. Lee, J. H. Seo, N. Y. Park, H. S. Kim, H. S. Kim, T. U. Youn, K. H. Noh, M. K. Lee, J. Y. Lee, K. H. Han, W. S. Woo, S. W. Cho, S. C. Lee, S. S. Kim, C. S. Hyun, W. J. Suh, S. D. Kim, M. K. Ahn, H. S. Kim, K. S. Kim, G. S. Cho, S. K. Park, S. Aritome, J. W. Kim, S. K. Lee, S. J. Hong, and S. W. Park, "A highly manufacturable integration technology of 20nm generation 64Gb multi-level NAND flash memory," in *2011 Symposium on VLSI Technology - Digest of Technical Papers*, pp. 70–71, June 2011.

[92] Y. Koh, "NAND flash scaling beyond 20nm," in *2009 IEEE International Memory Workshop*, pp. 1–3, May 2009.

[93] S. H. Chen, H. T. Lue, Y. H. Shih, C. F. Chen, T. H. Hsu, Y. R. Chen, Y. H. Hsiao, S. C. Huang, K. P. Chang, C. C. Hsieh, G. R. Lee, A. T. H. Chuang, C. W. Hu, C. J. Chiu, L. Y. Lin, H. J. Lee, F. N. Tsai, C. C. Yang, T. Yang, and C. Y. Lu, "A highly scalable 8-layer vertical gate 3d nand with split-page bit line layout and efficient binary-sum milc (minimal incremental layer cost) staircase contacts," in *2012 International Electron Devices Meeting*, pp. 2.3.1–2.3.4, Dec 2012.

[94] K. Parat and C. Dennison, "A floating gate based 3D NAND technology with CMOS under array," in *2015 IEEE International Electron Devices Meeting (IEDM)*, pp. 3.3.1–3.3.4, Dec 2015.

[95] T. Y. Chen, Y. H. Chang, C. C. Ho, and S. H. Chen, "Enabling sub-blocks erase management to boost the performance of 3D NAND flash memory," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2016.

[96] S. C. Lai, H. T. Lue, T. H. Hsu, C. J. Wu, L. Y. Liang, P. Y. Du, C. J. Chiu, and C. Y. Lu, "A bottom-source single-gate vertical channel (BS-SGVC) 3D NAND flash architecture and studies of bottom source engineering," in *2016 IEEE 8th International Memory Workshop (IMW)*, pp. 1–4, May 2016.

[97] Y. A. Chung, Z. Yang, Y. C. Chiu, S. P. Hong, H. J. Lee, N. T. Lian, T. Yang, K. C. Chen, and C. Y. Lu, "Novel hybrid 3D NAND flash memory containing vertical-

gate and gate-all-around structures," in *2016 27th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pp. 371–374, May 2016.

[98] Y. H. Hsiao, H. T. Lue, W. C. Chen, B. Y. Tsui, K. Y. Hsieh, and C. Y. Lu, "Ultra-high bit density 3D NAND flash-featuring-assisted gate operation," *IEEE Electron Device Letters*, vol. 36, pp. 1015–1017, Oct 2015.

[99] D. Kang, W. Jeong, C. Kim, D. H. Kim, Y. S. Cho, K. T. Kang, J. Ryu, K. M. Kang, S. Lee, W. Kim, H. Lee, J. Yu, N. Choi, D. S. Jang, C. A. Lee, Y. S. Min, M. S. Kim, A. S. Park, J. I. Son, I. M. Kim, P. Kwak, B. K. Jung, D. S. Lee, H. Kim, J. D. Ihm, D. S. Byeon, J. Y. Lee, K. T. Park, and K. H. Kyung, "256 gb 3 b/cell v-nand flash memory with 48 stacked wl layers," *IEEE Journal of Solid-State Circuits*, vol. PP, no. 99, pp. 1–8, 2016.

[100] Y. Park and J. Lee, "Device considerations of planar NAND flash memory for extending towards sub-20nm regime," in *2013 5th IEEE International Memory Workshop*, pp. 1–4, May 2013.