Neuron-based Digital and Mixed-signal Circuit Design:

From ASIC to SIMD Processors

by

Ankit Wagle

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved February 2023 by the
Graduate Supervisory Committee:

Sarma Vrudhula, Chair
Sunil Khatri
Aviral Shrivastava
Jae-Sun Seo
Fengbo Ren

ARIZONA STATE UNIVERSITY

May 2023

ABSTRACT

Among the many challenges facing circuit designers in deep sub-micron technologies, power, performance, area (PPA) and process variations are perhaps the most critical. Since existing strategies for reducing power and boosting the performance of the circuit designs have already matured to saturation, it is necessary to explore alternate unconventional strategies. This investigation focuses on using perceptrons to enhance PPA in digital circuits and starts by constructing the perceptron using a combination of complementary metal-oxide-semiconductor (CMOS) and flash technology. The use of flash enables the perceptron to have a variable delay, and functionality, making them robust to process, voltage, and temperature variations. By replacing parts of an application-specific integrated circuit (ASIC) with these perceptrons, improvements of up to 30% in the area and 20% in power can be achieved without affecting performance. Furthermore, the ability to vary the delay of a perceptron enables circuit designers to fix setup and hold-time violations post-fabrication, while reprogramming the functionality enables the obfuscation of the circuits. The study extends to field-programmable gate arrays (FPGAs), showing that traditional FPGA architectures can also achieve improved PPA by replacing some Look-Up-Tables (LUTs) with perceptrons. Considering that replacing parts of traditional digital circuits provides significant improvements in PPA, a natural extension was to see whether circuits built dedicatedly using perceptrons as its compute unit would lead to improvements in energy efficiency. This was demonstrated by developing perceptron-based compute elements and constructing an architecture using these elements for Quantized Neural Network acceleration. The resulting circuit delivered up to 50 times more energy efficiency compared to a CMOS-based accelerator without using standard low-power techniques such as voltage scaling and approximate computing.

DEDICATION

To my parents, my little sister, and my precious little cat

for enriching my life and inspiring me to reach for the stars.

ACKNOWLEDGMENTS

Writing a dissertation is a long and challenging journey, and I am grateful to have had the support of so many people along the way.

I want to thank my advisor, Prof. Sarma Vrudhula, for being an exceptional mentor. His expertise, guidance, and encouragement have been crucial in shaping this dissertation. Prof. Vrudhula taught me that writing is nature's way of correcting bad thinking, while math is nature's way of correcting bad writing. He also emphasized the importance of using a mathematical framework to transform ideas into innovations, leveraging existing optimization techniques. I look forward to continuing to learn from him in the years to come.

I was fortunate to collaborate with Prof. Sunil Khatri during my Ph.D. His initial discussion with Prof. Vrudhula about integrating flash transistors in threshold gates significantly boosted the work presented in this dissertation and led to further innovations. Prof. Khatri taught me not only valuable academic skills, but also the importance of working hard and enjoying life to the fullest.

I would like to extend special thanks to Jinghua Yang, for her utmost support in the formative years of my Ph.D. She taught me the discipline I needed as a student in the Ph.D. program. I will forever be grateful for her support.

I owe a great debt of gratitude to Gian Singh, whose dedication and contributions to the development of flash-based threshold logic have been of utmost importance.

I am also grateful to my colleagues Niranjan Kulkarni, Elham Azari, Mehdi Ghasemirahagi, Soroush Heidari, and Shail Dave, who have provided a supportive and encouraging environment throughout my academic journey.

A huge thanks to Lisa for enabling me to handle conferences, grants, events, etc., among many other things that not only allowed me to showcase my research but also have fun doing it. I will always treasure these memories.

I am deeply grateful for the resources provided by Alphacore, which have been instrumental in allowing me to conduct the research I set out to accomplish. A special thanks to Phanindra Kumar Bikkina and Esko Mikkola.

I am grateful to my colleagues at Maxlinear, specifically Parthasarathy Gopal and Anitha Yella, for introducing me to real-world challenges and teaching me how to tackle them. Sarathy's outstanding knowledge and clear communication style continue to inspire me, and learning from him was vital for my Ph.D. program. Meanwhile, Anitha taught me the importance of perseverance in the face of challenges, having overcome struggles of her own, and helped me prepare for similar obstacles.

I want to extend a heartfelt thank you to my committee members, Prof. Jae-Sun Seo, Prof. Aviral Shrivastava, and Prof. Fengbo Ren. Prof. Seo consistently ensured I had the resources necessary for my research and understood its requirements. Prof. Shrivastava expanded my knowledge of computer architecture, particularly data reuse. Prof. Ren played a key role in my research, as he motivated us to focus on BNN accelerator design in his class and provided valuable feedback.

I would like to thank sponsoring agencies National Science Foundation (NSF) I/UCRC Center for Embedded Systems from NSF grant # 1361926 and other NSF grants #1701241, #2008244 for supporting this work.

I would also like to thank the faculty and staff of the School of Computing and Augmented Intelligence (SCAI) for the guidance, resources, and opportunities.

I want to give a big shoutout to my uncle, Prof. Datta Gaitonde, whose extensive experience as a professor and mentor has provided me with a well-rounded understanding of the program's expectations, requirements, and goals.

Finally, I would like to express my gratitude to my parents and my sister, who have supported and encouraged me throughout my life. I will always be grateful for their unwavering support.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

Table Page

Table Page

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

For the past four decades, the overwhelming majority of digital microelectronics circuits have been designed using Complementary Metal Oxide Semiconductor (CMOS) technology. During that period, CMOS technology has undergone over 20 process generations, with each new generation doubling the transistor density and increasing the performance by 40% every generation. Around two decades ago, trends showed that the exponential increase in power consumption would halt further increases in transistor density and performance. The feared rise in power consumption was largely brought under control by various techniques that included device design and fabrication, circuit architecture, dynamic voltage, and frequency control, the transition to multi-core processors, and the development of energy-aware algorithms. A visible demonstration of decades of advances is the recent (2022) announcement of the Apple M2 chip with $20B$ transistorsapp (2020). This is over a 700,000X increase in transistor density over the first commercial microprocessor – the Intel 4004 released in 1971, which had 2,300 transistors on a 12 mm^2 die. The extraordinary advances in digital microelectronics have not just been due to technological strides, but substantial innovation in design methodology and automation also had to take place. This includes the development of highly scalable optimization algorithms for performance, power, and area (PPA) applicable to both logic and physical domains, the standardization of circuit and logic primitives, and the implementation of the algorithms in software tools.

Figure 1.1 shows the many techniques developed over the past several decades to enhance the performance, power, and area of digital systems. They include advances

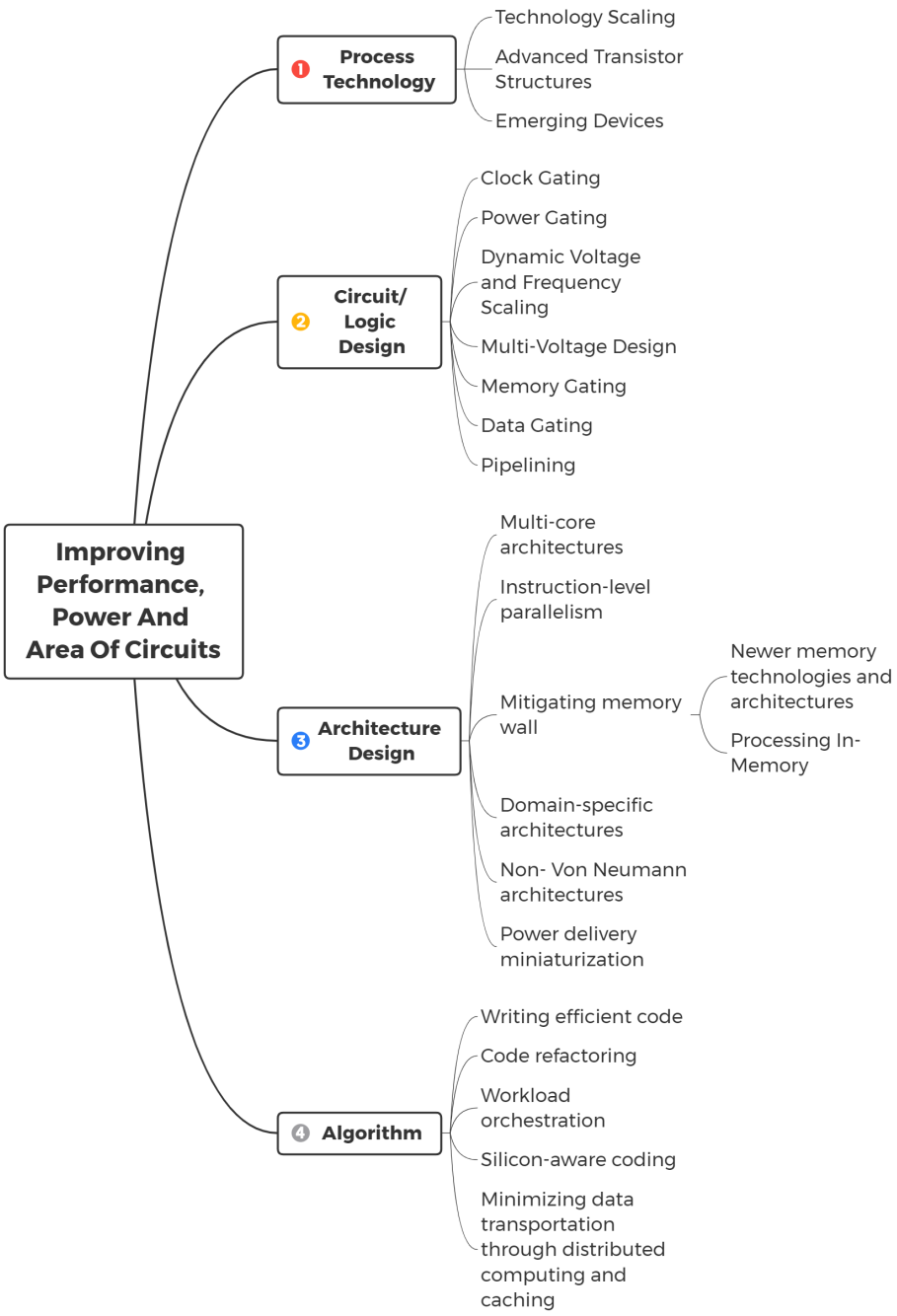


Figure 1.1: State of the Art for Improving Performance, Power, and Area (PPA) of Circuits

in (1) process technology device design and fabrication), (2) circuit and logic design, (3) architectural level design, and (4) algorithm design. Each of these techniques is

discussed in further detail below:

1.0.1 Process Technology

Technology scaling refers to improving or reducing the size of transistors, while maintaining or improving their performance. This results in an increase in the number of transistors that can fit onto a single IC. This leads to more compact and efficient circuits, as well as an increase in performance, such as higher clock speeds, greater storage capacity, and improved power efficiency. This is achieved through the development of new manufacturing processes, such as fully depleted silicon-on-insulator FETs (FDSOI) and fin-based FETs (FinFETs), which are more energy-efficient and offer improved performance compared to basic planar transistor structures (FETs). Despite the progress in silicon-based circuits, technology scaling faces many challenges, including a lack of robustness and the large costs of manufacturing ICs. As a result, researchers are also exploring new devices for specific purposes and applications as they offer unique features and properties suited for those applications without compromising on the robustness and cost constraints set by those applications. These emerging devices include PRAM (Phase-Change-based device), ReRAM (Filament Type device), and MRAM (Spintronic device), among many others. Often, emerging devices can also contribute to the demand for computing power by being applied in extreme environments and conditions that silicon cannot, or provide higher density and better performance with low power. However, they are still far from being commercially viable for mass manufacturing. Popular techniques related to process-technology that are used to improve energy efficiency are summarized in Figure 1.2.

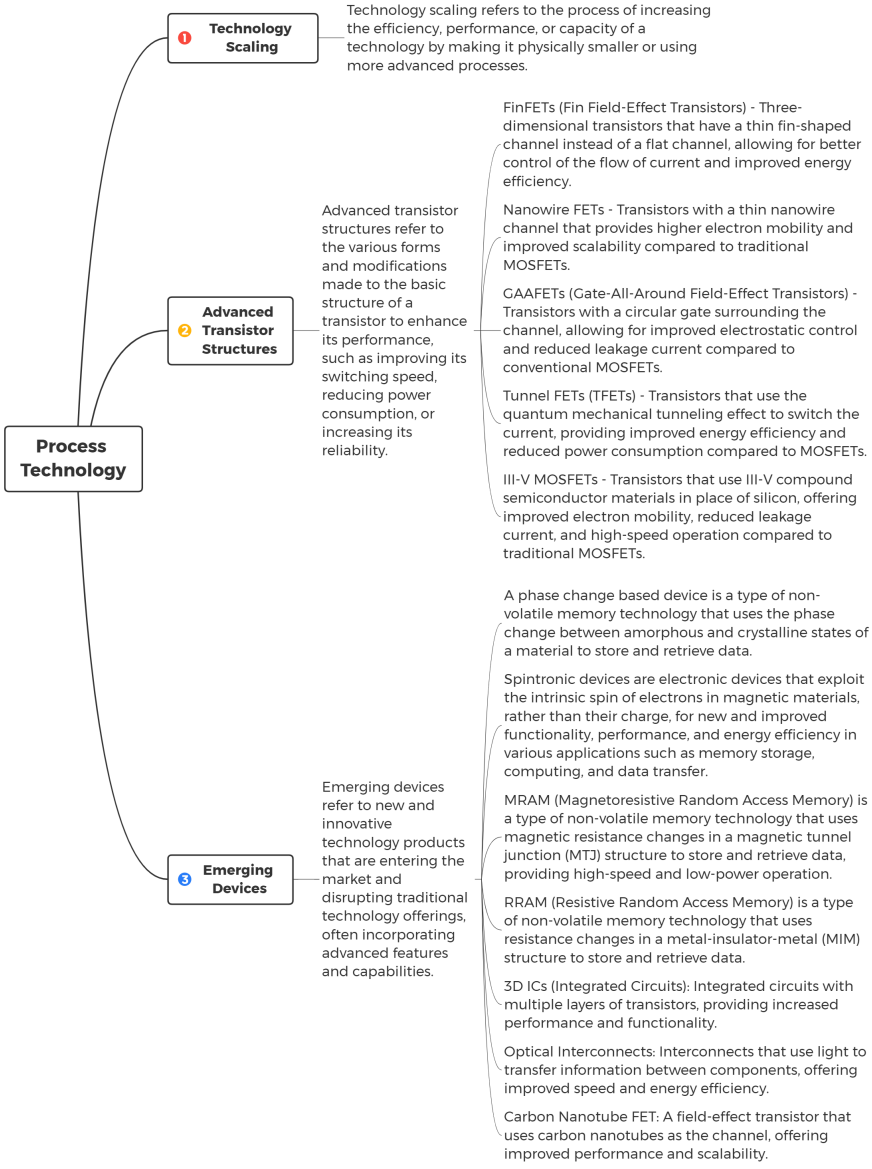


Figure 1.2: Process Technology: Advancements in Semiconductor Manufacturing Processes That Allow for the Creation of More Energy-Efficient Electronic Components, Such as Transistors, with Improved Performance, Reduced Power Consumption, and Increased Reliability.

1.0.2 Circuit/ Logic Design

Circuit/logic design techniques aim to reduce the power consumption of an integrated circuit (IC) without affecting its operation. Some techniques include clock

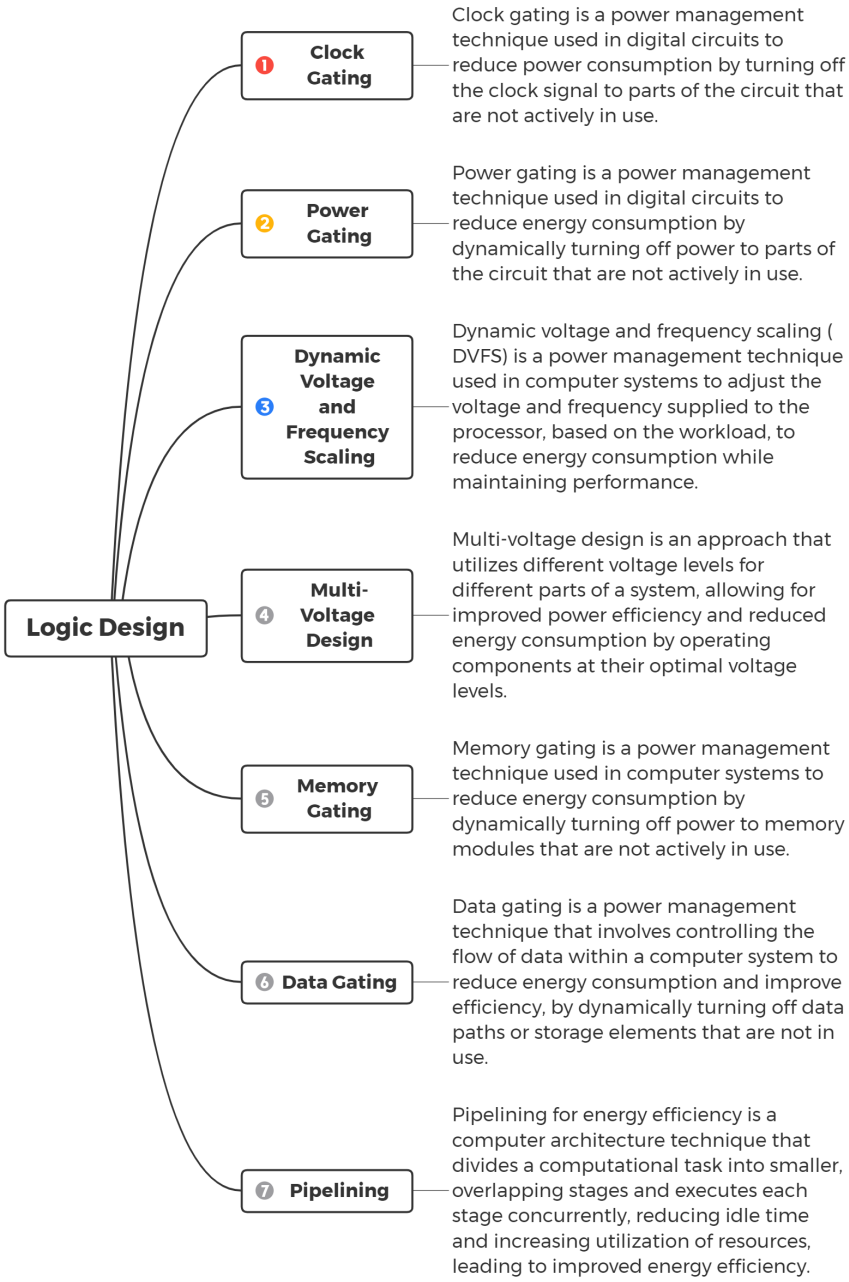


Figure 1.3: Logic Design: Approaches Such as Clock Gating, Power Gating, Etc. with the Aim of Minimizing Power Consumption While Maintaining the Performance and Functionality of the circuit.

gating, power gating, multiple voltage design, dynamic voltage and frequency scaling, memory gating, data gating, and pipelining, as shown in Figure 1.3. These techniques help control power consumption by turning off or scaling power to unused parts of the circuit, partitioning the logic into power domains with different voltage lines, adjusting voltage and frequency at run-time, enabling only used memory, blocking unused data, and adding extra register stages in the data path to improve timing.

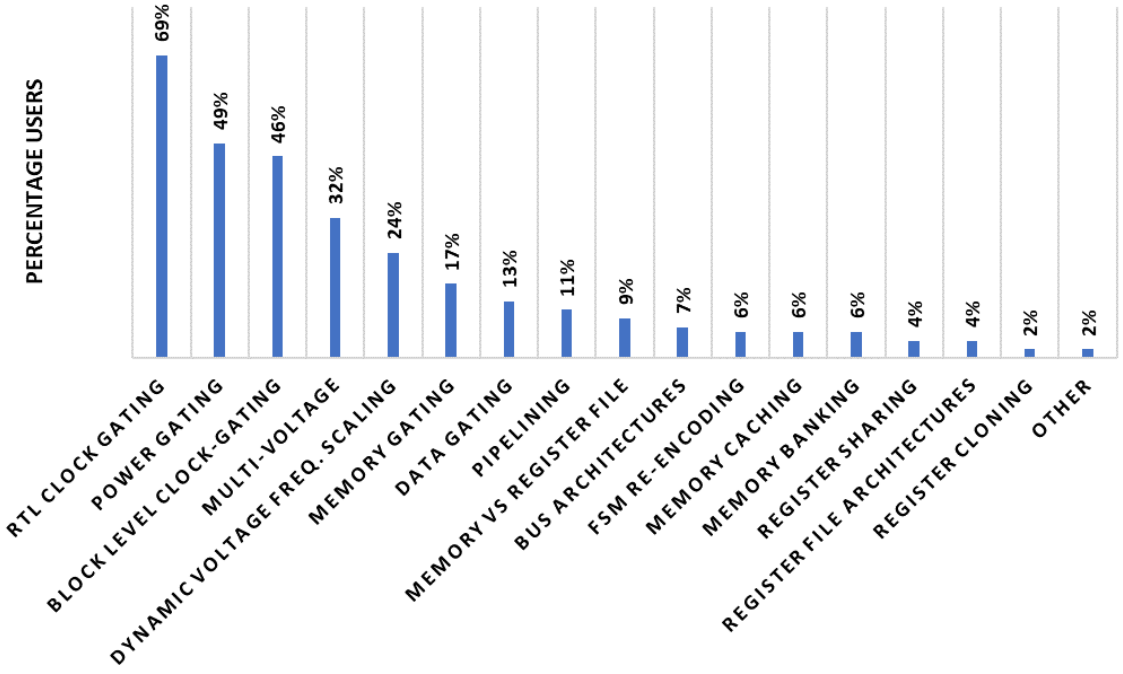


Figure 1.4: Conventional Hardware-Level Low Power Techniques Survey. Values Indicate the Percentage of Users Using the Associated technique.

Based on a recent survey from 446 VLSI design engineers, Figure 1.4 shows the popular circuit-level low-power techniques used in the industry for ASICs and FPGAs.

While clock gating, power gating, and DVFS are predominantly popular and are used by most engineers, several low-power techniques such as memory caching, memory banking, register sharing, register file architecture, register cloning, etc., are not used as often, primarily due to one or more of the following reasons:

1. They are harder to implement by anyone other than the engineer coding the RTL (unlike techniques such as clock-gating)

2. Techniques such as register sharing, memory caching, etc., that worked in one node may not work as effectively in the next node.

3. It takes lots of different tool runs, such as RTL synthesis, simulation, power analysis, etc., to evaluate these techniques.

4. They are often not compatible with industry-standard design flows and techniques.

5. They compromise the robustness of the circuit.

For the classification discussed above, a clear conclusion is that the ease of integration of a low-power technique directly determines how often designers would use it. The work presented in this dissertation is designed to seamlessly integrate with industry-standard design flows and provide improvements on top of the conventional low-power techniques that already exist in the industry.

1.0.3 Architectural Design

Architectural design techniques aim to optimize the energy efficiency of integrated circuits (ICs) by constructing and integrating hardware blocks. Some of the notable techniques Vasilakis *et al.* (2017); Corporation (2019); Palacharla *et al.* (1997); Henry *et al.* (1999); Sohi *et al.* (1995); Fisher (1981); Tomasulo (1967); Lee *et al.* (2010); Raoux *et al.* (2008); Wong *et al.* (2010); Research (2020); Chen *et al.* (2010); Bhattacharjee *et al.* (2017); Mehonic *et al.* (2020); Wong *et al.* (2012); Ha (2018); Ha *et al.* (2016); Liu *et al.* (2012); Chang *et al.* (2017); Mutlu *et al.* (2020); Li and Luo (2016); Tech (2018b,a); Jouppi *et al.* (2017); Chen *et al.* (2016); Gao *et al.* (2017);

Farabet *et al.* (2011); Li *et al.* (2018); Akopyan *et al.* (2015); Furber (2014); Davies *et al.* (2018); Haj-Yahya *et al.* (2019); Lee (2016), shown in Figure 1.5 include the use of multi-core architectures, instruction-level parallelism, mitigating the memory wall using advanced memory-based techniques, construction of domain-specific architectures, dataflow (non-Von Neumann) architectures, and power delivery miniaturization and use of reconfigurable power delivery networks. Most of these techniques require architecture exploration based on the application being supported.

1.0.4 Algorithm Selection

To reduce power consumption at the algorithmic level, several techniques (shown in Figure 1.6) are used, such as writing efficient code, code refactoringMenshawy *et al.* (2021), workload orchestration, silicon-aware coding, and minimizing data transportation through distributed computing and caching. Writing efficient code involves designing algorithms that use fewer instructions or require fewer memory accesses, leading to reduced energy consumption. Code refactoring involves restructuring existing code to enhance its energy efficiency. Workload orchestration involves moving cloud workloads based on renewable energy sources, and offloading computation from mobile devices to servers. Silicon-aware coding exploits the energy management capabilities of modern chipsets to reduce energy use. Minimizing data transportation through distributed computing and caching involves caching content locally to minimize buffering and reduce the energy used for transport.

The techniques identified in Figure 1.1 seem to be exhaustive, and there do not seem to be any fundamentally new approaches to improving the PPA of CMOS digital system. This dissertation presents another method for improving the PPA of CMOS digital circuits that is unlike any of the techniques shown in Figure 1.1.

This dissertation aims to show that significant improvements in the performance,

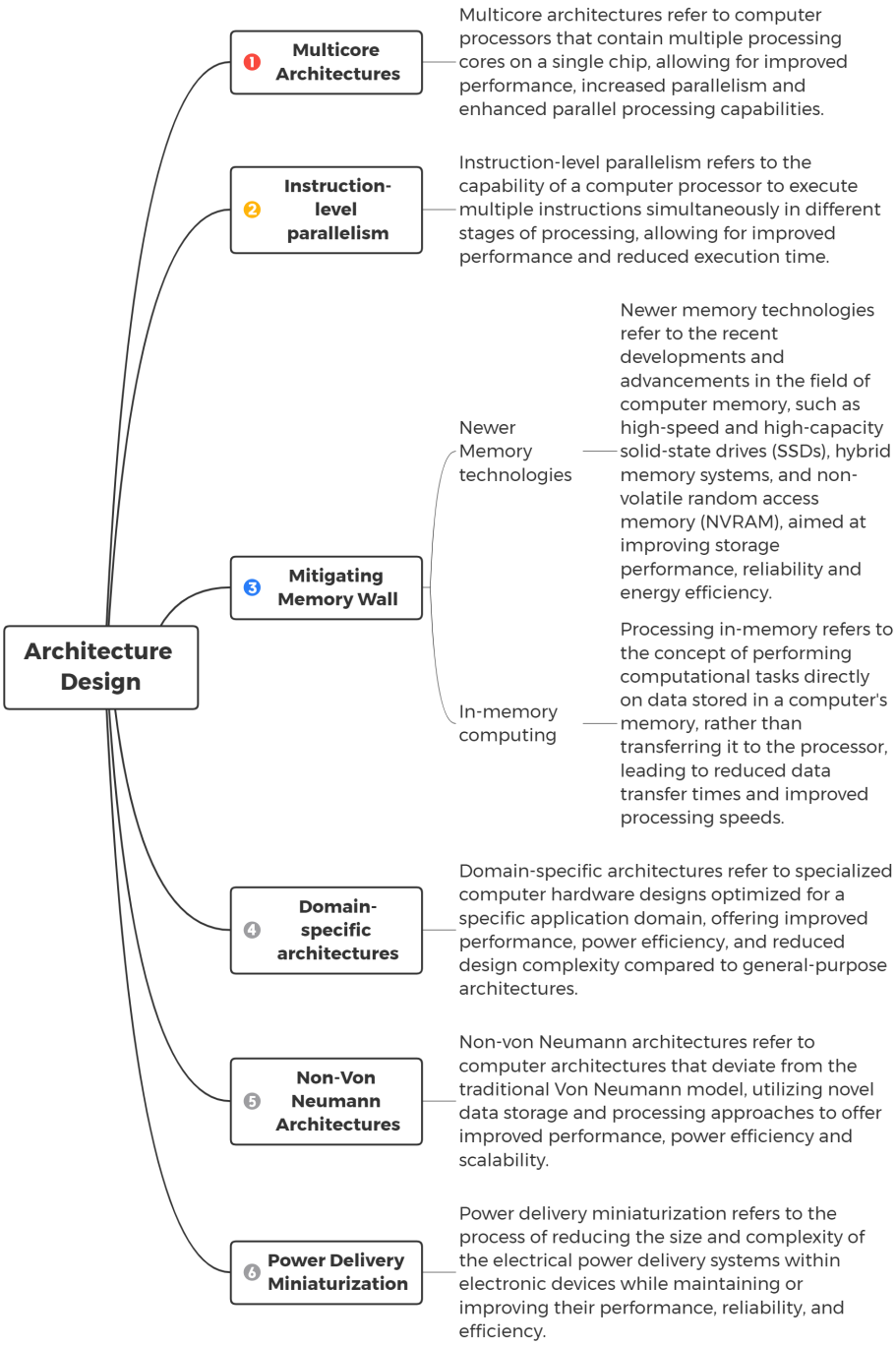


Figure 1.5: Architectural Design: Design Approaches for Computer Systems and Components That Aim to Reduce Energy Consumption and Improve Energy efficiency.

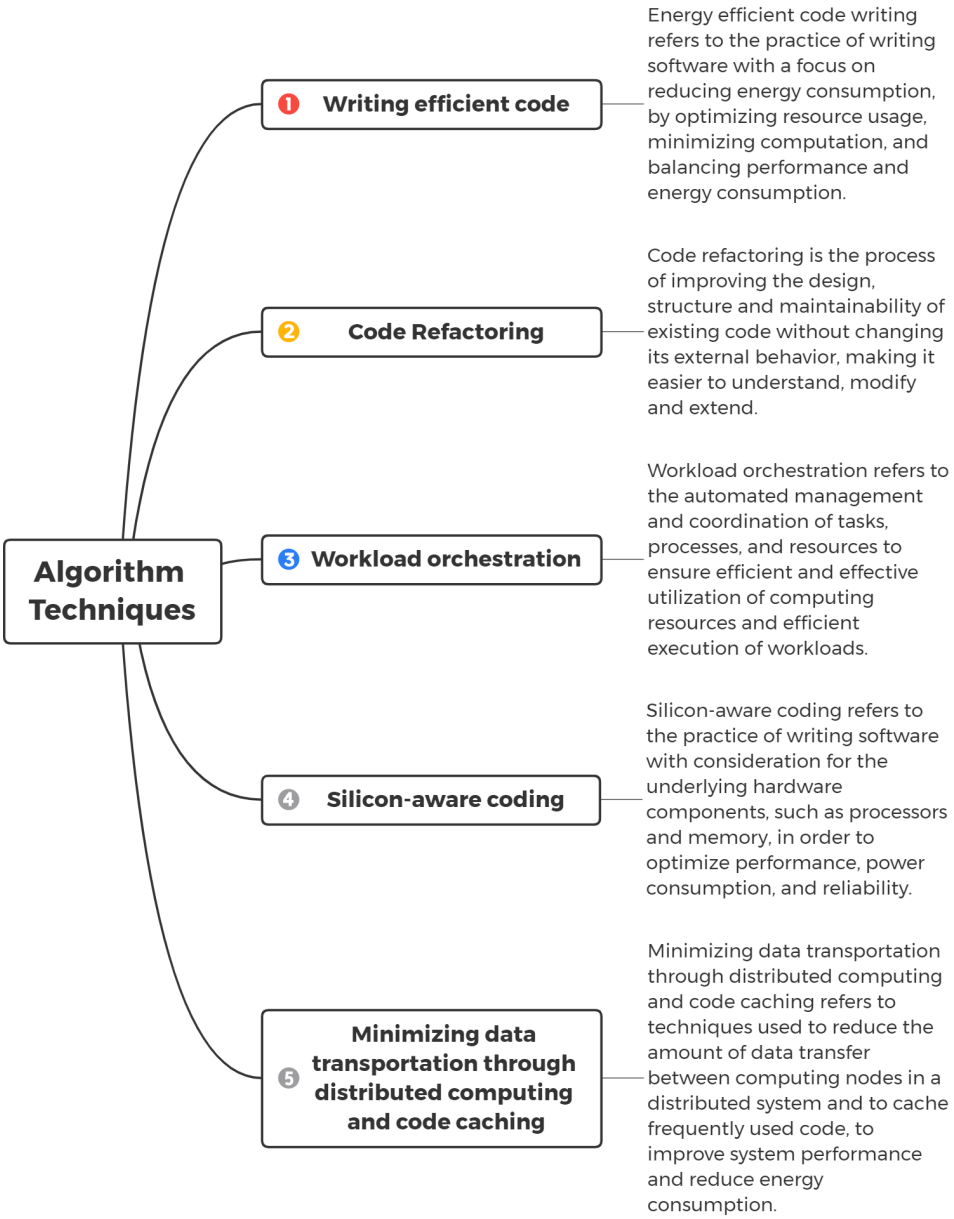


Figure 1.6: Algorithm Selection: Methods Used in Software Development That Aim to Reduce the Energy Consumption of a System, Such as Optimizing the Use of Resources, Reducing Computational Complexity, and Balancing Performance and Energy consumption.

power, and area of digital VLSI circuits can still be achieved by integrating unconventional but CMOS-compatible device technologies and unconventional circuit and system architectures with conventional CMOS circuits and design methodologies. Specifically, the unconventional approach being explored is the design of a perceptron, a digital-analog-digital circuit, as a conventional standard cell using programmable conductance devices such as flash transistors.

1.1 Research Contribution and Dissertation Outline

The research presented in this dissertation builds upon existing work by Niranjan et al. Kulkarni *et al.* (2016b) on integrating perceptrons in ASICs for improving the performance, power, area (PPA), and energy efficiency of digital circuits. The associated contributions are explained later in this section. The work in this dissertation is divided into three main parts, each of which focuses on a specific aspect of enhancing PPA and energy efficiency in digital circuits.

1.1.1 Design of Binary Perceptrons Using CMOS and Flash Transistors and Its Integration into ASICs

This part of the dissertation demonstrates the benefits of using of perceptrons (binary neurons) as logic primitives, to be integrated into existing ASIC designs, as well as reconfigurable architectures for PPA improvements. Similar work has already been explored by several other researchers, including Niranjan et al. Kulkarni *et al.* (2016a); Yang *et al.* (2015) who demonstrated significant improvements in the area and power of an IC through post-fabrication results. However, almost all the prior perceptron designsBeiu (2003); Celinski *et al.* (2003); Kulkarni *et al.* (2016a); Mozaffari *et al.* (2018) exclusively used CMOS technology, which limited the potential utility of those perceptrons.

This work improves upon the existing perceptron designs by using a combination of CMOS and flash transistors. The introduction of flash transistors to these designs notably improves the number of functions that the perceptron can implement (28 to 117 Boolean functions) Muroga (1971b). Furthermore, since the threshold voltages of the flash transistors can be changed post-fabrication, this property further enables the perceptrons to not only have variable delays and functionality post-fabrication, but also be significantly more robust to process, voltage and temperature (PVT) variations.

The perceptron discussed in this work offers several opportunities for improved circuit design. Similar to prior work, this perceptron can be used to improve the PPA of an ASIC. However, due to its delay flexibility, it is now possible to also fix setup and hold time violations post-fabrication using these perceptrons. It is also possible to obfuscate circuits from the foundry, as functions can only be assigned to the perceptrons post-fabrication using a special programming method. Since the flash transistors are reprogrammable, they can also be used to revert the effects of aging on the perceptrons.

1.1.2 Embedding Binary Perceptrons in an FPGA Architecture

FPGAs are one of the most popular platforms when it comes to applications that require reconfigurable architectures, and are often used for machine learning and IoT applications. The traditional FPGAs are constructed using lookup tables (LUTs), that can be reconfigured to implement various functions. Most of the research in the improvement of energy efficiency of the FPGA architectures deal with circuit-level techniques Ahmed and Rose (2004); Feng *et al.* (2018); Anderson *et al.* (2012); Goncalves *et al.* (2013); Nukala *et al.* (2012); Sampath *et al.* (2015) such as sweeping the size of LUTs, use of emerging devices, etc., and algorithm-level techniques Pan

and Liu (1996); Li *et al.* (2001); Lin *et al.* (2006); Kao and Lai (1999); Lipatov and Tiunov (2017) such as mapping and routing of logic to the LUTs. While a lot of work has been done on the algorithm-level, relatively much less work has been done on the circuit-level techniques. This part of the dissertation contributes to circuit-level techniques by introducing a new architecture, referred to as threshold logic FPGA (TLFPGA). This architecture is constructed by replacing a few of the LUTs from an FPGA architecture with perceptrons. Such integration of perceptrons is radically different from existing approaches that improve the PPA of an FPGA. We show that by using the TLFPGA architecture, especially for deeply pipelined circuits, it is possible to gain significant area, power, and performance improvements over the classic FPGA architecture, by strategically mapping the circuits to the perceptrons. Furthermore, the presented work is compatible with the conventional FPGA flow, and is compatible with all the innovations that were done at the algorithm-level. While the evaluation for this work was only done using 28nm technology node, future work has been planned in 40nm for comparison across nodes.

1.1.3 Hardware Acceleration of BNNs and QNNs Using Perceptrons

The acceleration of neural networks (NN) is one of the most important applications in the field of IC design today Hinton *et al.* (2012); Krizhevsky *et al.* (2012); Ren *et al.* (2015); Hunt *et al.* (1992); Liang and Hu (2015); He *et al.* (2015). The chips built for this application need to handle data-and-compute-intensive workloadsSimonyan and Zisserman (2014); Nurvitadhi *et al.* (2017). Some of these ICs are expected to deliver real-time performance using a battery as their energy source for portable applications. Researchers in the field of accelerator-design use several circuit-level techniques such as approximate computing Zhang *et al.* (2015), voltage scaling, frequency scaling, etc., and use data-reuse strategies such as loop unrolling Ma *et al.* (2018), cache memory

managementPisarchyk and Lee (2020), etc. to improve the PPA of these accelerators. The circuit-level techniques generally come at the cost of either performance, area, or accuracy of the neural network being inferred. Meanwhile, the data-reuse strategies Park *et al.* (2016) are already quite mature, and can be exhaustively explored through automated tools.

Quantized Neural Networks (QNNs)Hubara *et al.* (2017) are a type of neural network where the weights and activations are represented using a limited number of discrete values instead of continuous values. They were developed as a way of reducing memory usage and computation time slightly trading off accuracy, making it possible to deploy deep learning models on devices with limited computational resources. QNNs are represented as a directed acyclic graph (DAG), where the nodes represent quantized operations and the edges represent the flow of quantized values. The operations performed at each node can be simple arithmetic operations like addition and multiplication, but with quantized values. The output of one node serves as the input to the next node in the graph, and the final output is the prediction of the network.

Hardware accelerators used to execute QNNs use MAC units (described later in Chapter 5) to calculate the weighted sum of inputs. For all the other operations needed to execute QNNs, separate dedicated hardware blocks are needed. As a result, these accelerators face two major issues. The first issue is that a MAC unit always performs operations of maximum bit-width regardless of the bit-width of its operands. The second issue is that the use of dedicated blocks for less frequently used operations wastes area and consumes leakage power when these blocks are kept dormant.

This part of the dissertation presents an alternate hardware accelerator for QNNs called TULIP. It consists of special processing elements called TULIP-PEs constructed using a combination of perceptrons and local registers. TULIP-PEs can be used to

execute compute graphs in which each node is either an addition, comparison, or logic operation. These processing elements solve the previously mentioned issues as follows:

1. TULIP-PEs can be reconfigured to execute operations of custom bit-width that match the size of its operands, as opposed to always performing max bit-width operations as is done in MAC units.

2. Since TULIP-PEs can compute any compute graph that can be decomposed into addition, comparison, and logic operations, all the operations needed for QNNs can be evaluated on the same hardware, thereby eliminating the need for adding dedicated hardware for less frequently used operations.

TULIP-PEs, when used to construct a QNN accelerator, also enhance the reuse of data fetched from memory. This is because a TULIP-PE is much smaller than a MAC unit. As a result, if several TULIP-PEs are used in the same area as a MAC unit, they can simultaneously initiate multiple operations that share the fetched input data.

Chapter 2

BACKGROUND

A *threshold function* is a type of Boolean function that is used to determine whether the weighted sum of its inputs exceeds a certain threshold value. It is implemented by a perceptron, which is a type of hardware circuit that can perform this computation. This chapter introduces threshold functions and provides an overview of various perceptron architectures that have been developed in the past to implement them.

2.1 Threshold Functions and Threshold Gates

2.1.1 Threshold Functions

Threshold functions are an interesting and valuable subset of Boolean functions. They were first proposed in 1943 as simple models of neurons McCulloch and Pitts (1988), which generated substantial work on neural networks – a subject that has been revived recently with the emergence of machine learning. The use of threshold logic in digital design and synthesis was extensively investigated in the 1960s and 1970s, culminating in two authoritative works Mullin (1966); Muroga (1971c).

A threshold logic function $f(x_1, \cdots, x_n)$ Muroga (1971a) is a unate Boolean function whose on-set and off-set are *linearly separable*, i.e., there exists a vector of weights $\boldsymbol{W} = (w_1, w_2, \cdots, w_n)$[1] and a threshold T such that

[1]W.L.O.G, weights can be assumed to be positive integers Siu *et al.* (1995), and for a given truth table of a threshold function, there is a weight vector whose sum is minimum Siu *et al.* (1995).

$$f(x_1, x_2, \cdots, x_n) = 1 \Leftrightarrow \sum_{i=1}^{n} w_i x_i \geq T, \tag{2.1}$$

where $\sum$ here denotes the arithmetic sum. A threshold function can be equivalently represented by $(\boldsymbol{W}, T) = (w_1, w_2, \cdots, w_n; T)$.

2.1.2 Perceptron Learning Algorithm (Pla)

The Perceptron Learning Algorithm (PLA) was developed by Frank Rosenblatt in the 1950s and is considered to be the first artificial neural network algorithm. It is used to determine the weights corresponding to a given threshold function. PLA starts with an initial weight vector $\mathbf{W}$ and threshold T, and selects an ON-set or an OFF-set minterm x incorrectly classified by $\mathbf{W}$. It then adjusts $\mathbf{W}$ and T to classify x correctly. The pseudo-code of PLA is shown in Algorithm 1. The remarkable property of this algorithm is that it is guaranteed to converge if the given function is a threshold function.

Why PLA works: The update step in PLA (Lines 7,9) is the critical part of the Algorithm 1. The **if**-statement, on line 7, determines if $[\mathbf{W}, T]$ correctly classifies the input minterm x. When the decision boundary correctly classifies a minterm, nothing updates. However, when misclassification occurs, the decision boundary is updated in the "right direction" to prevent misclassification from occurring again. This addition moves the boundary in the right direction because the new $[\mathbf{W}, T]$ better classifies the input minterm x.

2.1.3 Determining Threshold Function Weights with Ilp

Another technique to determine the weights of a threshold function is to use Integer linear programming (ILP), which entails formulating the problem as an optimization problem with binary variables. The objective function is to minimize the

Algorithm 1 Perceptron Learning Algorithm

Input: $P \leftarrow$ **On-terms,** $N \leftarrow$ **Off-terms**

Output: [W,T]

$\mathbf{W} = [0, 0, ...0]$

$T = 1$

convergence = FALSE

while !convergence **do**

  Pick a random minterm x

  if $x \in P$ and $x \cdot \mathbf{W} < T$ **then**

    $\mathbf{W} = \mathbf{W} + x, T = T - 1$

  else if $x \in N$ and $x \cdot \mathbf{W} \geq T$ **then**

    $\mathbf{W} = \mathbf{W} - x, T = T + 1$

  else if All minterms produced expected output with [**W**,T] **then**

    convergence = $TRUE$

  end if

end while

sum of weights and threshold, such that all the minterms of a given truth table are correctly classified. The binary variables can be used to model the threshold, such that if the weighted sum of the inputs exceeds the threshold, the output is 1, and if it is less than the threshold, the output is 0. The ILP problem can then be solved to find the optimal weights that satisfy the constraints and optimize the objective function.

The advantages of using integer linear programming (ILP) over the PLA include:

- **Global optimality:** ILP can guarantee finding the global optimal solution, while the PLA only converges to a feasible solution.
- **Interpretability:** ILP solutions are more interpretable, as the weights and

thresholds can have direct physical meaning, whereas the weights and thresholds produced by the perceptron algorithm may not have a straightforward interpretation. For instance, for an OR function, an ILP may generate the weights to be $[1, 1; 1]$, whereas PLA may generate $[5, 3; 1]$. Although both sets of weight-threshold representations indicate an OR function, weight set $[1, 1; 1]$ is more interpretable because it shows that the weightage of both inputs in this OR function is the same.

- **Scalability:** ILP can be computationally expensive for large-scale problems, while the perceptron algorithm is computationally efficient and scalable.

It is worth noting that the choice between ILP and the perceptron algorithm depends on the specific problem and requirements. In some cases, the perceptron algorithm may be sufficient and faster, while in others, ILP may be necessary to handle complex constraints or to guarantee global optimality.

2.1.4 Standard Operations Using Threshold Functions

Many Boolean functions that commonly occur in logic circuits, particularly arithmetic operations, are threshold functions. Examples include parity, comparison, addition, multiplication, division, modulo, counter, exponent, and many more. This subsection will discuss how these functions can be constructed using threshold logic, and what the gate and depth complexity of each implementation is.

Let C be a directed acyclic graph where the nodes represent the threshold functions of their inputs, and the edges represent the flow of data between the nodes. The variables s and d represent the number of nodes and the maximum node depth in C, respectively. Using this notation, Table 2.1 summarizes prior results on threshold logic networks for realizing some of the frequently occurring functions in ASICs. For

Table 2.1: Survey of Threshold Logic Realizations of Commonly Used Functions in the Literature

Function	Without Fan-in Bounds	With Fan-in Bounds
Parity	An n-input parity function $PAR_n(X)$ can be computed using a depth d+1 threshold circuit with $\mathcal{O}(dn^{1/d})$ gates.	An n-input parity function $PAR_n(X)$ can be computed using a threshold logic circuit of depth $\mathcal{O}(d\ log\ n/log\ m)$, having $\mathcal{O}(dn/m^{1-1/d})$ gates.
Comparison	The comparison of two n-bit integers can be computed by a threshold logic circuit of depth 3 AND-OR circuits with $3n$ gates. Siu *et al.* (1991).	The comparison of two n-bit integers can be computed by a threshold logic circuit of depth $\lceil(2n/(m-1))\rceil$ with $\lceil(2n/(m-1))\rceil$ gates. Here, $m > 3$.Wagle *et al.* (2020a)
Addition	The sum of two n-bit integers can be computed by a threshold logic circuit of depth 3 AND-OR circuits with $\mathcal{O}(n^2)$ gates.Siu *et al.* (1991)	The sum of two n-bit integers can be computed by a threshold logic circuit of depth 2 having 2n gates, with all the weights of the threshold functions bounded by 2^n and a fanin of m=2n+1. Ramos *et al.* (2003)
Symmetric functions	A symmetric function of n variables, can be implemented using a depth-3 polynomial size threshold circuit having $\mathcal{O}(\sqrt{\sum_{j=1}^{n} w_j})$ threshold gates.Siu *et al.* (1991)	Any n-variable symmetric function can be computed by a threshold logic circuit of depth $\mathcal{O}(d\ log\ n/log\ m)$ of edge complexity $\mathcal{O}(2^{-d/2}nm^{1/(2d-1)}\sqrt{log\ m})$.
Multiplication	Multiplication of two n-bit numbers can be implemented using a depth-4 polynomial size threshold circuit.Siu *et al.* (1991)	The product of two n-bit integers can be computed by a threshold logic circuit of depth $\mathcal{O}(d\ log\ n/log\ m)$ of edge complexity $\mathcal{O}(2^{-d/2}n^2m^{1/(2d-1)}\sqrt{log\ m})$.Siu *et al.* (1991)
Matrix Multiplication	The product of two N by N matrices can be computed by a threshold logic circuit of constant depth $\mathcal{O}(d)$, having $\mathcal{O}(N^{\omega+}\mathcal{O}(\gamma^d))$ gates. Here, $\omega < 3$ and $\gamma < 1$ are constants. d is a positive integer.	Construct by decomposing a threshold logic network without fan-in bounds using the algorithm presented in Annampedu and Wagh (2013b).
Sorting	Let X_1,X_2...X_n be n-input n integers. These integers can be sorted using a threshold network of depth 3, with polynomially bounded number of gates. We call this sorter a basic sorter. Shi *et al.* (2014)	Construct by decomposing a threshold logic network without fan-in bounds using the algorithm presented in Annampedu and Wagh (2013b).
Division	The division of two n-bit integers x and y, i,e. x/y can be performed using a threshold logic network of depth 4, with polynomially bounded number of gates.Siu *et al.* (1991)	Construct by decomposing a threshold logic network without fan-in bounds using the algorithm presented in Annampedu and Wagh (2013b).
Counter	The number of ones in a bus of N-bits can be counted using a threshold logic network of depth 2, with polynomially bounded number of gates.Minnick (1961)	Construct by decomposing a threshold logic network without fan-in bounds using the algorithm presented in Annampedu and Wagh (2013b).
Modulo	If X is an n-bit integer, and m is an integer polynomially bounded by n, then X mod m can be computed using a depth-2 threshold circuit.Siu *et al.* (1991)	Construct by decomposing a threshold logic network without fan-in bounds using the algorithm presented in Annampedu and Wagh (2013b).
Exponent	If X is an n-bit integer, then Z=X^n can be implemented using a depth-4 polynomial size threshold circuit.Siu *et al.* (1991)	Construct by decomposing a threshold logic network without fan-in bounds using the algorithm presented in Annampedu and Wagh (2013b).
Arbitrary Functions	Any Boolean function of n variables can be computed by a depth-3 threshold circuit having $\mathcal{O}(2^{n/2})$ threshold gates. Such a circuit would have at-least $\mathcal{O}(2^{n/3})$ threshold gates.Siu *et al.* (1991)	Construct by decomposing a threshold logic network without fan-in bounds using the algorithm presented in Annampedu and Wagh (2013b).

all the Boolean functions for which a threshold network exists, but without fan-in bounds, Annampedu et al. Annampedu and Wagh (2013b) demonstrated that such networks can be decomposed into a network of threshold gates with a fixed fanin. However, it must be noted that the gate count that results from the work presented in Annampedu et al. Annampedu and Wagh (2013b) is very high.

Although a large body of theoretical work related to the node count, depth, and complexity analysis of networks of threshold functions exists, many of these works have never been realized on chip due to the lack of an energy-efficient solution needed to execute large fan-in threshold functions on hardware. In this dissertation, Chapter 5 will discuss how to decompose large networks of threshold functions into addition, comparison, and logic operations so that they can be executed on hardware in an energy-efficient manner.

2.1.5 Threshold Gates

This subsection presents a survey of a variety of threshold logic gate implementations available in the literature. Table 2.2 summarizes the properties of each of these gates.

The prior work on threshold logic has used different technologies such as bulk CMOS, flash devices, capacitance networks, memristors, RRAMs, etc., to design both sequential and combinational gates.

Table 2.2: Survey of Threshold Logic Gate Architectures in the Literature

TL gate	**Technology (nm)**	**Fan-in**	**Device**	**Stage**	**Cell Type**
Ozdemir *et al.* (1996)	1200	255	Bulk CMOS +CAP	Silicon	Sequential
Bohossian *et al.* (1998)	2000	16	Bulk CMOS +Flash CMOS	Silicon	Combinational
Bobba and Hajj (2000)	180	No info in the paper	Bulk CMOS	Spice	Sequential
Padure *et al.* (2001)	1600	High (Quantification missing)	Bulk CMOS +CAP	Spice	Sequential
Padure *et al.* (2002)	2500	NA. Spice simulations only	Bulk CMOS	Spice	Sequential
Rodriguez-Villegas *et al.* (2002)	350	No info in paper	Bulk CMOS +Flash CMOS	Silicon	Combinational
Padure *et al.* (2003)	250	64	Bulk CMOS	Silicon	Sequential
López-García *et al.* (2004)	600	127	Bulk CMOS +CAP	Layout	Sequential
Kaya *et al.* (2007b)	100	12	DGMOSFET Bulk CMOS	TCAD	Sequential
Rajendran *et al.* (2010)	45	No info in paper	Bulk CMOS +Memristor	Spice	Sequential
Rothenbuhler *et al.* (2013)	240	2	Bulk CMOS +Memristor	Silicon	Sequential
Dara *et al.* (2012)	45	150	Bulk CMOS	Spice	Sequential
Yang *et al.* (2014a)	65	7 (majority gate)	Bulk CMOS +RRAM	Layout	Sequential
Kulkarni *et al.* (2016b)	65	7 (majority gate)	Bulk CMOS	Silicon	Sequential
FTL (Dissertation work)	40	5	Bulk CMOS +Flash CMOS	Layout	Sequential

Threshold gates used to be built with a combination of pull-up/pull-down networks and operational amplifiers (OPAMPs), which led to a fan-in of over 64 (As shown in Table 2.2) but consumed a lot of power and were not ideal for low-power applications. To reduce power consumption, sense amplifiers were proposed to replace OPAMPs in threshold gates, but this resulted in a decrease in fan-in due to the reduced time window for the evaluation of the threshold function. As process technology advanced, there was increased error offset in sense amplifiers, increased variability in the weights of the threshold function, and increased vulnerability to voltage and EMIR fluctuations, which further reduced the fan-in of threshold gates to 5 with limited support for all 5-input threshold functions. This dissertation presents a threshold gate that uses a current-controlled non-volatile transistor to improve the area, power, performance, and robustness of the threshold gate in smaller technology nodes while providing full support for all 5-input threshold functions.

Chapter 3

STANDARD CELL PERCEPTRON

3.1 Introduction

In this chapter Wagle *et al.* (2022a), we introduce a new *programmable ASIC primitive*, referred to as a *flash threshold logic* (FTL) cell. FTL cells are used to replace parts of an ASIC that are threshold functions more efficiently than their equivalent standard-cell counterparts, in a way that substantially improves the area and power consumption of an ASIC, without increasing its delay. It is a mixed-signal circuit that is designed as a *standard cell* so that it is fully compatible with conventional CMOS logic synthesis, technology mapping, and place-and-route tools.

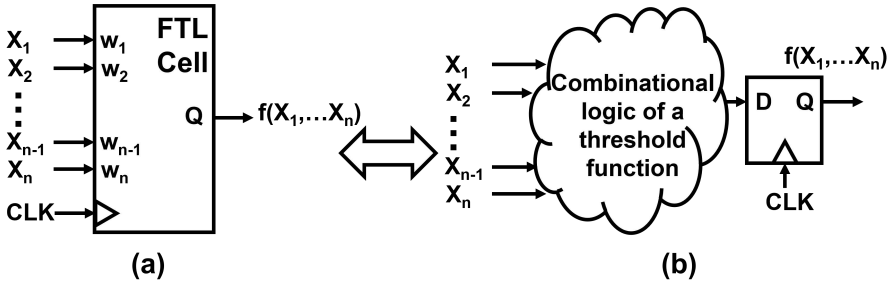
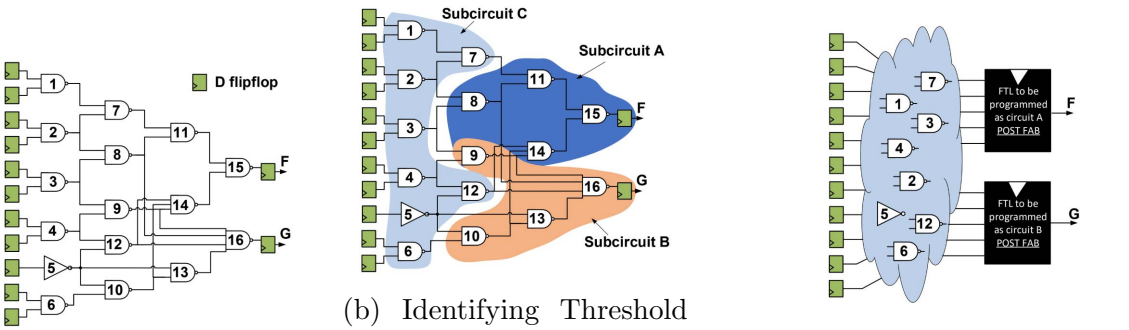


Figure 3.1: (a) FTL Schematic, (B) Functional Equivalent

An FTL cell of n inputs realizes any *threshold* function of n or fewer variables. Figure 3.1(a) shows a block diagram of an FTL cell, in which the weights $\boldsymbol{W}$ are shown as internal parameters of the cell and Figure 3.1(b) shows its functional equivalent. The schematic is meant to convey that the input-output behavior of an FTL cell may be viewed as an *edge-triggered*, *multi-input* flip-flop, whose output is the value of a threshold function, that is internally latched at the rising edge of the clock signal CLK.



(a) A Logic netlist.

(b) Identifying Threshold Functions in TFI Cones of Flip-Flops

(c) An FTL-CMOS Logic Hybrid

Figure 3.2: Use of FTL in ASIC design.

A distinctive characteristic of the FTL cell design is that the actual threshold function realized by an FTL instance within an ASIC is *programmed after the circuit is manufactured*. An FTL-based ASIC integrates *flash* or *floating gate* Cai *et al.* (2013a) transistors along with conventional MOSFETs within the FTL cell. Thus, unlike many of the *emerging technologies* Perricone *et al.* (2017); Yang *et al.* (2014b); Gupta and Jha (2005); Berezowski and Vrudhula (2005), an FTL cell employs mature IC technologies (CMOS and Flash) that are routinely integrated today and commercially manufactured with high yields.

3.1.1 FTL in ASIC Design – Overview

Before proceeding to the details of FTL design, it will be instructive to understand how it can be used in an ASIC Kulkarni *et al.* (2016a), and how it can improve performance, power, and area.

Consider the logic netlist shown in Figure 3.2a which has two registered outputs F and G. Suppose that the transitive fan-in (TFI) cones of F and G are traversed and two subcircuits labeled A and B (see Figure 3.2b) are found, such that A and B are threshold functions of their inputs. The remaining subcircuit is labeled as C.

Suppose that subcircuits A and B are replaced by FTL cells, which are to be later programmed to realize A and B. This replacement is shown in Figure 3.2c, where the FTL cells are shown as black boxes. Now, subcircuit C would be re-synthesized to account for the changes in the delay of FTL cells and the new loads that the inputs of the FTL cells present to the outputs of C. There are two reasons why the circuit in Figure 3.2c would have improved power, performance and area.

1. Subcircuits A and B and the two flip-flops are each replaced by an FTL cell which has much fewer transistors, resulting in a significant reduction in area and power.

2. The clock-to-Q delay of FTL cells turns out to be much less than the total delay (combinational logic delay plus clock-to-Q delay of DFF) of subcircuits A and B, which results in creating a substantial amount of slack (required time minus arrival time) on the outputs of subcircuit C. This in turn will allow synthesis and technology mapping tools to reduce the logic area of subcircuit C. The extent of the improvement depends on how the logic is *absorbed* into the FTL cell.

Note that the reason why the FTL cells are shown as black boxes in Figure 3.2c is to convey the fact that their functions are not known at the time of fabrication because the flash transistors are *programmed* after the chip is manufactured.

3.1.2 Main Contributions

1. An FTL cell is a mixed-signal circuit that is implemented as a standard cell. The new design incorporates flash transistors, which allow its function to be *programmed* after fabrication.

2. The set of threshold voltages of the flash transistors in an FTL cell serve as a proxy for the weights $[\boldsymbol{W}, T]$. The weights can be realized with great fidelity because the flash transistors can be programmed with high precision Cai *et al.* (2013a). However, the relationship between the weights and threshold voltages is a non-linear and multi-valued mapping that depends on the complex electrical and layout characteristics of the MOSFETs and flash transistors. We present a new algorithm called the *modified perceptron learning algorithm* (mPLA) Rosenblatt (1958) that works in concert with HSPICE and *learns* a mapping between the weights and threshold voltages. The mPLA algorithm also maximizes the noise tolerance and robustness of the FTL cell in the presence of process and environmental variations.

3. We present an efficient architecture and methodology for programming the threshold voltages of each flash transistor within each FTL cell that is embedded in an ASIC. We also demonstrate how the post-fabrication threshold voltage assignment capability can be used to improve functional yield and correct timing errors.

4. FTL cells are designed as standard cells to be compatible with existing CMOS design methodologies and tools. We demonstrate this compatibility by using commercial CAD tools to perform synthesis, technology mapping, and place-and-route of several complex function blocks with FTL cells included in the cell library. The results show that automatic embedding of FTL cells results in substantial improvements in the area, power, and wirelength, without sacrificing performance.

The remainder of this chapter is organized as follows. Section 3.2 gives a very brief overview of threshold logic and flash transistor technology. Sections 3.3, 3.4,

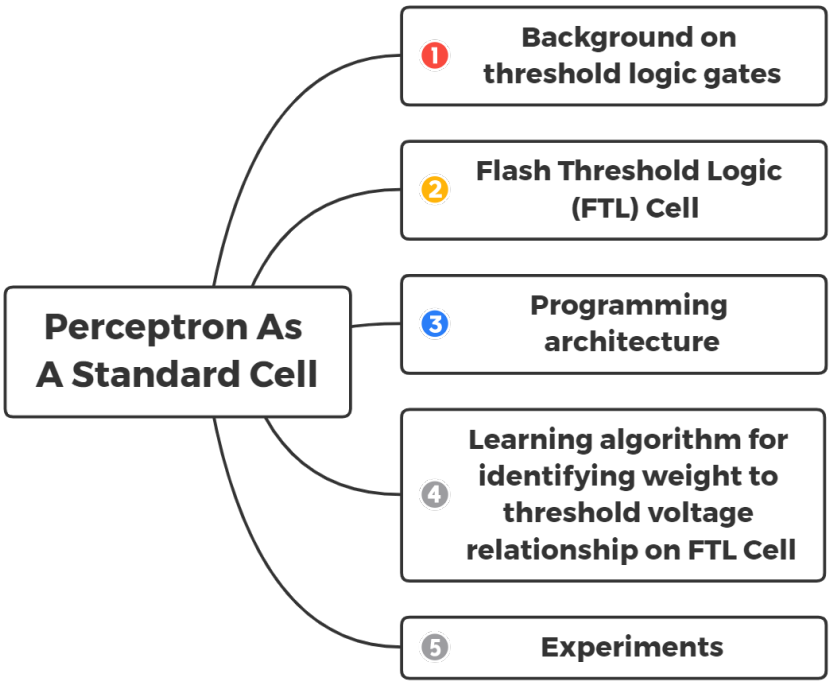


Figure 3.3: Organization of the Chapter

3.5 and 3.6 contain the main body of this work. The architecture and operation of the FTL cell are described in Section 3.3. Section 3.4 explains the mechanism for programming FTL cells once they are embedded in an ASIC, using a separate scan chain reserved for that purpose. Section 3.5 describes the mapping between the weights of a threshold function and the threshold voltages of the flash transistors in the FTL, considering factors such as robustness to noise, process variations and circuit delay. Section 3.6 contains an extensive set of experimental results, demonstrating the significant improvements in PPA of FTL cells over their CMOS equivalents both at cell-level and when they are integrated into ASICs. It also includes results validating several uses of post-fabrication programming/tuning of the flash devices. Conclusions appear in Section 3.7.

3.2 Background

3.2.1 Threshold Logic Gates

Since 1960s then there has been a substantial body of work on new circuit architectures and implementations of threshold logic. Surveys of designs prior to 2003 appear in Beiu *et al.* (2003); Beiu (2003); Celinski *et al.* (2003), detailing over fifty different implementations. The earlier works and even later ones such as Lageweg *et al.* (2002); Mozaffari *et al.* (2018); Zhang *et al.* (2005); Annampedu and Wagh (2013a); Mozaffari and Tragoudas (2018); Kaya *et al.* (2007a); Yang *et al.* (2014b), have not been integrated into mainstream VLSI design. It is, however, still is very valuable to develop efficient implementations of threshold functions. This is due to the fact that many Boolean functions that require large AND/OR networks can be realized by smaller, fixed depth threshold networks Siu *et al.* (1995) and nearly 70% of the functions in two standard cell libraries (observed in a 65nm and a 40nm library from different vendors) are threshold functions.

Recently, Kulkarni *et al.* (2016a) reported an architecture of a threshold gate and showed how it can be integrated with the standard-cell ASIC design methodology using commercial tools. Unlike our approach, Kulkarni *et al.* (2016a) uses only CMOS devices. In addition, they reported significant improvements in PPA of an actual silicon implementation of ASIC with threshold gates Yang *et al.* (2015). Their architecture, however, severely limits the number of threshold functions that can be implemented because the width of the transistors are made proportional to the weights. This limits the fan-in and consequently, only 12 of the 117 threshold functions of 5 or fewer inputs were implemented in Kulkarni *et al.* (2016a). In contrast, our work implements all 117 threshold functions of 5 or fewer inputs because of the use of flash transistors.

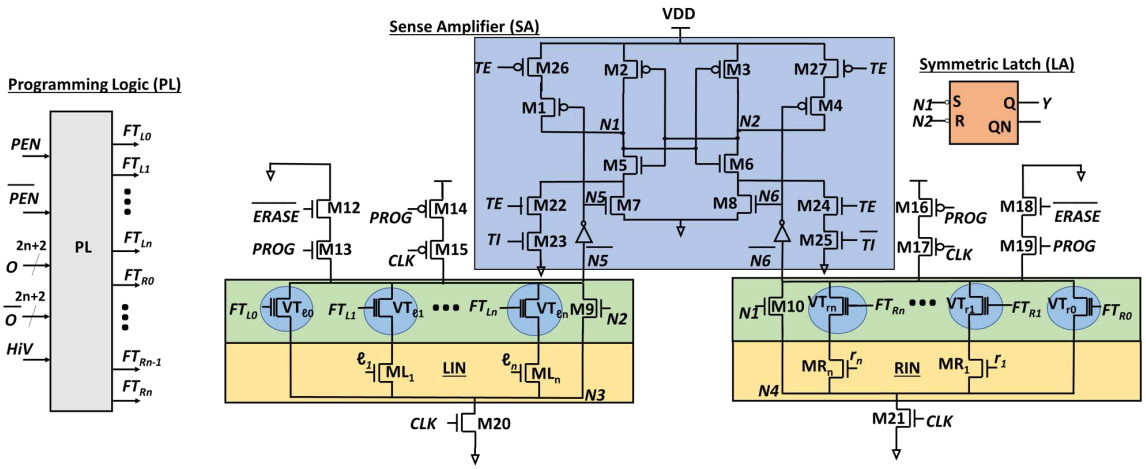


Figure 3.4: FTL Cell Architecture Showing LIN, RIN, Sense Amplifier (SA), Latch (LA), and Programming Logic (PL).

3.3 Flash Threshold Logic (FTL) Cell

Figure 3.4 shows the architecture of the FTL cell. It has five main components: (i) the left input network (LIN), (ii) the right input network (RIN), (iii) a sense amplifier (SA), (iv) an output latch (LA), and (v) a flash transistor programming logic (PL). The LIN and RIN consist of two sets of inputs $(\ell_1, \cdots, \ell_n)$ and $(r_1, \cdots, r_n)$, respectively, with each input in series with a flash transistor. In our implementation, $\ell_i = r_i = x_i$ for all i. The state of the inputs and the threshold voltages of the flash transistors determines the conductance of the LIN and RIN. The assignment of signals to the LIN and RIN is done to ensure sufficient difference in their conductance across all minterms.

There are two differential signals $N1$ and $N2$ in an FTL cell, which serve as inputs to an SR latch. When $[N1, N2] = [0, 1]$ ($[1, 0]$), the latch is set (reset) and the output $Y = 1(0)$. The magnitudes of the two sides of the inequality in the definition of a threshold function (see Equation 2.1) are mapped to the *conductance* G_L of the LIN and G_R of the RIN. Ideally, the mapping is such that $[N1, N2] = [0, 1] \Leftrightarrow$

$G_L > G_R$ and $[N1, N2] = [1, 0] \Leftrightarrow G_L < G_R$. As stated earlier, the flash transistor threshold voltages serve as a proxy to the weights of the threshold function – the higher the weight, the lower will be the threshold voltage. For a given threshold function, this non-linear monotonic relationship is *learnt* using a modified perceptron learning algorithm described in Section 3.5.

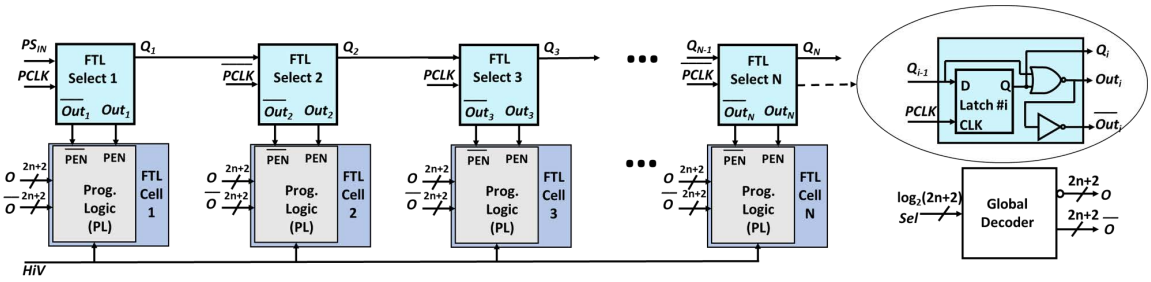


Figure 3.5: Programming Scan Chain for FTL Cells in an ASIC.

The FTL cell has four modes: *regular*, *erase*, *programming* and *scan-testing* mode. The VT values of the flash transistors are set in the programming mode and erased in the erase mode. The logic operation of an FTL cell takes place in regular mode.

FTL Regular Mode: $(PROG = ERASE = 0,\ TE = 0, HiV = 0)$. Assume that the VTs of the flash transistors have been set to appropriate values corresponding to the weights of the threshold function, and their gates are being driven to 1 by setting FT_i to VDD. When $CLK = 0$, the circuit is reset. In this phase, the nodes $\overline{N5}$ and $\overline{N6}$ of LIN and RIN are connected to the supply, $N5 = N6 = 0$, and $N1 = N2 = 1$. Therefore, the output Y remains unchanged.

Assume now that an on-set minterm is applied to the inputs in the LIN and RIN. With properly assigned VT values to the flash transistors, suppose that $G_L > G_R$ for the given minterm. When $CLK : 0 \rightarrow 1$, both the LIN and RIN will conduct, and $N5$ and $N6$ will both transition from $0 \rightarrow 1$. Assuming $G_L > G_R$, $N5$ rises faster than $N6$, and hence $N5$ will make $M7$ active before $N6$ makes $M8$ active. This will start to discharge $N1$ before $N2$. When $N1$ falls below the threshold voltage of

$M6$, it will stop further discharge of $N2$, and turn on $M3$, resulting in $N2 : 0 \rightarrow 1$. Finally, [N1,N2] = [0,1] sets the SR latch, resulting in $Y = 1$. For an off-set minterm, $G_L < G_R$, and $[N1, N2] = [1, 0]$ resulting in $Y = 0$.

FTL Program, Erase and Scan-testing mode: Figure 3.4 shows a circuit block labeled PL (programming logic) that generates the signals to select and program the FTL cells at the chip-level. Details of the programming architecture and protocol are given in Section 3.4. During flash-programming of a single FTL, the PL redirects HiV to FT_{i}, to program the i^{th} flash transistor.

FTL Programming Mode: ($ERASE$=0, $PROG$=1, CLK=0, HiV=20V, TE=0). The ERASE and PROG signals turn on M12 and M13 and turn off M14. In this state, the source of the flash transistor is floating while the drain and bulk are connected to the ground. Activating the appropriate signals in the PL unit causes high voltage pulses to be applied to the HiV line and the gate of the flash transistor to set the desired threshold voltage (VT).

FTL Erase Mode: ($ERASE$=1, $PROG$=1, CLK=0, HiV=-20V, TE=0). The ERASE signal turns off M12. Both the source and drain of the flash transistors are floating in this state, while the bulk is connected to the ground. Using the PL unit, the gate terminals of all the flash transistors in the FTL are connected to HiV. A negative high voltage pulse at HiV in this state will tunnel the charge from the floating gate, thereby erasing the flash transistor.

FTL Scan-testing Mode: ($ERASE$=0, $PROG$=0, CLK=0, HiV=0, TE=1). The scan-testing mechanism in the FTL cells is implemented in the same way as described in Kulkarni *et al.* (2016a). It uses the *test enable* (TE) and *test input* (TI and $\bar{TI}$) signals. In this mode, TE acts as the clock with the main clock $CLK = 0$. Hence the scan-testing chains for the D flip-flops and FTL cells are kept separate. The procedure to inject data into the scan-testing chain of FTL cells is straightforward.

On each TE cycle, the bit to be scanned in is applied to TI. Then $TE : 0 \rightarrow 1$, which causes either $N1$ or $N2$ to discharge, resulting in the output latch being set or reset. This process is repeated to load all FTLs with the data in a test vector. Transistors $M26$ and $M27$ block potential DC paths from VDD to VSS during testing.

Note that the problem of read and write disturb Bez *et al.* (2003); Cai *et al.* (2015) found in NAND flash memories do not exist with an FTL cell because there is only one flash transistor in each stack in the input network. Also, the problem of *write endurance* Boboila and Desnoyers (2010) in flash memories, which refers to a limit on the number of writes (from 10K - 100K cycles), is not an issue with FTL cells, because an FTL cell would be programmed or erased only a handful of times over the life of the design.

3.4 Architecture for programming FTL cells

In this section, we describe the programming architecture used for FTL cells embedded in an ASIC. This architecture individually addresses the flash transistors of the FTL cells and then redirects HiV pulses to them. Although this architecture is a part of the ASIC, its use ends once the FTL cells are programmed. Therefore, its design must meet its functional requirements without severely impacting the overall area and wirelength of the ASIC. This is achieved by a scan-chain programming architecture.

Figure 3.5 shows the structure of the programming scan chain. Each stage of this chain consists of an FTL cell with its programming logic and a select cell that identifies the FTL cell to be programmed. Suppose that the FTL cells realize all threshold functions of n or fewer variables.[1] Then each such cell has $2n + 2$ flash transistors. Suppose further that there are N FTL cells. Initially, all Q_is are set to

[1] In the experimental results, $n = 5$.

1. Then cell i is selected by making $Q_i = Q_{i-1} = 0$, while all other Qs remain at 1. Thus, clocking in the appropriate sequence using PCLK selects cell i. Since each FTL cell has $2n + 2$ flash cells, a global decoder with $\log(2n + 2)$ inputs and $2n + 2$ outputs is used. These outputs of the decoder activate the appropriate path for the HiV pulses to the inputs of the flash transistors of the selected FTL cell.

The programming architecture involves the use of high-voltage nets. In the physical layout, the high voltage wires are bundled, and wire-shielding Mehri and Masoumi (2015) is used to avoid any cross-talk due to high voltage signals to the other low voltage lines and transistors. Programming is done with a dedicated scan chain, and all the associated high-voltage nets are systematically bundled and shielded. This results in reducing the total wirelength of those nets. Furthermore, since it is a linear iterative array, it easily scales to accommodate any number of cells.

Assuming that FTL cells use floating gate transistors, the program and erase modes require +20V and -20V (HiV) pulses to be applied to their inputs (see Section C). Note that other flash technologies such as SONOS Nii *et al.* (2020), MONOS-Tsuda *et al.* (2016), and High-K Metal Gate (HKMG) Khan *et al.* (2019b) require similar infrastructure for programming and erasure, but may differ in the voltage levels of the pulses.

3.5 Computing the relationship between the weights and the V_T values for an FTL cell

3.5.1 Overview

Let $\boldsymbol{VT}_\ell(f) = (VT_{\ell_0}(f), \cdots, VT_{\ell_n}(f))$, and $\boldsymbol{VT}_r(f) = (VT_{r_0}(f), \cdots, VT_{r_n}(f))$ denote the threshold voltages of the flash transistors in the LIN and RIN of an FTL, respectively (see Figure 3.4). Further, let $\boldsymbol{VT}(f) = (\boldsymbol{VT}_\ell(f), \boldsymbol{VT}_r(f))$. In this

section, we present an algorithm to determine these voltages for an FTL to realize a given threshold function f having a weight vector $[\boldsymbol{W}, T]$.

To configure an FTL, the method described herein determines $\boldsymbol{VT}(f)$ for each f a priori, using models that include circuit parasitics and global and local process variations in the device and circuit parameters. This is to ensure that an overwhelming majority ($\gg 99\%$) of the FTL instances on a chip can be programmed by attempting at most a few pre-computed values of $\boldsymbol{VT}(f)$. The remaining small fraction of FTLs for which a feasible, model-based $\boldsymbol{VT}(f)$ could not be found, can be programmed directly on the chip.

Let $G_L(x|\boldsymbol{VT}(f))$ and $G_R(x|\boldsymbol{VT}(f))$ for $x \in B^n$, denote the conductance of the LIN and RIN as functions of a minterm x of f and the flash transistor threshold voltages. Henceforth, for clarity, we refer to $G_L(x|\boldsymbol{VT}(f))$ and $G_R(x|\boldsymbol{VT}(f))$ as G_L and G_R respectively.

The problem is to find a $\boldsymbol{VT}(f)$ that determines a mapping between the Boolean space B^n and the conductance space (G_L, G_R) such that, in the *ideal* case,

$$\begin{aligned} G_R < G_L, \text{ if } f(x) = 1 \\ G_R > G_L, \text{ if } f(x) = 0. \end{aligned} \tag{3.1}$$

This mapping, depicted in Figure 3.6, is one-to-many, since there can be many (an uncountable number) *feasible* values of $\boldsymbol{VT}(f)$ for a given f with a weight vector $[\boldsymbol{W}, T]$.

In practice, to avoid variations due to parasitics which could make the circuit behavior erroneous, we require finding a subset of the feasible set where

$$\begin{aligned} G_R < G_L - \Delta_L, \text{ if } f(x) = 1 \\ G_R > G_L + \Delta_R, \text{ if } f(x) = 0 \end{aligned} \tag{3.2}$$

for some some sufficiently large $\Delta_L \in (0, G_L)$ and $\Delta_R \in (0, G_R)$. Note that Δ_L and Δ_R are related due to the constraints imposed by the truth table.

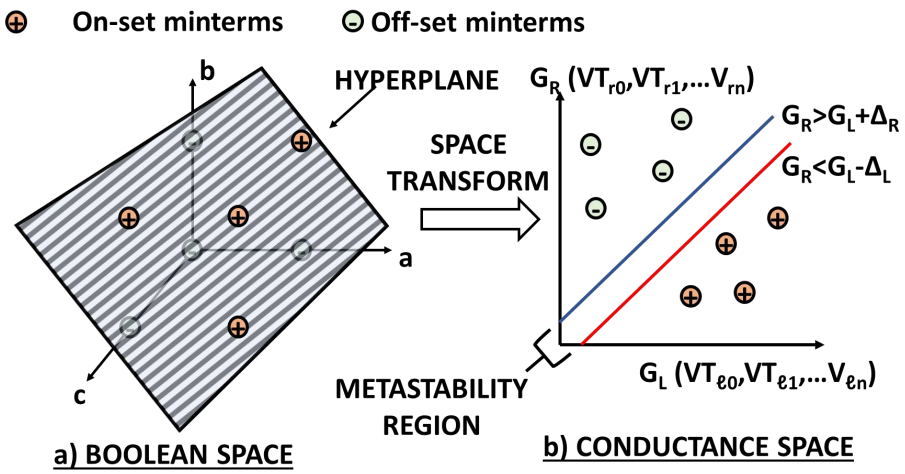


Figure 3.6: Transformation from Boolean Space to Conductance space.

Our approach to find $\boldsymbol{VT}(f)$ consists of three steps which are implemented by Algorithms mPLA^0, mPLA$^+$, and mPLA^{++}. These are described in the following sections.

3.5.2 Algorithm mPLA^0

Algorithm mPLA^0 is a modified version of the classical perceptron learning algorithm (PLA) Rosenblatt (1958) that works in concert with HSPICE (for verifying the truth table of f) to search through the space of values of $\boldsymbol{VT}(f)$ until each minterm (a point in the (G_L, G_R) space) of f is correctly classified. It does this by iteratively adjusting the threshold voltages of flash transistors such that points in the conductance space that correspond to the on-set and off-set minterms satisfy the constraints in Equation 3.2 (see Figure 3.6). It calls HSPICE (line 3 of Algorithm mPLA^0) to determine whether any point falls above or below the lines corresponding to these inequalities. Since a layout extracted FTL circuit is being used, the circuit parasitics are accounted for in the HSPICE simulation. Consequently, Algorithm mPLA^0 implicitly finds values of Δ_L and Δ_R.

Algorithm mPLA^0 Modified Perceptron Learning Algorithm

Input: Truth table TT of threshold function f

Output: $\boldsymbol{VT}_0$ to program FTL cells with f

1: Initialize $\boldsymbol{VT}_0$
2: **for** $k = 0$ to $kmax - 1$ **do**
3: OT = SPICE($\boldsymbol{VT}_0$) // Truth table from simulation
4: **if** TT and OT disagree on some minterm m **then**
5: **if** TT(m)==1 **then**
6: Update $\boldsymbol{VT}_0$: decrement (increment) the VT of every active transistor in LIN (RIN) that is '1' in m by δ
7: **else**
8: Update $\boldsymbol{VT}_0$: increment (decrement) the V_T of every active transistor in LIN (RIN) that is '1' in m by δ
9: **end if**
10: **else**
11: Break
12: **end if**
13: **end for**

Given the truth table (TT) of f, mPLA^0 applies all the minterms of f to the FTL cell and records the HSPICE response in OT (output table). If $TT(m) \neq OT(m)$, for some minterm m, then the constraint in Equation 3.2 was not satisfied. In such a case, mPLA^0 adjusts the values of $\boldsymbol{VT}^0(f)$ (Algorithm mPLA^0 line 4-9) associated with the active input transistors within the interval $[\delta, V_{DD} - \delta]$, by a minimum increment δ, according to Equation 3.3. Here, m_ℓ (m_r) is a binary vector that identifies the active input transistors in the LIN (RIN).

$$\mathbf{VT}_{\ell}^{k+1} = \begin{cases} \mathbf{VT}_{\ell}^{k} - \delta m_{\ell} & \text{if } m \cdot W \geq T \\ \mathbf{VT}_{\ell}^{k} + \delta m_{\ell} & \text{if } m \cdot W < T \end{cases}$$
$$\mathbf{VT}_{r}^{k+1} = \begin{cases} \mathbf{VT}_{r}^{k} + \delta m_{r} & \text{if } m \cdot W \geq T \\ \mathbf{VT}_{r}^{k} - \delta m_{r} & \text{if } m \cdot W < T \end{cases} \tag{3.3}$$

The term δm_l (or δm_r) is a vector that has a value δ at all locations in LIN (RIN), which are 1 for a minterm m, and zero elsewhere. For instance, consider the threshold function $b + c \geq a + T$. Let $m = 110$ be an on-set minterm that was incorrectly evaluated. If $TT(m) \neq OT(m)$ then $G_R > G_L - \Delta_L$. Therefore G_L needs to be increased (threshold voltages corresponding to the flash transistors of b and c will be decreased) and G_R needs to be decreased (threshold voltages corresponding to the flash transistors of a and T will be increased) for minterm m. Consequently, the threshold voltages of all the flash transistors associated with the active input transistors should be decreased (increased) by δ in the LIN (RIN). A similar change is made if m is an off-set minterm. This is what is expressed in Equation (3.3). $\boldsymbol{VT}^0(f)$ is the value returned by Algorithm mPLA^0.

If a given set of points in B^n is linearly separable (i.e. a threshold function), then the PLA algorithm will terminate in a finite number of iterations Siu *et al.* (1995); McCulloch and Pitts (1988). Similarly, given a threshold function f, a sufficiently small δ and an FTL instance for which there exists a feasible $\boldsymbol{VT}(f)$, Algorithm mPLA^0 will terminate in a finite number of steps (see Siu *et al.* (1995) for proof of termination). For an n-input threshold function, the upper bound on the number of iterations of the PLA given in Siu *et al.* (1995) becomes $kmax = 2(n+1)||\boldsymbol{VT}^0(f)||^2/\delta^2$. For instance, with $n = 5$ and $\delta = .02V$, $kmax = 3000||\boldsymbol{VT}^0(f)||^2$.

3.5.3 *Algorithm mPLA$^+$: Improving Noise Tolerance*

Algorithm mPLA^0 does not consider the relative location of the points with respect to the metastability region defined by the lines $G_R = G_L - \Delta_L$ and $G_R = G_L + \Delta_R$ (see Figure 3.6b). Even though minterms are classified correctly, they can be arbitrarily close to the metastability region. The further a minterm is from this region, the easier (and faster and more robust) it will be for the sense amplifier to detect the difference between $N5$ and $N6$, and discharge the appropriate side ($N1$ or $N2$) first. Thus, maximizing Δ_L and Δ_R within the feasible set will maximize its noise tolerance.

Algorithm mPLA$^+$ repeatedly calls mPLA^0 to maximize Δ_L and Δ_R. It does this by introducing a *hypothetical* capacitance C_1 on node $\overline{N5}$ (which is introduced in HSPICE) when classifying an on-set minterm, and determining the maximum value of C_1 for which Algorithm mPLA^0 converges. This modification *handicaps* node $\overline{N5}$ and directs the algorithm to find a solution, that will result in an increased gap between G_L and G_R. Similarly, we add a capacitance C_0 on node $\overline{N6}$, when classifying an off-set minterm. Since the values of Δ_L and Δ_R are linearly proportional to C_1 and C_0, respectively, the separation between the lines $G_R = G_L - \Delta_L$ and $G_R = G_L + \Delta_R$ increases, which in turn forces the training algorithm to produce a threshold voltage assignment $\boldsymbol{VT}^+(f)$ in a more robust (and also faster) FTL cell. Note that C_1 and C_0 are only used during HSPICE simulations, and are not part of the actual FTL cell.

Figure 3.7 shows the results of running Algorithms mPLA^0 and mPLA$^+$ on a test function (Muroga (1971c)) $f_{115}(a,b,c,d,e) : (\boldsymbol{W}, T) = [4,1,1,1,1;5] = a(b+c+d+e)$. It is a plot of the minterms in the conductivity space that was obtained by using HSPICE after programming the FTL using $\boldsymbol{VT}^0(f)$ and $\boldsymbol{VT}^+(f)$. The largest values of C_1 and C_0 for which a feasible solution was obtained was $0.1fF$. The plot shows

that training with the hypothetical capacitance values separates the two closest on-set and off-set minterms in the conductivity space by more than five times. Furthermore, the delay of an FTL programmed with $\boldsymbol{VT}^{+}(f)$ will be smaller than the one that is programmed with $\boldsymbol{VT}^{0}(f)$.

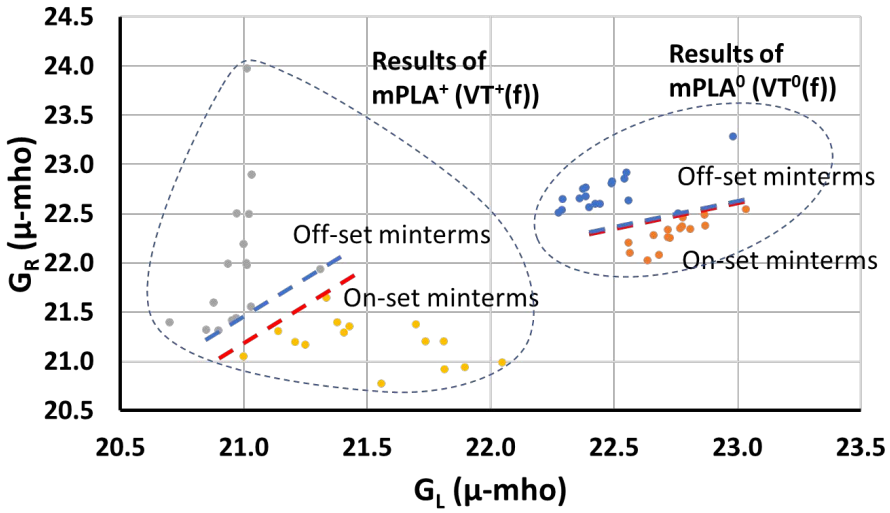


Figure 3.7: Conductance G_L and G_R of FTL Cell Programmed for $F = [4, 1, 1, 1, 1; 5]$ Using mPLA^0 and mPLA$^+$ ($[TT, 0.9V, 25°C]$).

3.5.4 Algorithm mPLA^{++}: Optimizing Yield

The threshold voltages $\boldsymbol{VT}^{+}(f)$ computed by the mPLA$^+$ are aimed at achieving maximal separation between the on-set and off-set minterms based on model-based estimates of parasitics. This has the twin advantages of increasing the noise margin and reducing the delay. Despite this, inevitable manufacturing variations can still result in reducing the difference between G_L and G_R associated with $\boldsymbol{VT}^{+}(f)$ of the two closest minterms, which may result in the incorrect evaluation of the intended threshold function. In this section, we present a *predictive* technique to *pre-compute* a small **set** of $\boldsymbol{VT}s(f)$ for each threshold function f which would cover a very high percentage of manufactured variations.

Among the N manufactured FTL cells programmed to realize function f using $\boldsymbol{VT}^{+}(f)$, suppose that N_e were erroneous and let $\{\mathrm{FTL}_1(f), \mathrm{FTL}_2(f), \cdots, \mathrm{FTL}_{N_e}(f)\}$ be the erroneous instances. The problem is to find individual threshold voltage assignments for each of these N_e instances so that each will correctly realize f. Our approach is motivated by two observations.

First, each erroneous function in $\{f_i^e, 1 \leq i \leq N_e\}$ is itself a threshold function. This is simply due to the fact that by construction, an FTL cell only computes threshold functions (see Figure 3.1). Second, our experiments show that a large number of different instances of an FTL cell, which are reprogrammed with $\boldsymbol{VT}^{+}(f)$ and are to realize the same function f, realize the same erroneous function f^e. This suggests that all the erroneous FTL cell instances can be grouped into a few equivalence classes, called *error-types*, with two FTLs belonging to the same error-type if they realize the same erroneous function.

Given a threshold function f, Algorithm mPLA^{++} first generates a *set* of N_{MC} Monte Carlo (MC) instances of an FTL cell and identifies the N_e erroneous instances (i.e. those when programmed with $\boldsymbol{VT}^{+}(f)$ do not realize f). The N_e erroneous instances are grouped into M_f error-types. Let $f_i^e, 1 \leq i \leq M_f$, denote the logic functions of the *distinct* error-types observed in a sample of N FTLs. Algorithm mPLA^{++} selects one MC instance from each error-type class and computes one $\boldsymbol{VT}^{+}(f)$ assignment for that instance using mPLA^{+}. It returns a set of threshold assignments,

$$\boldsymbol{VT}^{++}(f) = \{\boldsymbol{VT}^{+}(f_1^e), \boldsymbol{VT}^{+}(f_2^e), \cdots, \boldsymbol{VT}^{+}(f_{M_f}^e)\}, \tag{3.4}$$

one for each error-type for each function f.

Results presented in Section 3.6 show that using the $\boldsymbol{VT}^{+}(f_i^e)$ computed for one FTL instance from i^{th} error-type ($1 \leq i \leq M_f$) resulted in *all* the instances of the same error-type correctly realizing f. This works because instances that have the

Algorithm mPLA^{++} Modified Perceptron Learning Algorithm Accounting for Process Variations

Input: TT of f, N_{MC}

Output: $\boldsymbol{VT}^{++}(f)$ to program FTL cells with f

1: Execute mPLA^{+} to compute $\boldsymbol{VT}^{+}(f)$

2: Using MC sampling of the parameter space, generate $N_{MC}(f)$ instances of an FTL cell, and program them with $\boldsymbol{VT}^{+}(f)$.

3: Among the set of N_{MC} instances, let N_e be the number of instances, which when programmed with $\boldsymbol{VT}^{+}(f)$, realize a function other than f, and among these N_e, let M_f be the number of erroneous functions that are distinct.

4: Execute the mPLA^{+} on one MC instance from each of the M_f erroneous functions to obtain

$$\{\boldsymbol{VT}^{++}(f)\} = \{\boldsymbol{VT}^{+}(f_1^e), \cdots, \boldsymbol{VT}^{+}(f_{M_f}^e).\}$$

same error-type share similar parasitic variations. Thus, all instances of our sample of FTL cells were correctly programmed using one distinct $\boldsymbol{VT}^{+}(f_i^e)$ for each error-type.

There is no guarantee that the erroneous functions found in a sample set N_{MC} will capture all manufacturing outcomes. This means that there may be some manufactured FTLs that could not be correctly programmed using any of the threshold voltage vectors computed by Algorithm mPLA^{++}. For these remaining FTLs, our approach is to apply Algorithm mPLA^{0} directly on the chip. In each iteration of mPLA^{0}, the step that adjusts the threshold voltages of flash transistors is replaced by the application of an appropriate number of positive or negative pulses to the FTL cell on the chip using the programming scan chain. This capability of correcting the function of a cell after fabrication to increase yield is a signature attribute of the

proposed design methodology.

3.6 Experimental Results

3.6.1 Experiment Setup

An FTL cell with $n = 5$ (see Figure 3.4 in Section 3.3) was designed, and a complete layout (including the programming devices) was created using the TSMC 40nm LP library. It was laid out as a double-height cell requiring 24 tracks. The flash transistor models were obtained from Abusultan and Khatri (2016a) and were suitably modified to reflect the characteristics and variations of the TSMC 40nm LP library. The design rules for the flash transistors were obtained from ITRS. The standard cell area of the FTL was 15.6 μm^2.

There are a total of 117 distinct positive-form threshold functions of five or fewer variables. A numbered list of these is given in Muroga (1971c). The one cell that was designed was copied 117 times, and each was trained to realize one of the 117 threshold functions. In this section, we use the same numbering scheme as in Muroga (1971c) to identify the functions. The FTL cell trained to implement the threshold function numbered n in Muroga (1971c) will be referred to as FTL_n, and the corresponding CMOS implementation will be denoted as $CMOS_n$. The threshold function itself will be denoted as f_n. **Note:** In all the bar charts shown in this section, the numbers on the x-axis identify the threshold function. Function f_0 is a buffer and is omitted because this would correspond to a DFF, which by itself would never be replaced by an FTL in an ASIC. The first function shown is f_1, which is a two-input AND.

3.6.2 *Training Iterations*

mPLA$^+$ was used to train the FTL cell for robustness (see Section 3.5.3) for all 117 functions. Figure 3.8 shows the number of iterations needed for training each of the 117 functions. The actual number of iterations was about 10X lower than the theoretical upper bound, presented in Section 3.5.2.

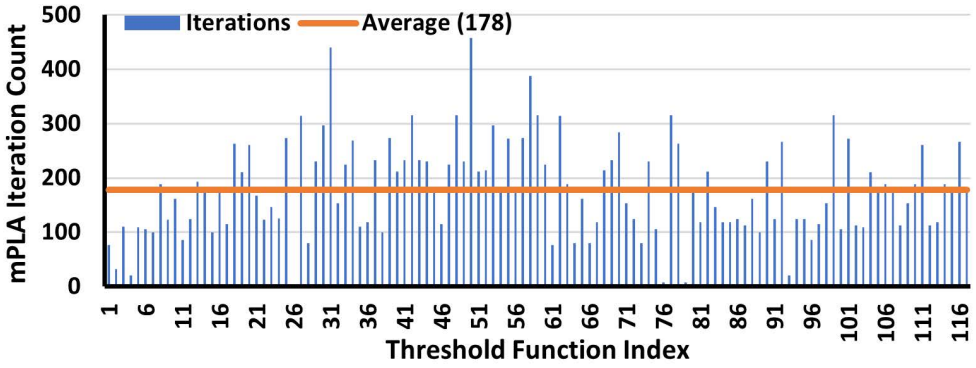


Figure 3.8: Iteration Count for mPLA$^+$ for All 117 Functions of 5 or Fewer variables.

3.6.3 *Individual Cell Area, Delay, and Power Comparison*

All 117 threshold functions of five or fewer variables were implemented using FTL cells. These functions were also synthesized by Cadence Genus and placed and routed using Cadence Innovus, using the conventional TSMC 40nm LP standard cells. Two sets of experiments were performed to compare the CMOS equivalent designs to the FTL cells: (1) delay optimal and (2) area optimal synthesis. The results comparing the total delay (sum of logic delay, setup-time, and clock-to-Q delay), area, and power of these circuits and the corresponding FTL implementations are shown in Figures 3.9(a) and 3.9(b), respectively.

The results show that FTL cells have the advantage of speed. Optimizing their CMOS equivalents to meet the delay of the corresponding FTL cells forces the synthesis algorithms to use high drive strength cells (larger area) for the combinational

logic and larger DFFs. As the FTL implementations are faster than the fastest CMOS equivalent implementation, delay optimal synthesis results in an across-the-board improvement in all FTL cells in delay, area, and power.

When synthesizing individual cells for the minimum area, FTL cells are still uniformly faster. However, the synthesis algorithm now uses the smallest combinational cells and DFFs in the CMOS equivalents. In this case, although the CMOS implementations of simpler functions are much smaller than their FTL equivalents (see Figure 3.9(b)), the FLT versions are still smaller for 74 out of 117 functions because the *logic absorbed* by the FTL cell results in greater area savings than the smaller drive strength cells used in the CMOS equivalents.

The dynamic power of every FTL implementation is higher than its CMOS equivalent for area optimal synthesis. The reasons for this are (1) an FTL cell resets and then evaluates its function on every clock cycle and (2) the much smaller switched capacitance of the low-drive strengths of the combinational logic in the CMOS equivalents. Figure 3.9(a) shows that FTL cells have a much lower power-delay product (i.e. energy) when their CMOS equivalents are synthesized for minimum delay. Figure 3.9(c) shows that this is also true for the majority of the CMOS equivalents when they are synthesized for the minimum area. Hence, FTL cells are, in general, more energy efficient.

Figures 3.9(d) and 3.9(e) show a comparison of the leakage power of FTL cells and their CMOS equivalents. The leakage of FTL cells is practically independent of the function, and in the case of delay optimal synthesis, it is far lower than every CMOS equivalent circuit. Exactly the opposite is true for the area optimal synthesis due to reduced sizes of the combinational cells and DFFs. In these plots, the circuit indices are ordered by increasing area. The area trend lines show that the leakage increases with the area for the CMOS implementations.

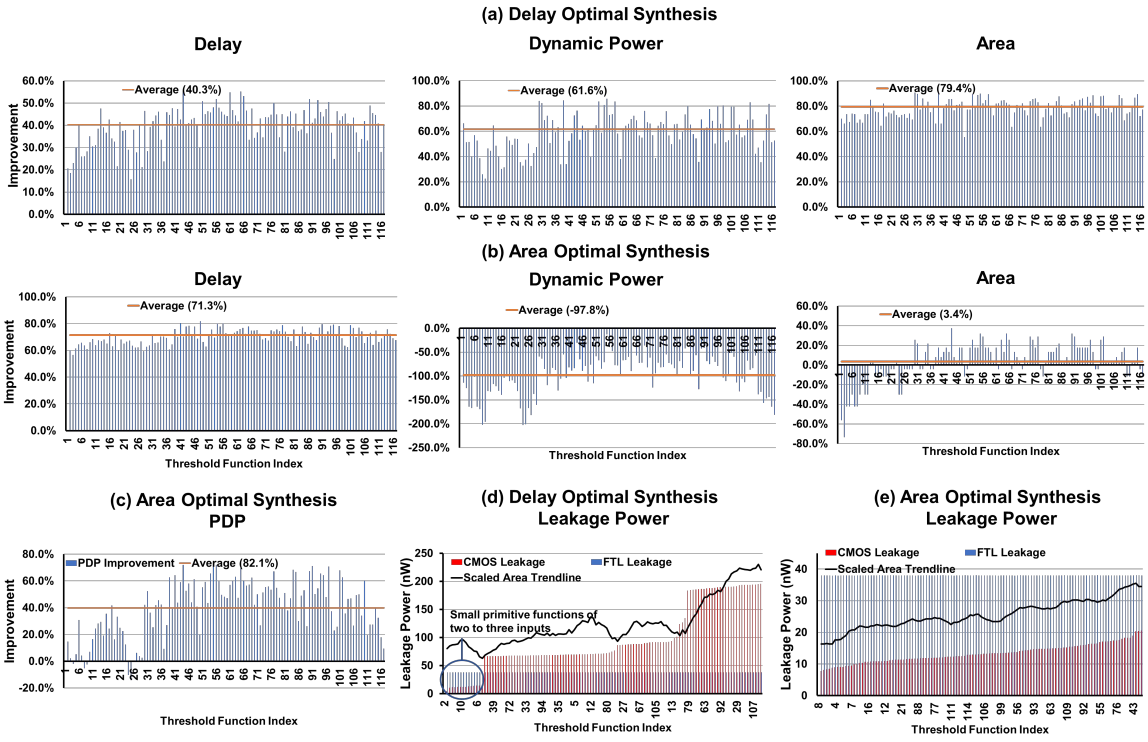


Figure 3.9: PPA Improvements of FTL over CMOS Implementations. Simulations Done at 25°C Assuming a 20% Input Switching activity.

3.6.4 Delay Distributions

This experiment compares the distributions of delays of FTL and CMOS implementations. We show the results for the threshold function f_{35} with a weight vector $[\boldsymbol{W}; T] = [3, 3, 2, 1, 1; 8]$. The PVT corner setting was $[TT, 0.9V, 25°C]$. 100K Monte Carlo instances were generated for both FTL_{35} and $CMOS_{35}$. Each of the 100K FTL instances was verified against the truth table for functional correctness, for both FTL_{35} and $CMOS_{35}$. Figure 3.10 shows the histogram of delays for both circuits. These demonstrate the delay advantage of the FTL cell over its CMOS equivalent, even in the presence of process variations. The difference in standard deviation between the two is insignificant. Note that the FTL instances with large delays can be

re-programmed to reduce the delay further. This capability is not possible for the CMOS versions.

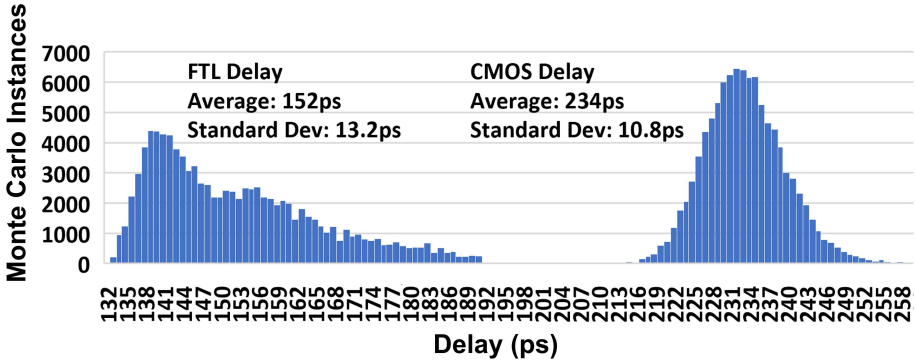


Figure 3.10: Delay Histogram of FTL_{35} and $CMOS_{35}$ with 100K Monte Carlo Simulations. $PVT = [TT, 0.9V, 25°C]$.

3.6.5 Dynamic Voltage Scaling

Voltage scaling is a common mechanism to trade off performance against power. Table 3.1 shows the results of training FTL_{35} at $0.9V$. The FTL cell was programmed with the determined set of flash threshold voltages, and then operated over the voltage range $[0.8V, 1.1V]$. To ensure proper operation across all voltages, the gate voltages of the flash transistors were scaled accordingly. This result demonstrates how a single $\boldsymbol{VT}^{+}(f)$ assignment can be used for dynamic voltage scaling. The delay of the FTL_{35} varies by 2.5X (its CMOS equivalent by 2.8X), power varies by 5.9X (CMOS equivalent by 1.9X), and the PDP (energy) varies by 2.3X (CMOS equivalent by 1.43X), as the supply voltage varies over [0.8V, 1.1V].

3.6.6 Number of programming pulses

Figure 3.11 shows the number of high voltage pulses needed to program the 117 threshold functions. The number of high voltage pulses is estimated, assuming that

Supply Voltage (V)	Flash Gate Voltage (V)	Power (uW)	Delay (ps)	PDP
0.8	0.8	14.3	198.1	2837.1
0.85	0.825	20.5	157.6	3228.7
0.9	0.85	26.1	130.2	3396.9
0.95	0.875	40.3	111.2	4482.7
1	0.9	53.1	97.0	5148.6
1.05	0.925	76.0	86.4	6562.9
1.1	0.95	85.0	78.2	6644.0

Table 3.1: Delay, Total Power and Power-Delay-Product (PDP) of FTL_{35}, Trained at $V_{DD} = 0.9V$, and $C_0 = C_1 = 0.1fF$.

each high voltage pulse would increment the threshold voltage of a flash transistor by 20mV. This assumption will vary across flash transistors. As shown in Figure 3.11, the number of high voltage pulses needed to program a given FTL cell increases with an increase in the number of variables of the threshold function being implemented.

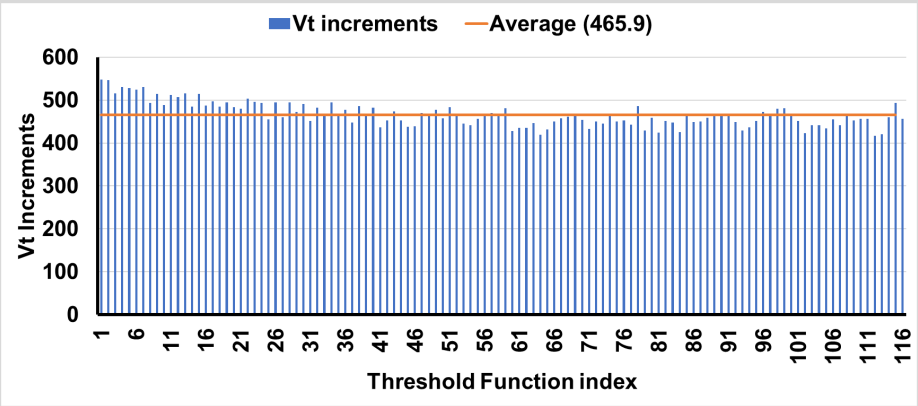


Figure 3.11: Number of High Voltage (HiV) Pulses Needed to Program the FTL Cells with All 117 Threshold Functions of up to 5 Inputs

3.6.7 *Experiments on Training for Robustness*

In this section, we present the results of Algorithms mPLA^{+}, and mPLA^{++} for training FTL cells taking into account parasitics and manufacturing variations. The test function $f_{35}(a,b,c,d,e) : (\boldsymbol{W}, T) = [3,3,2,1,1;8] = ab(c+de)$ was chosen for this evaluation as this function generated the most number of error-types (M_f=61) out of all the 117 threshold functions when Monte Carlo simulations were run on 20K training samples.

The first experiment consisted of training FTL_{35} using mPLA^{+} for various values of the capacitances C_1 and C_0, and for each solution, extracting the delay values. The results for this experiment are as shown in Table 3.2. There are two important observations to be made here. First, even though the weights of the inputs d and e are equal, the corresponding flash transistors (V_4 and V_5) may be assigned different threshold voltages. This is because mPLA^{+} compensated for the irregular layout parasitics of both the flash transistors using threshold voltages to realize equal weights. Second, the delay *improves* with increasing robustness, as discussed earlier in Section 3.5.3. This is because the separation between the lines $G_R = G_L - \Delta_L$ and $G_R = G_L + \Delta_R$ increases with an increase in C_1 and C_0. This increased separation results in a higher voltage difference at the inputs of the sense amplifier, which leads to a faster evaluation of the FTL cell.

The second experiment was aimed at validating mPLA^{++}. We used f_{35} as a test function. The first step is to create the *database* $\{\boldsymbol{VT}^{++}(f_{35})\}$. Algorithm mPLA^{++} was given f_{35} and $N_{MC} = 20K$ as inputs. The erroneous instances were grouped into $M_{f_{35}} = 61$ error-types. Algorithm mPLA^{++} generated $\{\boldsymbol{VT}^{++}(f_{35})\} = \{\boldsymbol{VT}^{+}(f^{e}_{35,1}), \cdots, \boldsymbol{VT}^{+}(f^{e}_{35,61})\}$.

Next, 100K new MC instances were generated and programmed first with $\boldsymbol{VT}^{+}(f_{35})$.

C_1, C_0	Average Vt Values (V) $(V_1, V_2, V_3, V_4, V_5; V_{l0}, V_{r0})$	Delay (ps)
0	0.64, 0.64, 0.66, 0.70, 0.72; 1.00, 0.58	224
0.01	0.60, 0.60, 0.64, 0.68, 0.70; 1.00, 0.50	178
0.02	0.60, 0.60, 0.64, 0.68, 0.70; 1.00, 0.50	178
0.05	0.60, 0.60, 0.64, 0.70, 0.70; 1.00, 0.50	172
0.1	0.56, 0.56, 0.60, 0.66, 0.66; 1.00, 0.42	163
0.15	0.52, 0.54, 0.58, 0.64, 0.64; 1.00, 0.34	154

Table 3.2: Delay Values of FTL_{35} = [3, 3, 2, 1, 1; 8], Trained for Robustness Using Various Capacitor Values (fF).

Among the erroneous instances, 99.96% of them were one of 61 error-types that were previously found. When each FTL cell in group $j, (1 \leq j \leq 61)$ was programmed with the threshold voltage set $\boldsymbol{VT}^+(f_{35,j})$, all the erroneous instances correctly computed f_{35}. The remaining .04% of the 100K were correctly programmed by executing mPLA^0 directly to the chip, starting with $\boldsymbol{VT}^+(f_{35})$. This required fewer than five iterations on average for the instances. Since f_{35} had the most failure types, all of the other 117 functions, which exhibit fewer failure types, would be equally easy to program correctly in the presence of variations. Thus, all errors caused by process variations were corrected, with the vast majority requiring a single, precomputed VT set and a small fraction requiring on-chip programming.

3.6.8 Robustness Against PVT Variations

Figure 3.12 shows the delay variations in delay of five sample threshold functions w.r.t process, temperature, and voltage variations. As expected, FTL cells are slowest in the SS corner and fastest in the FF corner. Furthermore, as the process moves from

Stage	Procedure	Yield (%)
Training (20K instances) M_f=61	mPLA^{++}	100%
Testing (100K instances)	mPLA^{++}	99.96%
	mPLA^{0} (On-chip)	100%

Table 3.3: Yield When mPLA^{++} and mPLA^{0} (On-Chip) Are Used for Programming Instances of FTL_{35} = [3, 3, 2, 1, 1; 8].

the SS corner to the FF corner, the delay improves, as expected. When the voltage increases from 0.81 V to 0.99 V, the delay improves. The FTL cells were also tested for reliability for the consumer temperature range of 0°C, 25°C, and 55°C. This result demonstrates that a $\boldsymbol{VT}^{+}(f)$ solution, generated using TT 0.9V 25°C, can reliably work with PVT variations.

3.6.9 Robustness Against VT Drift

Over the lifetime of an FTL cell, the charge stored in the gate of flash transistors eventually leaks into the channel due to the deterioration of thin oxide layer Degraeve *et al.* (1999), signal disturbances Bersuker *et al.* (2001), etc. This leakage effectively changes the VT of the flash transistors. By extension, it also changes the weights programmed on the FTL cell. Table 3.4 shows the effect of decreasing VT on the threshold functions programmed on the FTL cells. All 117 FTL cells operated correctly with a VT drift of up to $5mV$. Beyond $5mv$, some cells failed. However, after testing, their VTs can be reprogrammed to compensate for this drift. Furthermore, all the FTL cells that were selected by Genus when synthesizing ASIC designs (See Section 3.6.11) operated correctly with $20mV$ drift in VT.

Index	Threshold Function	Sum of Products Form
2	[1,1,0,0,0;1]	a+b
7	[2,1,1,0,0;2]	a+bc
18	[3,2,1,1,0;5]	ab+acd
37	[2,2,2,1,1;6]	abc+abde+acde
60	[4,3,2,2,1;8]	abc+abd+acd+abe+bcde

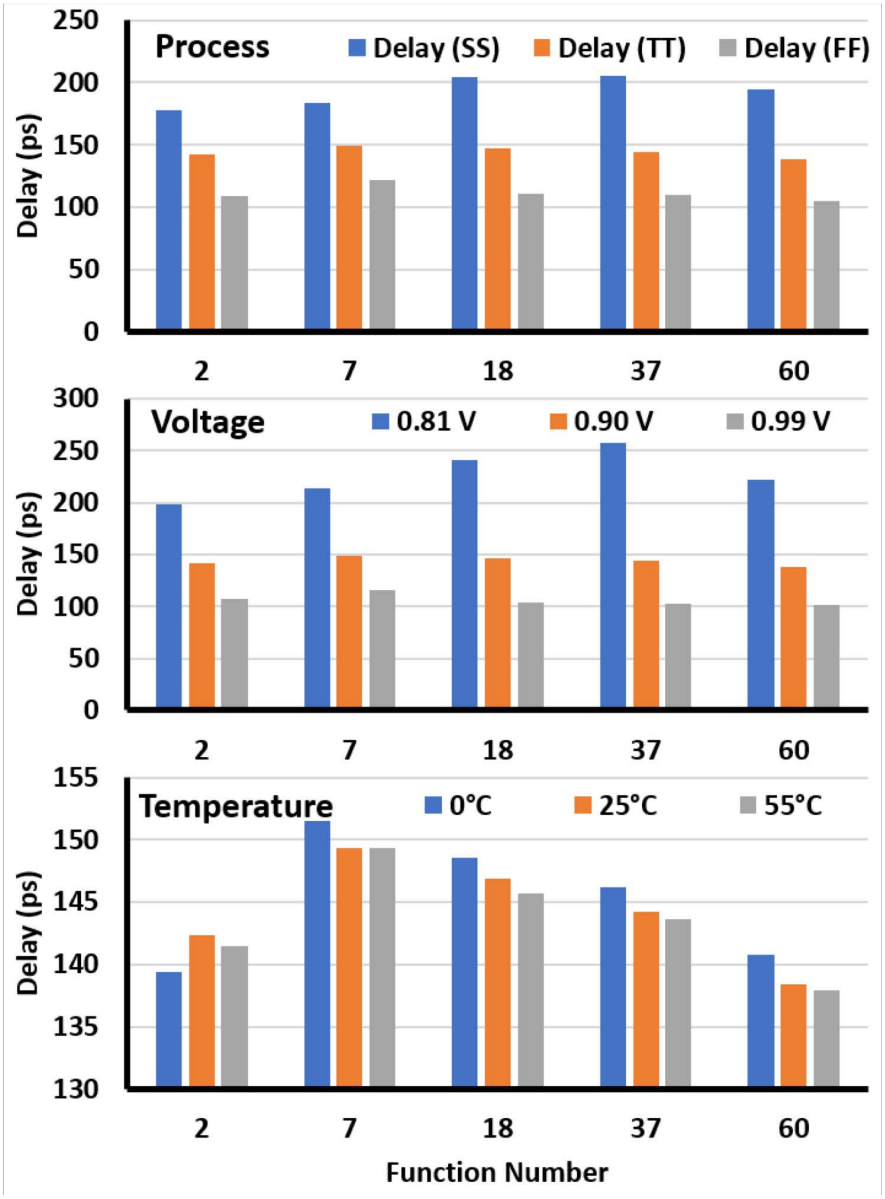


Figure 3.12: Delay of an FTL Cell for Threshold Functions, with the Process (SS, TT, FF), Voltage (0.81 V, 0.9 V, 0.99 V), and Temperature (0°C, 25°C, 55°C) variations.

Vt Drift (mV)	% FTL cells operated correctly
1	100
2	100
5	100
10	96.55

Table 3.4: Robustness Against V_T Drift for FTL Cells Programmed with All 117 Threshold Functions of up to 5 inputs.

3.6.10 Post-fabrication Timing Correction

The experiments described in Sections 3.6.7, 3.6.4 and 3.6.5 demonstrate the flexibility of FTL due to the possibility of configuring its function after fabrication. This characteristic can also be used to correct timing errors.

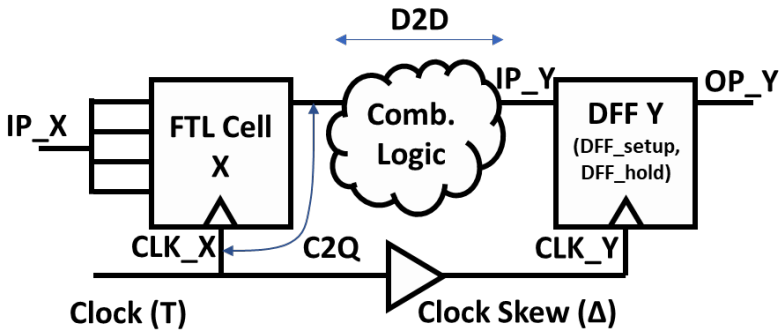


Figure 3.13: Datapath to Demonstrate Post-Fabrication Timing Corrections.

Figure 3.13 shows a small datapath that was constructed to demonstrate how to correct setup-time and hold-time violations after fabrication in an FTL design. The datapath consists of clock-to-Q (C2Q) delay, combinational delay (D2D), and DFF specifications for setup (DFF_setup) and hold (DFF_hold) times. The clock is skewed by an appropriate amount Δ, to generate either a setup-time or a hold-time

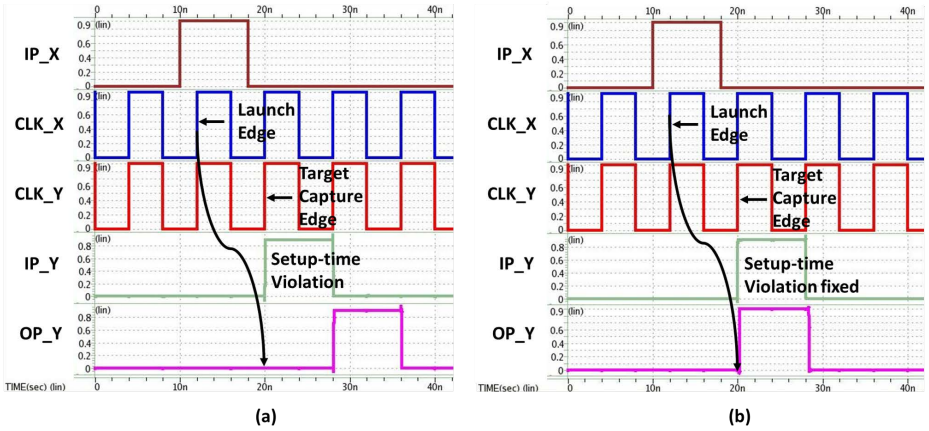


Figure 3.14: Post-Fabrication Setup-Time Correction Using an FTL cell.

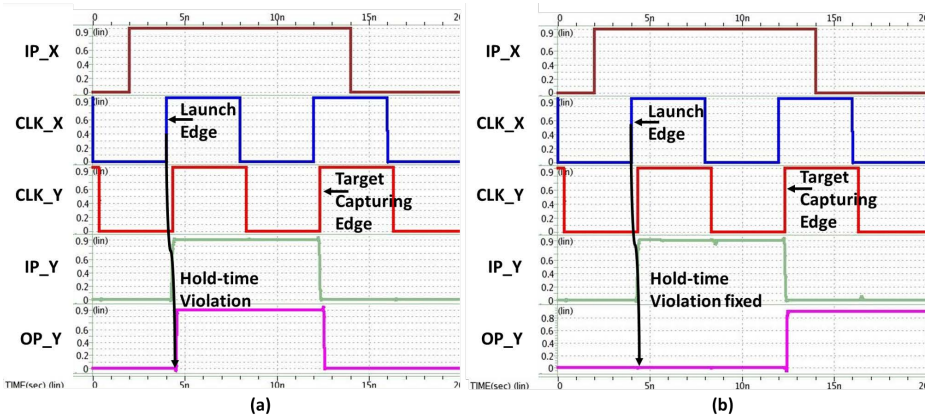


Figure 3.15: Post-Fabrication Hold-Time Correction Using an FTL cell.

violation. The violations are corrected by reprogramming the FTL cell to produce different C2Q values.

Figure 3.14(a) shows how the data launched from FTL X misses the target clock edge at DFF Y, thereby violating setup-time. Figure 3.14(b) shows that decreasing the C2Q of FTL X fixes the setup-time violation. Similarly, Figure 3.15(a) shows how the data launched from FTL X is captured by the target clock edge at DFF Y one cycle early, thereby violating hold-time. Figure 3.15(b) shows that increasing the C2Q of FTL X fixes the setup-time violation. Note that we can extend post-fabrication V_T adjustment to mitigate delay increases due to aging.

3.6.11 Delay Optimal Synthesis of ASICs with FTLs

		Conventional					FTL-integrated					Improvements			
Design	Freq. (MHz)	Std. Cells	DFF	Area (μm^2)	Power (mW)	Wire-length (μm)	Std. Cells	DFF/FTL	Area (μm^2)	Power (mW)	Wire-length (μm)	Area	Power	Wire-length	Prog. Time (μsec)
Mul	417	19536	343	51855	6.00	118906	11493	272/71	32339	4.64	88592	37.6%	22.7%	25.5%	204.3
Filter	406	53588	529	157482	36.26	436000	41711	281/248	107420	28.41	322400	31.8%	21.6%	26.1%	585.8
FPU	392	48992	1734	132655	27.82	484096	40937	1693/41	98879	24.14	406091	25.5%	13.2%	16.1%	113.2
FFT	667	156242	9614	443356	100.14	1405565	140650	9286/328	368509	86.62	1199160	16.9%	13.5%	14.7%	807.1
SHA	308	33204	2161	109170	15.95	396267	26511	2147/14	66001	13.23	343852	39.5%	17.1%	13.2%	47.9
Avg.				139290	25	425689			96468	21	343504	30.7%	17.7%	19.3%	351.7

Table 3.5: Improvement in Area, Power, and Wirelength Improvement in ASICs with FTL Integrated, over Conventional ASICs, Without Trading off Performance. Average Improvements Are Calculated Using the Geometric mean.

In this section, we show how commercial design tools can accommodate FTL cells in synthesis, and placement and routing. Five circuit blocks were synthesized using the 40nm TSMC standard cell library, which was augmented with FTLs to realize 117 positive forms of all 5-input threshold functions. This was done by creating one cell and making 117 copies and then determining the V_Ts of the flash transistors and signal assignments to realize each threshold function. Then each FTL standard cell was characterized in a conventional way. Only the positive forms of the threshold functions were included in the library to keep the increase in the library size to a minimum (about 7%) and to exploit the capability of Genus to recognize NPN equivalents of the cells (see below).

The ASIC benchmarks are: 1) 32-bit Wallace multiplier (Mul), 2) 28-bit FIR filter (FIR), 3) 64-bit floating-point unit multiplier, 4) 16-bit Fast Fourier Transform (FFT), and 5) 512-bit Secure Hash Algorithm (SHA). Designs were synthesized using Cadence Genus and then placed and routed using Cadence Innovus. Standard cell libraries for FTL cells were characterized using Synopsys HSPICE and generated in

Liberty format. Timing checks were performed using cross-corner analysis at {SS, 125C, 0.81V}, {TT, 25C, 0.9V} and {FF, 0C, 0.99V} corners. After placement and routing, the *select* cells and the FTL programming logic cells (see Figure 3.5 are *paired*. Then engineering change order (ECO) commands stitch the programming scan chain. Since the latter uses high-voltage nets, shielding nets are added to protect neighboring nets from high-voltage signals. Both versions of each ASIC were verified using Cadence Conformal.

The results of synthesis and P&R, summarized in Table 3.5, demonstrate significant improvements in the area (30.7%), power (17.7%), and wirelength (19.3%) averaged over the designs. These improvements include the *overhead* of the programming infrastructure described in Section 3.4, which was less than 5% in the worst case. Note that these *across-the-board* improvements were obtained under *delay-optimal synthesis*. This would not be the case for area-optimal synthesis.

Wherever it was beneficial to improve timing, Genus found and replaced *threshold logic cones* (not necessarily maximal fanout-free cones) driving DFFs with the appropriate FTL cell. This led to a reduction in the number of standard cells. It ranged from 10% to 42%. There are two causes for this reduction. First is the absorption of part of the fanin cone that is a threshold function driving the DFF into the FTL. This eliminates all those cells. A second source is the reduction of the subcircuit (e.g. C in Figure 3.2b) that *feeds* the fanin cone. The significant speed advantage of the FTL cell creates large positive slack at the outputs of the *feeder* subcircuit. Consequently, to meet timing, Genus re-synthesizes the feeder with slower logic. Standard logic primitives such as inverters, 2-input gates, 3-input gates, inverters, and even AOI/OAI gates are reduced, and the number of complex cells increases, reducing the total cell count.

The last column of Table 3.5 shows estimates of the time (i.e., number of pulses)

required to program the FTL cells, which increases linearly with the number of FTLs. Although the actual programming time will depend on the technology, it is expected to be on the order of microseconds Richter (2014).

Table 3.6 shows the run-time of Genus during synthesis, for all the ASIC designs. While the inclusion of all the 117 FTL cells increases the library size slightly (about 7%), FTL cells allow *faster timing closure* by generating positive slack. Table 3.6 also shows the peak memory usage of Genus during synthesis, for all the ASIC designs. The peak memory requirements are almost identical even after adding the 117 threshold functions in the library.

	Runtime(sec)		Peak memory (MB)	
	Conv.	FTL-integ.	Conv.	FTL-integ.
Multiplier	1451	636	1269	1292
Filter	2596	2893	1401	1439
FPU	2947	2724	1273	1262
FFT	3102	2653	1421	1416
SHA	1838	1790	1297	1292

Table 3.6: Runtime and Peak Memory Usage for the Synthesis of ASIC designs.

To demonstrate that Genus can recognize NPN equivalences of positive-form threshold functions, we selected a number of threshold functions and negated and permuted their inputs and negated their output. Table 3.7 shows the result of one of the more complex functions. The interpretation of Table 3.7 is as follows. Consider the threshold function $ab + ace + ade + bcd + acd$. The weight-threshold description is $[4, 3, 2, 2, 1; 7] = 4a + 3b + 2c + 2c + d \geq 7$, which is an FTL_{93}. When Genus found a sub-circuit with input negation, $\bar{a}b + \bar{a}\bar{c}e + \bar{a}de + bcd + a\bar{c}d$, , it replaced it with an FTL_{93} with $\bar{a}$ and $\bar{c}$ driving inputs a and c. The last row shows that Genus can

Threshold function	Verilog description of NPN equivalent	Synthesis result
ab+ace+ade +bcd+acd [4,3,2,2,1;7]	y <= ((4*a) + (3*b) + (2*c) + (2*d) + (1*e)) >= 7 ? 1:0;	FTL_93 (4,3,2,2,1;7)
	y <= ((4*(!a)) + (3*b) + (2*(!c)) + (2*d) + (1*e)) >= 7 ? 1:0;	FTL_93 (4,3,2,2,1;7) and two inverters for "a" and "c"
	y <= !((4*(!b)) + (3*c) + (2*(!d)) + (2*a) + (1*e)) >= 7 ? 1:0;	FTL_94 (4,3,2,2,1;6) and three inverters for "a", "c" and "e"

Table 3.7: Detection of NPN Equivalents of Threshold Functions Using a Library of 117 5-Input FTL cells.

detect output negation and maps it to a different cell FTL_{94} whose positive form is $[4,3,2,2,1;6] = 4a + 3b + 2c + 2c + d \geq 6$. In each case, the synthesis tool detected the threshold functions and their NPN equivalents, and added inverters as necessary, without using any additional standard logic gates such as AND, OR, etc.

The last experiment conducted was aimed at discovering what threshold functions would be detected if there were no area or delay constraints. Figure 3.16 shows all possible threshold functions that could be detected in the 32-bit Wallace tree multiplier. The multiplier has 343 DFFs. Excluding the 64 input DFFs, all 279 remaining DFFs and cones of logic feeding them were replaced by FTL cells, showing that complex multi-level logic circuits that are threshold functions frequently occur in logic circuits, and synthesis tools can recognize them.

3.7 Conclusion

In this chapter, we demonstrated that there could be substantial value in going beyond the traditional use of flash technology as memory and using it in CMOS logic.

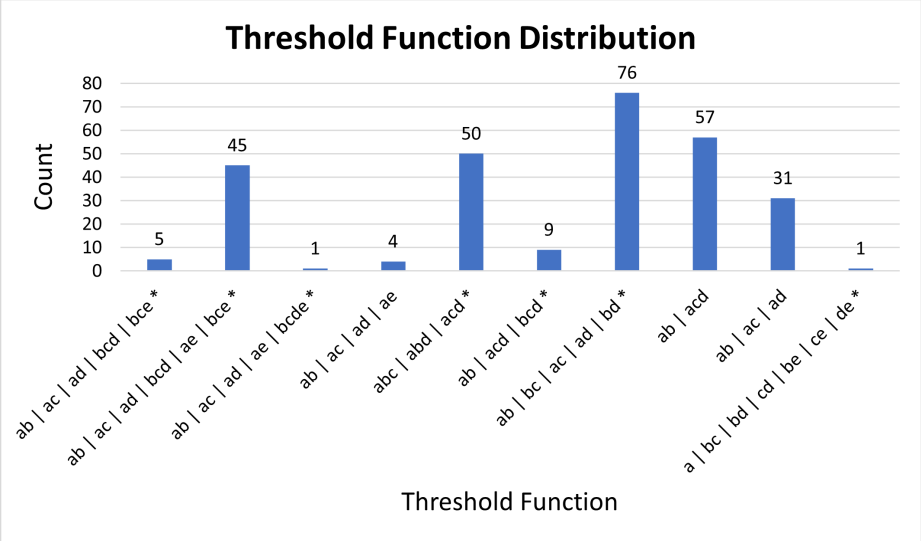


Figure 3.16: Distribution of Threshold Functions in 32-Bit Multiplier When Synthesized Using FTL Cells with Zero-Delay Zero-power.

Unlike many emerging memory technologies, flash technology is mature and compatible with CMOS fabrication. Using flash transistors in conjunction with CMOS transistors, we developed a design of a binary neuron, referred to as FTL, that can realize a large number of threshold functions in a single standard cell. We demonstrated several novel features of an FTL cell: (1) it is a *configurable standard cell*, whose function can be configured after fabrication; (2) the configuration is achieved by conventional techniques of tuning the threshold voltages of flash transistors with high fidelity; (3) its design could be optimized to make it very robust in the presence of circuit parasitics and to improve robustness also improves its performance; (4) the ability to tune its performance after fabrication provides a novel way to improve the yield in the presence of process variations and correct timing errors; (5) it was designed so that it can automatically be embedded within ASICs using commercial CAD tools, and resulting in significantly improved area and power while still operating at the maximum possible frequency.

Chapter 4

THRESHOLD LOGIC FPGA

4.1 IntroductionWagle and Vrudhula (2021)

Techniques for improving the power, performance (clock frequency) and area (PPA) of circuits to be implemented on FPGAs have matured over three decades Chen *et al.* (2006); Kuon *et al.* (2008). Interest in this topic has resurged due to the increasing use of FPGAs for implementing compute-intensive algorithms that arise in machine learning such as deep neural networks (DNNs) Zhou and Jiang (2015); Ma *et al.* (2017), augmented reality Guimarães *et al.* (2007) or virtual reality Fohl and Hemmer (2015). Broadly speaking, techniques for improving the PPA of FPGA implementations fall into two, inter-related approaches: (1) improving design mapping algorithms and (2) modifying the architecture of the basic logic element (BLE) (also referred to as Configurable Logic Block or CLB), which is most often implemented by a look-up table (LUT) circuit. Extensive optimizations of design mapping algorithms targetting general netlists have taken place over two decades, and are now part of design tools offered by FGPA manufacturers. However, the focus of improving design mapping algorithms has now shifted to a specific class of applications such as DNNs Moolchandani *et al.* (2021).

Research into alternate BLE architectures has received less attention, partly because there is little opportunity to improve the design of static CMOS logic circuits. Most of the variations in the design of the BLE have included changing the support set and the set of functions implemented by LUTs Ahmed and Rose (2004); Feng *et al.* (2018); Anderson *et al.* (2012). More recently, BLE architectures employing

various memristor-type devices have been proposed, such as RRAM, STT-MTJ and DWT Tang *et al.* (2016); Perricone *et al.* (2017). Although existing literature claims that these will result in ultra-compact and energy-efficient FPGAs, large-scale, commercial demonstration is still far away.

In this chapter, a different structure of a BLE is described, resulting in a new architecture for an FPGA, referred to here as *threshold logic FPGA* (TLFPGA). The main features of the new design are as follows. The BLE comprises a conventional LUT and a CMOS digital implementation of a threshold logic cell (TLC), commonly known as a perceptron or a binary neuron. Such a BLE provides ways to explore performance-area tradeoffs that are not present with conventional BLEs. This leads to a new design mapping algorithm that exploits the presence of conventional LUTs and TLCs, resulting in significant improvements in PPA.

As the BLEs in a TLFPGA are designed with conventional CMOS devices, TLFPGAs are compatible with existing design tools and can be manufactured today. A detailed evaluation of circuits mapped to FPGAs and TLFGPA is done, exploring BLEs with different sizes of LUTs and a fixed number of TLCs. The FPGAs and TLFPGAs were designed down to the layout level using the public domain FPGA design tools VPR Betz and Rose (1997), OpenFPGA Liu (2014), and Genus® for synthesis and Innovus® for placement and routing. Estimates for performance and power are based on simulation of netlists with all parasitics extracted from the layouts. The circuit benchmarks that were mapped to the FPGAs and TLFPGAs include the traditional set of ISCAS-85 as well as larger and more complex function blocks from OpenCores.

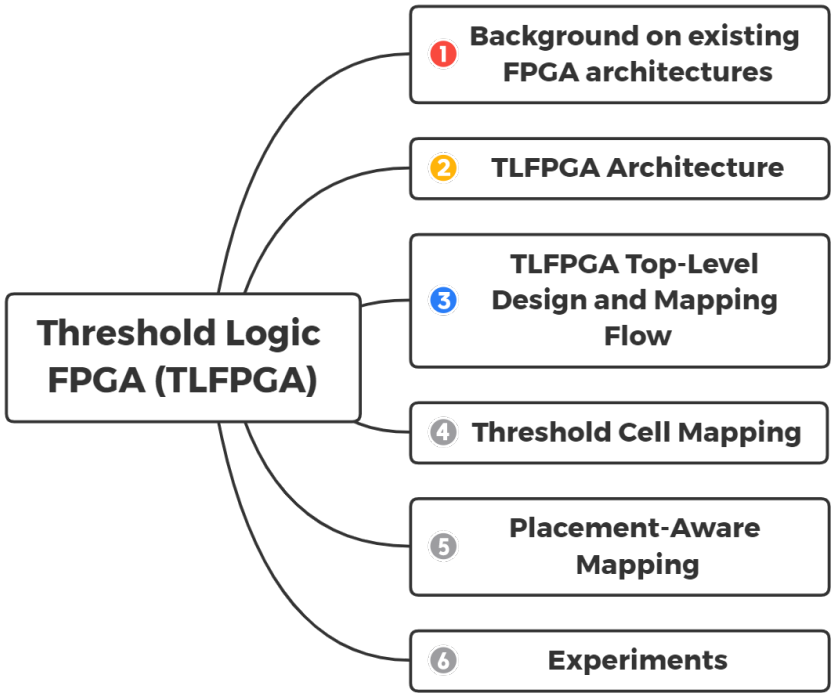


Figure 4.1: Organization of the Chapter

4.1.1 Organization of the Chapter

Section 4.2 briefly describes the published research for improving FPGA architectures and how the present work fits in this research. Section 4.3 then describes the TLFPGA architecture and compares LUT-n[1] for $n = 4, 5, 6, 7$ with a TLC-7 in terms of area, delay, and power. Section 4.4 describes the design flow that results in a complete layout of the FPGA and TLFPGA. Sections 4.5 and 4.6 describe the logic mapping algorithm. Experimental results using standard benchmark circuits and larger complex function blocks are presented in Section 4.7. Section 4.8 concludes this chapter.

[1]LUT-n refers to an LUT whose support set is n.

4.2 Related Work

Prior efforts to improve the PPA of FPGAs focused on the design mapping algorithms or re-architecting the BLE. Design mapping algorithms are central to automating FPGA design. One of the earliest is FlowMap Cong and Yuzheng Ding (1994), which showed that FPGA technology mapping minimizing logic-depth can be solved optimally in polynomial time. They reported up to a 7% reduction in the depth of mapped benchmark circuits. This was subsequently extended in the work by Cong et al. Cong and Ding (1994), in which estimated interconnect delays are used to minimize the delay. In the method described by Pan et al. Pan and Liu (1996), retiming is done during technology mapping to reduce area and power. PowerMap, proposed by Hao et al. Li *et al.* (2001), reduces the area and power without sacrificing performance by selectively applying depth-optimal LUT mapping for timing-critical paths and area-optimal LUT mapping for non-critical paths. Tang et al. Tang *et al.* (2005) proposed rewiring as a way to replace connections without altering the function, and demonstrated a substantial (17%) reduction in the number of LUTs when compared to several previously reported methods. The method proposed by Lin et al. Lin *et al.* (2006) performs clustering and mapping simultaneously, resulting in a significant performance improvement, albeit with an area penalty. Based on this partial review, it is clear that the domain of mapping algorithms have been extensively researched and as a result, FPGA manufacturers offer them as part of their suite of design tools.

Approaches to modifying the building blocks of an FPGA have mostly examined how different sizes of LUTs would impact performance and area, through empirical evaluations of benchmark circuits. Ahmed et al. Ahmed and Rose (2004) shows that an LUT-4 is better for reducing area and an LUT-6 is better for improving perfor-

mance. Anderson et al. Anderson *et al.* (2012) used an extended LUT-5 structure and showed a 9% reduction in the area with a 5% performance penalty. Feng et al. Feng *et al.* (2018) constructed an LUT-7 with reduced functionality, by interconnecting two LUT-4 circuits. This reduced the area by 9.5%, with a small (1.6%) reduction in performance. Instead of exploring the different variations of the LUTs, the approach described in this chapter replaces a few LUTs in a BLE with TLCs. Although a TLC supports a smaller set of functions as compared to an LUT with the same support set, TLCs are significantly smaller, faster, and consume much less power than an LUT Kulkarni *et al.* (2014); Wagle *et al.* (2018a). On a significant number of benchmark circuits, their use leads to simultaneous improvements in the PPA after mapping. On circuits where there is a small degradation in performance, the track-count reduces substantially.

Several researchers have also explored the use of emerging devices in FPGAs. Goncalves et al. Goncalves *et al.* (2013) presented a radiation-hardened FPGA, built using MTJs (Magnetic Tunnel Junctions). The work presented by Nukala et al. Nukala *et al.* (2012) used STT-MTJ (Spin Torque Transfer-Magnetic Tunnelling Junction) devices that were used to construct TLC-based logic elements. Their architecture allows the mapping of complex logic circuits into an extremely compact area, while also simultaneously delivering power improvements. Sampath et al. Sampath *et al.* (2015) used MTJs to implement both the LUT and the interconnect fabric to achieve area reductions. Kumar et al. Kumar *et al.* (2014) also used MTJs to implement LUTs, focusing on improving the delay and the energy-delay product. Cong et al. Cong and Xiao (2011) used memristor-based interconnects to reduce the area, power, and delay footprint of interconnects needed in an FPGA. Although these technologies are still under development and not yet viable commercially, existing literature provides compelling evidence that they have the potential for realizing ultra-compact

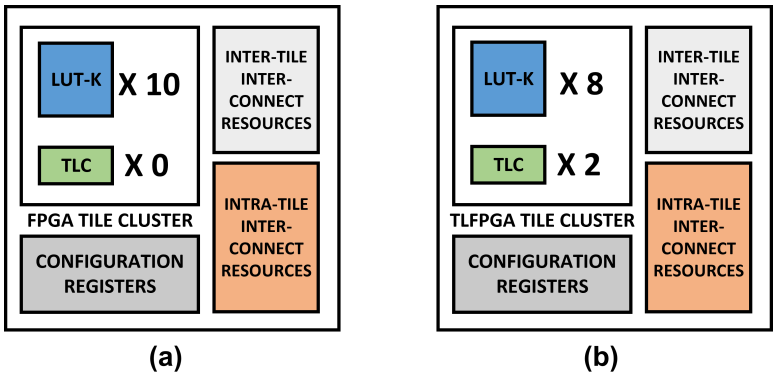


Figure 4.2: Sample Tile Structure for (A) FPGA and (B) TLFPGA. Two of the LUTs in (A) Have Been Replaced with TLCs in (b).

and energy-efficient FPGAs. The work presented in this chapter focuses on building FPGAs using TLCs that are constructed using conventional CMOS transistors and can therefore be fabricated easily using currently existing technologies.

4.3 TLFPGA Architecture

Figure 4.2 shows the structure of a standard FPGA tile and a TLFPGA tile. Each tile is a cluster of ten BLEs, with configuration registers and multiplexers (mux) that are used to program the BLEs. In the FPGA tile, all the BLEs are LUTs, whereas the BLEs in the TLFPGA tile are eight LUTs and two TLCs. The reason for having this specific distribution of LUTs and TLCs will be explained later in this section.

Operation of a TLC: The transistor-level structure of the TLC is shown in Figure 4.3. It consists of four components: a sense amplifier, a latch, a left input network, and a right input network. The circuit is designed to operate in two phases: Reset phase and Evaluation phase. In the reset phase, the clock signal CLK = 0. N5 and N6 are discharged in this phase, which in turn discharges all the paths from N1 and N2 to the ground line. The transistors M1 and M4 pull up the outputs of N1 and N2 to 1. In the evaluation phase, CLK transitions from $0 \rightarrow 1$. Assume the inputs arrived

before the clock started transitioning. As soon as CLK transitions, N5 and N6 start charging through the left and the right input network respectively. M13 and M14 are turned OFF, and both N5 and N6 will rise to 1. Without the loss of generality, assume that the left input network has a higher conductivity than the right input network during the evaluation phase. Subsequently, N5 will rise before N6, and turns M7 on. Prior to the evaluation, N1 and N2 were both 1. Therefore, M5 is active when M7 turns on. This discharges N1 through M5 and M7. The discharge of N1 stops the further discharge of N2 by turning off M6 and turning on M3. Consequently, the final values of the outputs are N1 = 0, N2 = 1, which resets the output latch. If the right input network had high conductivity, the result would have been N1 = 1, N2 = 0, which results in setting the latch. Notice the extra transistors M9 and M10. These are feedback transistors that help ensure that once the clock transition completes, further changes on the inputs will not affect the output.

The assignment of signals to the left and the right input networks is done to ensure that their conductivities are never equal over all the minterms of the function. This avoids incorrect evaluation by the sense amplifier. Consequently, transistors in the input networks are sized to minimum width to ensure that the input networks do not transition too quickly for the sense amplifier to evaluate. Transistors M2, M3, M5, M6, M7, M8, M13, and M14 are sized to 4 times the minimum width to reduce the overall clock to Q delay of a TLC. All the other transistors are sized to minimum width to keep the parasitics and the leakage power low.

Configuration registers are used to optionally invert the polarity of the inputs to the TLC. A configuration register $R_i = 0$ if input X_i is to appear in positive polarity, and $R_i = 1$ if X_i is to be complemented.

For the TLC to properly realize a threshold function, the predicate shown in Equation 2.1 has to be converted to a strict inequality, and the variables in Equation 2.1

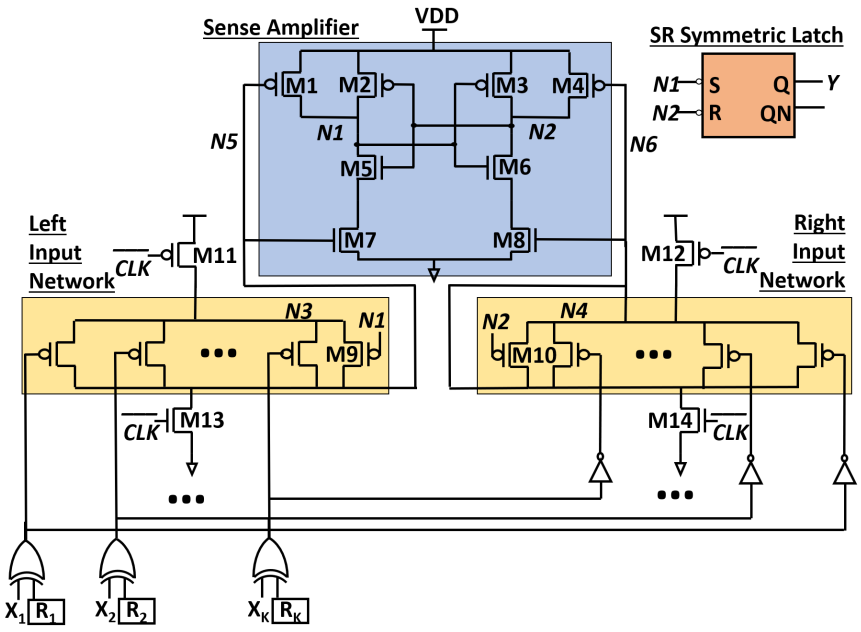


Figure 4.3: Threshold Logic Cell (TLC) Structure. the Sense Amplifier Detects the Difference in Conductivity of the Left and Right Input Networks and Sets the Outputs $N1$ and $N2$ accordingly.

have to be mapped to its inputs. Thus, Equation 2.1 is replaced with $\sum_{i=1}^{n} 2w_i x_i > 2T - 1$. As the signals driving the input networks are complementary, to realize this inequality the same number of literals representing a signal must appear in both networks. For example, consider $f(a, b, c) = a \vee bc \equiv 2a + b + c \geq 2 \equiv 4a + 2b + 2c > 3$. This is rewritten as $2a + b + c + 1 > 2(1 - a) + (1 - b) + (1 - c)$. Therefore the signals assigned to left input network in Figure 4.3 would be $X_1 = a$, $X_2 = a$, $X_3 = b$, $X_4 = c$, $X_5 = 1$, $X_6 = 1$, and $X_7 = 0$, and $R_i = 1$, for $i = 1, 2, \cdots 7$.

Note that this particular signal assignment just requires the seven XORs and configuration registers to program the TLCs and does not need any extra control mechanism. Besides, signal replication does not influence the required track-count in the TLFPGA. The configuration registers of a TLC can be programmed alongside the configuration registers of other LUTs in the FPGA. For this chapter, they were

programmed using a scan chain mechanism.

The advantages of the TLC have already been validated using 65nm technology in Yang *et al.* (2015); Kulkarni *et al.* (2016a). A TLC-based Wallace multiplier ASIC was fabricated, and then compared against a functionally equivalent, conventional standard cell implementation on the same die. Both simulation and chip measurement results demonstrated a 24% drop in area, 33% drop in dynamic power, 50% drop in leakage power, and 45% drop in wirelength, without any degradation in performance. Such significant improvements were also demonstrated for other designs in Kulkarni *et al.* (2016a).

TLC vs LUT: Tables 4.1a and 4.1b show a comparison of the delay and power of a TLC and several LUTs, all designed as standard cells in 40nm bulk CMOS and 28nm FDSOI.

A TLC-7 can realize all 4-input threshold functions and a subset of 5, 6, and 7-input threshold functions. These functions occur frequently in practical circuits. Figure 4.4 shows how *logic absorption* is performed. If part of a logic cone that drives the D-input of a flipflop happens to be a threshold function of some set of internal signals that can be realized by a TLC-7, then that part along with the flipflop can be replaced by a single TLC-7 cell. The remaining part of the cone that is not *absorbed* into a TLC is called the feeder circuit. Thus, a key step in mapping circuits to a TLFPGA is to identify threshold functions that are part of the logic cone driving flipflops. Note that logic absorption can always be done because the basic logic gates (e.g. AND, OR, NAND) are threshold functions.

TLC Tile vs LUT Tile: Subcircuits of numerous arithmetic and control circuits turn out to be threshold functions of some internal signals, and a vast majority have a support set of four variables. This means that each output of the feeder circuit (see Figure 4.4) could be mapped to an LUT. Hence, clustering four LUTs with a

BLE Type	Config. Regs	MUX/XOR	Delay (ps)	Power (μW)
LUT-4	16	15	549	35.0
LUT-5	32	31	613	42.4
LUT-6	64	63	881	64.9
LUT-7	128	127	916	124.7
TLC	**7**	**7**	**143.5**	**26.1**

(a) 40nm

BLE Type	Config. Regs	MUX/XOR	Delay (ps)	Power (μW)
LUT-4	16	15	220	33.2
LUT-5	32	31	226	64.0
LUT-6	64	63	294	125.0
LUT-7	128	127	331	248.0
TLC	**7**	**7**	**109**	**22.8**

(b) 28nm FDSOI

Table 4.1: Delay and Power for LUTs (With DFFs) and TLC. Compared to LUT-4, a TLC Is 3.8X Faster and 38% Lower Power in 40nm, and Is 2X Faster and 31% Lower Power in 28nm.

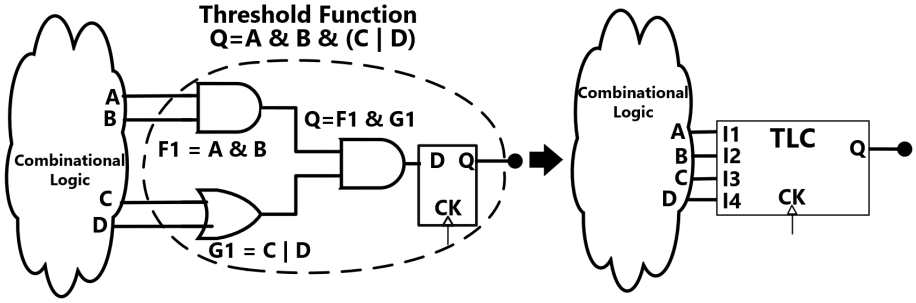


Figure 4.4: Logic Absorption. Part of the Logic Cone Driving the DFF Is a Threshold Function $abc \vee abd$. That Logic and the DFF Are Replaced by a Single TLC.

single TLC would allow the routing between the LUTs and the inputs of the TLC to remain within a single tile, thereby reducing the inter-tile routing. This leads to an improvement in the overall performance of the circuit. Since a cluster size of 5 is too small for practical circuits, and does not adequately provide for TLC to TLC intra-tile routing, a cluster size of 10 BLEs was chosen. Therefore, by extension, a cluster of 8 LUTs and 2 TLCs was chosen for a TLFPGA tile. Figure 4.5 depicts a typical cluster.

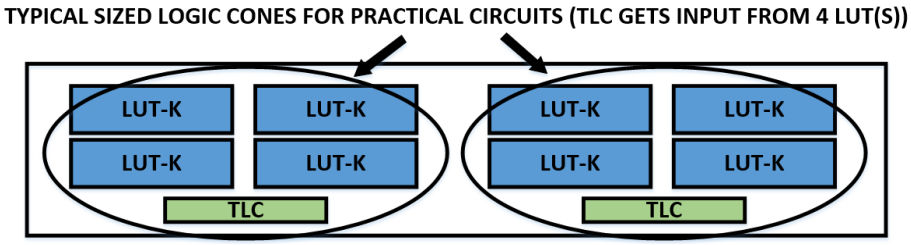


Figure 4.5: Cluster Size of 10 for TLFPGA; a TLC Typically Gets Inputs from Four Variables. the Number of LUTS in Each TLC Is Set to Four to Ensure Sufficient LUTs to Feed the TLCs Within a Single tile.

In order to study the effect of LUT sizes on the area, power, and performance of a TLFPGA, tiles with 4, 5, 6, and 7-input LUTs were designed. Since a tile in the TLFPGA contains fewer configuration registers and BLE multiplexers as compared to a standard LUT tile, it is significantly smaller than an LUT tile. This is demonstrated in both 40nm and 28nm technology using Table 4.2a and Table 4.2b respectively. At both individual cell level, and tile level, the TLFPGA results in a substantial reduction in the area over the FPGA. Later in Section 4.7, it is shown that the use of the TLFPGA architecture improves the PPA by reducing both the logic and routing resources during technology mapping.

K	LUT-K FPGA (μm^2)	LUT-K TLFPGA (μm^2)	Improvement (%)
4	572	524	8.3%
5	1144	982	14.2%
6	2088	1737	16.8%
7	4047	3305	18.4%

(a) 40nm

K	LUT-K FPGA (μm^2)	LUT-K TLFPGA (μm^2)	Improvement (%)
4	192	176	8.3%
5	272	236	13.2%
6	428	353	17.5%
7	780	617	20.9%

(b) 28nm FDSOI

Table 4.2: Tile Area of FPGA vs. TLFPGA in (A) 40nm and (B) 28nm. Replacement of a Large LUT with a Small TLC Helps Shrink the Tile Size. NOTE: These Numbers Do Not Include the Area of Inter-Tile Routing Resources, as They Are Subject to Change Based on the Number of tiles.

4.4 TLFPGA Design and Mapping Flow

This section describes two flows (sequence of steps) needed for the development of a TLFPGA. These are shown in Figures 4.6(a) and 4.6(b). The first, called FPGA Generation Flow, is used to design the TLFPGA, while the second, called FPGA Mapping Flow, is used to map the designs to the TLFPGA. Both these flows were created based on the OpenFPGA flow originally presented in Liu (2014). All the modifications done to the original flow are shown in yellow boxes in the figures. They have been explained later in this section.

Figure 4.6 (a) shows the automated flow used to generate an FPGA/TLFPGA architecture. The inputs to this flow are the parameters that define the FPGA, which

include tile count, tile interface, cluster size, and the number of tracks. OpenFPGA builds the Verilog description based on these parameters. After generating the Verilog, the desired number of LUTs in the Verilog of the tile is replaced with TLCs. After integrating all the blocks in the TLFPGA, standard synthesis, placement, and routing are performed on the resulting Verilog specification of the TLFPGA. Finally, the flow generates a parasitic extracted Verilog from the TLFPGA layout. The characteristics of this placed and routed TLFPGA are used to build the TLFPGA model for VPR (Versatile Packing, Placement, and Routing are public domain FPGA design tools Betz and Rose (1997)).

Figure 4.6 (b) shows the flow used to map a circuit to an FPGA/TLFPGA. It is a modified version of the design steps in OpenFPGALiu (2014), used to generate the bit-stream file for the TLFPGA. The bit-stream generator was modified to support the programming of TLCs. The sequence of steps is as follows. A behavioral netlist is synthesized to a logic gate network. Then *threshold cell mapping* (TCM) is performed on this netlist (Section 4.5), followed by LUT mapping using ABCBrayton and Mishchenko (2010), to generate a BLE mapped netlist. The TCM is a new algorithm that is specific to a TLFPGA and is not a part of the conventional FPGA FlowFarooq *et al.* (2012). VPR places and routes the BLE mapped netlist. Finally, OpenFPGA uses the final placement and routing results to generate the bit-stream file required to program the TLFPGA.

4.5 Threshold Cell Mapping

In this section, an algorithm to map a general logic network G onto a TLFPGA is presented, with the goal of improving one or more of area, power, and performance, without degrading any of them. The algorithm first maps subnetworks of G onto TLCs, followed by mapping the remaining parts of G to LUTs.

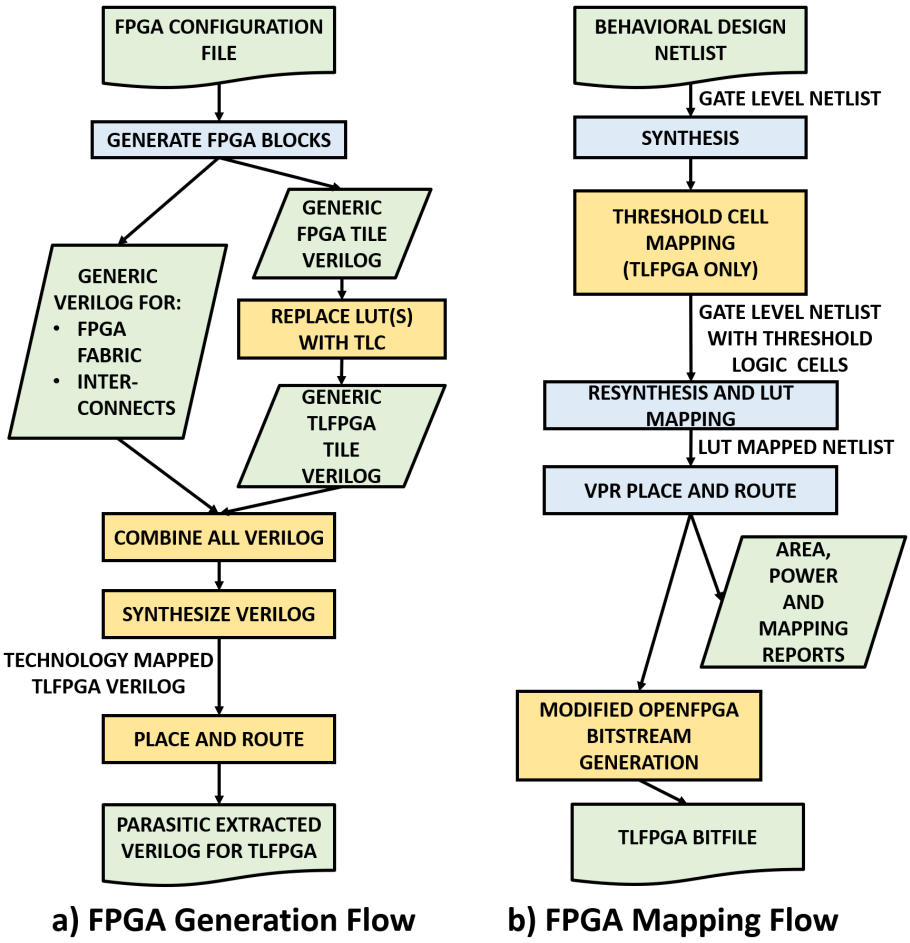


Figure 4.6: Modified OpenFPGA Flow for TLFPGA

4.5.1 Problem Definition

A given logic network is represented as a graph $G = (V, E)$, where $V = CL \cup FF$, is the set of nodes and $E = \{(s_i, D_i) | s_i \in V, D_i \subset V\}$ is the set of edges. CL is the set of combinational logic gates, FF is the set of flipflops, s_i is a source node of a connection, and D_i is the set of sink nodes of a connection. A *logic cone* $LC_i \subset CL$ associated with flipflop ff_i is a set of nodes such that there is a path from each node in LC_i to the node that connects to ff_i. $|LC_i|$ is number of inputs of LC_i.

A logic cone with p or fewer inputs and its associated flipflop can be realized by a

TLC with p inputs, if the Boolean function of the logic cone is a threshold function of its inputs. On the other hand, an LUT with q inputs can be used to realize the combinational part of a logic cone with q or fewer inputs, including or excluding the corresponding flipflop. In this way, the LUT is more general than a TLC. The technology mapping problem for the TLFPGA is to optimally map subgraphs of a given logic network to a network of TLCs and LUTs, with the goal of minimizing the latency, followed by minimizing the area, i.e. total number of BLEs.

4.5.2 Threshold Cell Mapping Algorithm (TCM)

In this section, an algorithm based on sound heuristics to map a logic network G onto a TLFPGA is presented, with the aim of improving the overall PPA compared to an equivalent network made up of only LUTs. The pseudo-code for this algorithm is shown in Algorithm 4.

Let $GF = (FF, E_D)$ denote an undirected graph on the set of flipflop nodes FF in G, and the set of edges $E_D = \{(ff_i, ff_j) | \exists k \ni LC_i \cap LC_k \neq \phi, LC_j \cap LC_k \neq \phi\}$. That is, two flipflops in GF have an edge if their logic cones overlap or if their logic cones overlap with another common logic cone. By definition GF is a an equivalence relation and hence it consists of a collection of components, each of which is a clique. Hence $GF = \{Q_1, Q_2, \cdots Q_K\}$, where Q_x denotes a clique on FF. Note that TCM can be performed on each clique independently as the logic cones do not overlap.

After performing static timing analysis Cortadella and Sapatnekar (2017) on G with a given technology and cell library, each flipflop in FF will be assigned a *slack* value, which the maximum difference between the *required arrival time* and the *actual arrival time* of a signal at a flipflop's input. The lower the value of the slack, the more *timing critical* is the flipflop. The flipflops in each clique $Q_x \in GF$ are sorted in increasing order of their slack values. Let SF denote the sorted set of flipflops.

We iterate through the elements of SF (See Algorithm 4 line 4) and apply an algorithm called TCM-single (Pseudo-code shown in Algorithm 5). For $ff_i \in SF$, LC_i is a single output logic network. A *cut* in LC_i is a set of edges, whose removal disconnects the output from all of its inputs. The procedure TCM-single iteratively selects a cut, and determines whether or not the logic function of the LC_i is a threshold function of its cut signals Kulkarni *et al.* (2016a); Kulkarni and Vrudhula (2016). In Kulkarni and Vrudhula (2016), a fast and general cut-enumeration algorithm based on network flow is presented. A much simpler and greedier procedure was later described in Kulkarni *et al.* (2016a). This new algorithm can be easily modified with constraints specific to the present application. In this chapter, the cut-enumeration algorithm in Kulkarni *et al.* (2016a) is applied to each LC_i corresponding to each component of GF, with the following constraints.

1. **Library constraint:** The cut enumeration is restricted to subcircuits that realize a given set of threshold functions, namely those whose support set is p or less (See Algorithm 5 line 5A). Since these checks are applied to unate Boolean functions, matching the function in a library using a function hash table is very fast. Note that TCM when applied to a logic cone LC always succeeds, because a single logic gate driving a flipflop is a valid threshold cut. The signal assignment method described in Section 4.3 determines the configuration bits needed to configure the TLC for the chosen threshold function. For creating a hash table of all the threshold functions a TLC can implement, the constraint shown in Equation 4.1 is used, which involves the number of input variables (n) and the total weight (W) and the threshold (T) (see Equation 2.1), the fact that the weights and threshold are integers, and the signal assignment to the left and right input networks are complementary.

$$n = max(W, 2T - W + 1) \tag{4.1}$$

We create a hash table of all threshold functions of a given number of variables based on the above constraint. Given n, the maximum value of T will be $T_{max} = \lceil \frac{n}{2} \rceil$. So, all values of T from 1 to T_{max} and all values of W are evaluated, distributing each over the number of variable from 1 to n. TCM is much faster with hash tables as opposed to the ILP solvers used in prior works.

2. **Cell fanout constraint:** Suppose a subcircuit (a feasible threshold function) of a logic cone LC_i associated with a flipflop ff_i is replaced by a TLC. Let g be a gate in the given subcircuit that has fanout to a logic cone associated with another flipflop, ff_j. Then the cone of the logic associated with g must be replicated within LC_j. This is shown in Figure 4.7. It is clear that logic replication reduces the fanout of the cell by one each time the cell's functionality is replicated. Since excessive logic replication tends to increase the overall LUT requirements, logic replication is restricted to only cells with a fanout of 2. (See Algorithm 5 line 5B)

3. **Complex gate constraint:** The purpose of this constraint is to avoid mapping TLCs to logic functions that are better implemented using LUTs. Industry-standard synthesis tools generally use complex gates (e.g., half-adders) to implement these functions. Although a complex gate C_i can be decomposed into simpler threshold functions, such a decomposition typically leads to a complex structure that contains a fanout of 2 or more. To avoid increasing fanouts in the circuit during TCM, complex gates are directly mapped to LUTs instead of decomposing them into threshold functions. (See Algorithm 5 line 5C)

We now describe the reasoning behind how subcircuits are replaced with LUTs

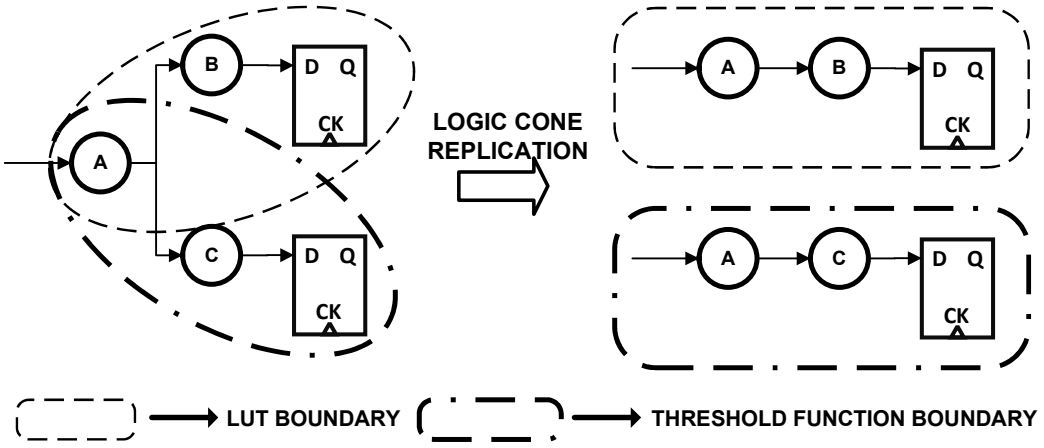


Figure 4.7: Logic Replication During Threshold Cell Mapping: Logic Cell a Is Replicated, so That It Can Be Mapped to a TLC (Represented Using Threshold Function boundary).

or TLCs. Let $d_{i,0}$ and $n_{i,0}$ denote the LUT depth (number of levels) and LUT area (number of LUTs) that would result when a logic cone LC_i associated with flipflop ff_i is mapped to LUTs only. Suppose when enumerating cuts in a logic cone LC_i, r feasible threshold cuts, $L_{i,j}, j = 1, 2, \cdots r$, are found. Let $\tilde{L}_{i,j}, j = 1, 2, \cdots r$ be the corresponding subcircuits that are in $LC_i - L_{i,j}, j = 1, 2, \cdots r$. That is, $\tilde{L}_{i,j}$ is the logic that is in the logic cone LC_i but not in the cut $L_{i,j}$. All the logic in $\tilde{L}_{i,j}$ is mapped to LUTs, each resulting in a corresponding logic depth $d_{i,j}, j = 1, 2, \cdots r$, and correspond area (number of LUTs) $n_{i,j}, j = 1, 2, \cdots r$.

For timing-critical flipflops ff_i (i.e., flipflops with zero slack), the cut $L_{i,j}$ is identified as a possible candidate for replacement by a TLC if $d_{i,j} = \min(d_{i,1}, d_{i,2} \cdots d_{i,r})$ (Algorithm 5 line 13-16). Similarly, for all other non-timing critical flipflops, the cut $L_{i,j}$ is a candidate if $n_{i,j} = \min(d = n_{i,1}, n_{i,2} \cdots n_{i,r})$ (Algorithm 5 line 18-21). The TCM-single algorithm maps the subcircuit $L_{i,j}$ in logic cone LC_i of a timing critical flipflop ff_i to a TLC only if $d_{i,j} < d_{i.0}$. For non timing critical flipflops ff_i, the criterion is $n_{i,j}| < n_{i,0}$. Note that this approach ensures that the logic depth is never increased by TCM-single, ensuring that the speed is never degraded.

Figure 4.8 shows an example of how mapping a TLC to a circuit helps reduce the number of LUTs that are needed. When a TLC is mapped to the circuit, which was originally mapped exclusively to LUTs, the mapping of the LUTs changes. As a result, the overall number of LUTs reduces by 1 when a TLC is mapped to the circuit in the example.

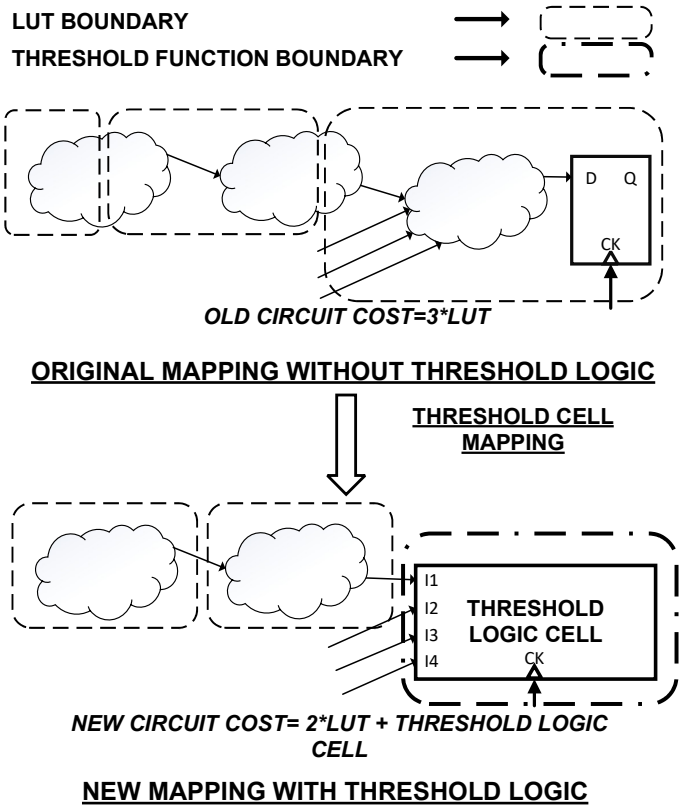


Figure 4.8: Reduction in Circuit Implementation Cost in a TLFPGA as Compared to an FPGA. the Cost of Mapping the Circuit to a TLFPGA Is Lower than the Cost of Mapping the Same Circuit to an FPGA.

TCM iterates through all the flipflops in G. During each iteration, it maps TLCs to flipflops only when they lead to an improvement in performance or area. An additional global constraint on the total number of TLCs that can be mapped to G is imposed. The purpose of this constraint is to improve the utilization of the tiles

of TLFPGA. Since the TLFPGA tile architecture has an LUT to TLC ratio of 4:1, maintaining that ratio globally allows the placement algorithm to proportionately pack the LUTs and TLCs in each tile, such that there are fewer unused BLEs in the tiles that were used. Maintaining the 4:1 LUT:TLC ratio, priority is given to the timing-critical flipflops for mapping to the TLCs (See Algorithm 4 line 3, 7-8). Once the TCM iterates through all the flipflops, G is transformed into a network of TLCs and logic gates. Finally, all the remaining logic gates that were not mapped to TLCs are mapped to LUTs. This step delivers the final output of TCM, which is a network of LUTs and TLCs. Once TCM is done, VPR takes G, and places and routes the design on the TLFPGA architecture. It must be noted that TCM is applicable to any arbitrary sized TLC, and not just a TLC of size 7.

Algorithm 4 TCM: Pseudo Code for Mapping All Suitable Subcircuits in a Netlist G to TLCs

Input: $\mathbf{G = (V, E)}$, **where** $\mathbf{V = CL \cup FF}$, $\mathbf{ff_i \in FF}$, $\mathbf{LC_i \subset CL}$ **and** $\mathbf{E = \{(s_i, D_i) | s_i \in V, D_i \subset V\}}$

Output: $\mathbf{G_{mapped}}$**:: G mapped to TLCs and LUTs**

1: $GF = (FF, E_D)$ where $E_D = \{(ff_i, ff_j) | \exists k \ni LC_i \cap LC_k \neq \phi, LC_j \cap LC_k \neq \phi\}$
2: **for** $Q_x \in GF$ **do**
3: SF_x = sort based on descending timing criticality(Q_x)
4: **for** $ff_i \in SF_x$ **do**
5: G = TCM-single(ff_i)
6: G_{mapped} = Perform LUT mapping on G
7: **if** LUT:TLC ratio in $G_{mapped} <$ 4:1 **then**
8: End TCM
9: **end if**
10: **end for**
11: **end for**

Algorithm 5 TCM-Single: Algorithm to Map a TLC to a Given Flipflop and Its Logic Cone

Input: $\mathbf{ff_i}$, **G**

Output: Updated G

LC_i = Input Logic Cone(ff_i)

LC_i^{LUT} = Map LUTs to LC_i

$d_{i,0}$ = logic depth (LC_i^{LUT})

$n_{i,0}$ = number of LUTs (LC_i^{LUT})

L_i = Enumerate logic cuts on LC_i originating from ff_i, with the following constraints:

1. **Library Constraint:** Subcircuit extracted by enumerated cut must be realizable using a TLC-7
2. **Fanout Constraint:** Cells can only be included in a cut if their fanout does not exceed 2
3. **Complex Cell Constraint:** Complex gates that can be better implemented using LUTs are not included in the cut.

for $L_{i,j} \in L_i$ **do**

  $\tilde{L}_{i,j} = LC_i - L_{i,j}$

  $\tilde{L}_{i,j}^{LUT}$ = Map LUTs to $\tilde{L}j, j$

  $d_{i,j}$ = logic depth ($\tilde{L}_{i,j}^{LUT}$)

  $n_{i,j}$ = number of LUTs ($\tilde{L}_{i,j}^{LUT}$)

end for

if ff_i is timing-critical **then**

  $L_i, d_i = L_{i,j}, d_{i,j}$, if $d_{i,j} ==$ min $(d_{i,1}, d_{i,2}, .., d_{i,r})$

  if $d_i \leq d_{i,0}$ **then**

    G = Replace ff_i and L_i in G with TLC

  end if

else

  $L_i, n_i = L_{i,j}, n_{i,j}$, if $n_{i,j} ==$ min $(n_{i,1}, n_{i,2}, .., n_{i,r})$

  if $n_i \leq n_{i,0}$ and $d_i \leq d_{i,0}$ **then**

    G = Replace ff_i and L_i in G with TLC

  end if

end if

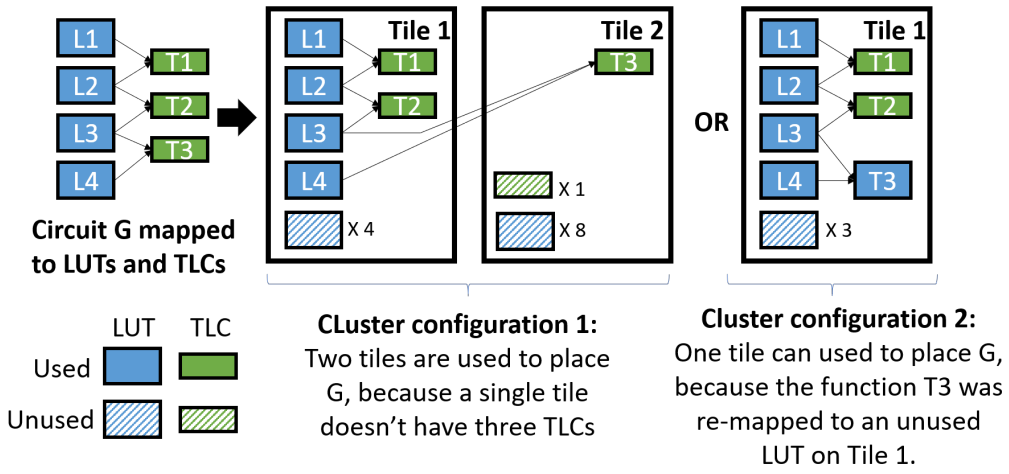


Figure 4.9: Placement-Aware Remapping: Clustering Algorithm in VPR Requires Two Tiles to Implement the Example Circuit G on a TLFPGA. Inter-Tile Routing Delay Will Be Added to the Inputs of T3, as Its Inputs Are Generated in Another Tile. Instead, If the Function of T3 Is Re-Mapped to an LUT, Then the Clustering Algorithm Only Requires One Tile to Implement G on the TLFPGA. This Improves Tile Utilization, and Removes Inter-Tile Routing Delay from the Inputs of T3 Thereby Improving the Overall performance.

TCM transforms a given netlist G into a network of LUTs and TLCs, which is then placed and routed by VPR. Before placing and routing G on TLFPGA, VPR creates multiple clusters (packing algorithm Betz and Rose (1997)) of the LUTs and TLCs of G, such that each cluster can be implemented using a single tile of TLFPGA. Here, the sub-goal of VPR is to minimize the number of clusters, without degrading the performance. Consider the example circuit G in Figure 4.9, made of 4 LUTs (L1, L2, L3, and L4) and 3 TLCs (T1, T2, and T3). Since each tile can only contain two TLCs, VPR packs T1 and T2 on one tile, and T3 on another tile. Since T3 is not in the same tile as its input logic cone, an inter-tile routing delay is introduced at the inputs of T3, which degrades the overall performance of G. A heuristic algorithm to

improve the clustering of LUTs and TLCs is used, as follows. In this algorithm, TLCs which receive inputs from other clusters are re-mapped to an LUT in one of those clusters that can accommodate an additional LUT. In the example, the function of T3 is remapped to an LUT, such that remapped G is now a network of 5 LUTs and 2 TLCs. VPR can now fit the remapped G in a single cluster, and remove the inter-tile routing from the inputs of T3. Similar to this example, the heuristic algorithm also remaps LUTs to TLCs, when LUTs get their inputs from other clusters which have available TLCs, and if the function of the LUT can be implemented using the TLC. This feature is important, as LUTs and TLCs can be used interchangeably during mapping to improve the overall PPA. To the best of our knowledge, most mapping algorithms designed for heterogeneous FPGA architectures do not re-map the logic of one basic logic element (BLE) with another (for example, LUT-6 re-mapped to LUT-4 or vice versa). Due to the lack of re-mapping algorithms, FPGA architectures are made homogeneous (one type of LUT) so that LUTs can be packed as tightly as possible, in the available tiles. The use of a placement-aware mapping algorithm, although very simple, enables tighter packing of tiles even when using a heterogeneous architecture.

4.7 Experimental Results

This section presents an evaluation of the TLFPGA architecture. Three separate sets of experiments were conducted: The first set of experiments (Subsection 4.7.2) was aimed at demonstrating how the PPA improvement depends on the ratio of the amount of logic absorbed into a TLC to the amount of logic that is left behind (see Figure 3). The higher the ratio, the greater the improvement in PPA. The motivation behind this experiment stems from the fact that a TLC can be viewed as an *edge triggered*, *multi-input flipflop* that realizes a threshold function of its inputs.

An equivalent static CMOS logic circuit would often require several levels of logic gates. For this reason, a set of combinational benchmark circuits (ISCAS-85) were selected, and pipeline stages were introduced by adding flipflops between levels of logic, and then applied the TCM procedure that replaces flipflops and portion of their cones with a single TLC. One to nine pipeline stages were introduced in each of ten selected ISCAS-85 circuits and the resulting 90 circuits were mapped to an FPGA and a TLFPGA. Evaluations were done using VPR.

The second set of experiments (Subsection 4.7.3) was carried out to explore how TLFPGA performs against a conventional FPGA design across several measures. For this, more realistic circuits from the OpenCores benchmark suite were chosen. These circuits were also pipelined until the logic depth of the critical path was between eight to twelve levels. These were determined to allow for a sufficient number of LUTs and TLCs to be used in the TCM algorithm. These evaluations were also done using VPR.

Finally, the third set of experiments (Subsection 4.7.4) was conducted using a small physically designed layout of a TLFPGA, to validate the evaluation results from VPR.

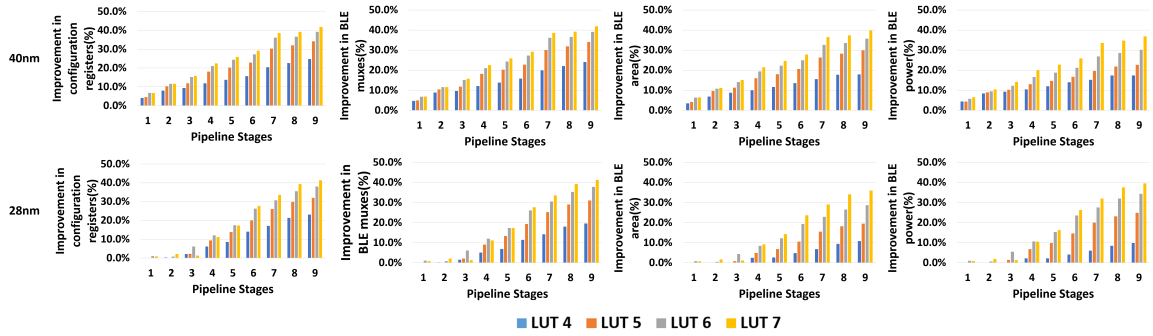


Figure 4.10: Average Percentage Reduction (Based on Geomean) in the Number of (A) BLE Configuration Registers, (B) Multiplexers, (C) Area and (D) Power in TLFPGA as Compared to FPGA for ISCAS-85 Circuits (Higher Is better).

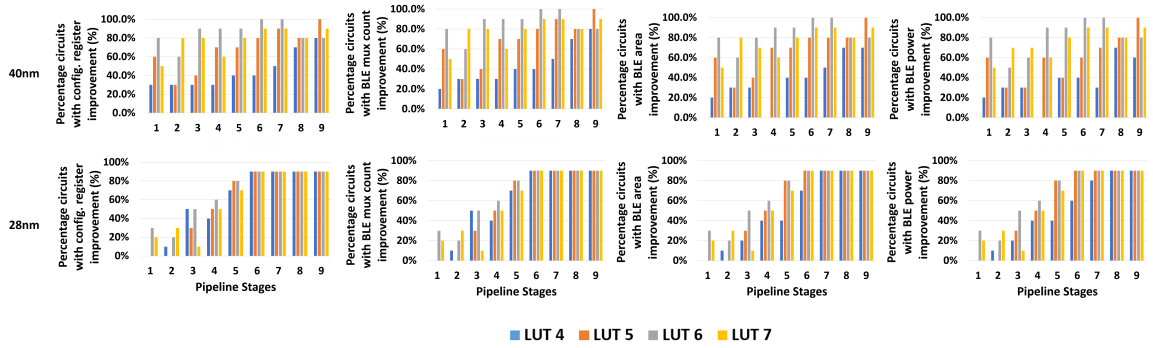


Figure 4.11: Fraction of Circuits That Showed Improvements in the Number of (A) BLE Configuration Registers, (B) Multiplexers, (C) Area and (D) Power in TLFPGA as Compared to FPGA for ISCAS-85 Circuits (Higher Is better).

4.7.1 Experimental Setup

The TLFPGA architecture was evaluated using ISCAS-85 and OpenCores benchmark circuits. Evaluations were conducted for two technologies: 40nm and 28nm FDSOI. The evaluation of the 28nm FDSOI from STMicroelectronics was done at $Slow/Slow$ $0.9V$ VDD $125°C$ simulation corner, while the 40nm-LP CMOS from TSMC was done at $Slow/Slow$ $0.81V$ VDD $125°C$. All measures such as performance, power, area, track-count, etc. are computed based on layout extracted netlists. Tool-related details for all the experiments are included in their respective subsections.

Note that during the experiments, TCM could find very few, if any, subcircuits with a support greater than four. So in order to realize all threshold functions of support four and a few larger ones, a TLC-7 was chosen for the experiments.

4.7.2 Impact of Pipelining on TLFPGA

VPR requires model files to place and route the FPGA architectures. To get the most reliable results using VPR, models of the FPGA and the TLFPGA were

generated based on their respective layout parasitics. The layouts of the tile structures were placed and routed using Cadence Innovus®.

Figure 4.10 shows the percentage improvement in the BLE-parameters of a TLFPGA as compared to an FPGA for both 40nm and 28nm. Improvements in the technology-independent parameters such as configuration registers and muxes indicate that the TLFPGA can potentially offer benefits at other technology nodes as well. This is further demonstrated as both 40nm and 28nm showed consistent improvements in technology-independent parameters. Minor differences in these parameters between the two technologies are due to the technology-dependent tradeoffs being made by employing heuristics. The results show that both technology-independent parameters and technology-dependent parameters such as area and power improve with increasing number of pipeline stages. This is a consequence of the TLC replacing a flipflop and part of a combinational logic cone.

Figure 4.10 also shows that for a circuit with a fixed number of pipeline stages, as the sizes of the LUTs increase, the advantages of using a TLFPGA over an FPGA also increase. This is due to the exponential increase in the BLE configuration registers and muxes as the size of the LUT increases. The number of configuration registers and XORs remains constant for the TLC. Hence, when the LUT requirements are reduced during technology mapping, there is a substantial reduction in the area and power requirements, which in turn results in significantly higher improvements when the TLFPGA is used. Figure 4.12 shows the improvement in the maximum frequency (performance) and track-count in the ISCAS-85 circuits when comparing an LUT-6 FPGA with an LUT-6 TLFPGA in 40nm and 28nm respectively. Generally, the routing algorithms in VPR either trade-off performance for track-count, or track-count for performance. This is because fewer tracks result in a drop in routing power, but the signals may have to take longer routes to fit within the limited number of tracks.

Conversely, a higher number of tracks may improve performance because signals can now utilize the available tracks to take the shortest path from their source to their destination. The simultaneous average improvement (Calculated using geomean) in both the parameters indicates that one parameter was not traded off for the other. This observation is further proven by observing that 54% of the circuits in 28nm showed a clear improvement in both frequency (5%) and track-count (2.3%), while only 4% of the circuits showed a simultaneous degradation in both the parameters. In 40nm, while the average performance of the TLFPGA is the same as an FPGA, there was a 6% improvement in the track-count. These results show that the performance and track-count improvements gained from the use of TLFPGA are consistent across technologies.

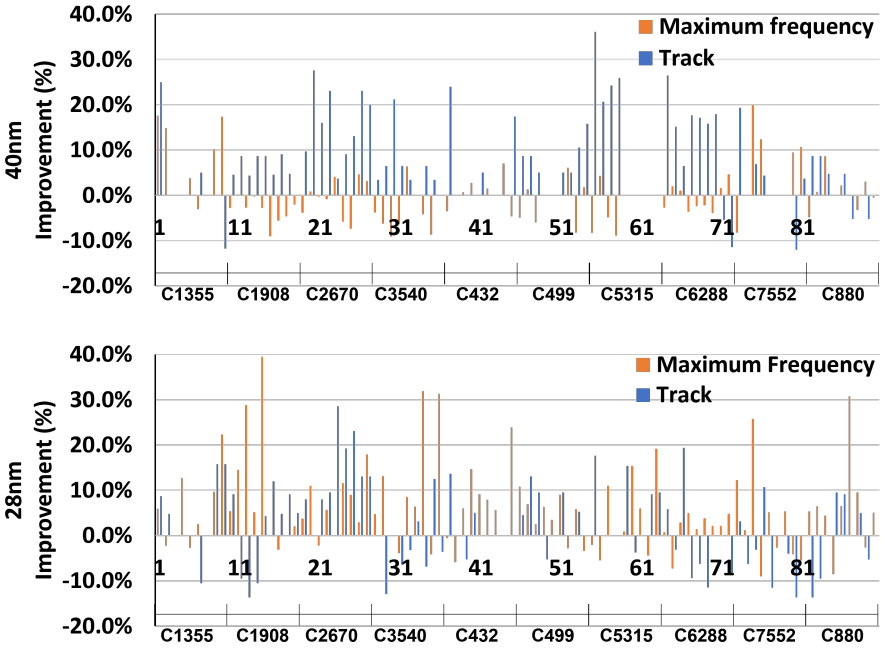


Figure 4.12: Improvements in Maximum Frequency and Track-Count for LUT-6 TLFPGA over LUT-6 FPGA for All ISCAS-85 Circuits in 40nm and 28nm. X-Axis Represents Circuit indices.

		% Improvement Solution 1		% Improvement Solution 2	
Circuit	Pipeline Stages	Tracks	Maximum Frequency	Tracks	Maximum Frequency
C432	9	17.4	-4.6	13.0	0.0
C499	2	17.4	-8.3	8.7	1.3
C6288	5	17.1	-2.4	11.4	2.3

Table 4.3: Track-Count and Frequency Trade-off Using VPR for Simultaneous Improvements of TLFPGA over FPGA. Solution 1 Is Generated Using the Default Settings of VPR, While Solution 2 Is Generating by Prioritizing the Maximum Frequency in VPR.

Circuit set 1	LUT-K		Config Regs		Mux Count		CLB Area		CLB Power		Critical Path		Freq		Track count		TLC:LUT		Stages	
	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7
64-bit Adder	18.1	24.7	15.1	22.0	15.0	21.9	12.0	18.8	7.0	14.2	2.3	10.9	2.4	12.2	5.0	0.0	25.0	25.0	(0,8)	(0,8)
64-bit Comparator	3.5	5.7	3.0	4.4	3.0	4.4	2.5	3.0	1.6	0.8	16.2	8.2	19.4	8.9	0.0	0.0	4.9	24.9	(0,10)	(0,10)
32-bit Multiplier	44.9	40.0	40.0	37.8	39.9	37.8	35.1	35.3	26.8	31.6	2.9	6.0	3.0	6.4	0.0	0.0	25.0	25.0	(0,10)	(0,10)
8-bit Hartley FFT	12.1	13.4	10.8	12.2	10.8	12.2	9.6	10.9	7.6	8.9	1.6	6.5	1.6	7.0	0.0	0.0	12.6	25.0	(0,2)	(0,2)
16-dit Divider	20.6	20.6	18.2	19.4	18.2	19.4	15.8	18.0	11.8	16.0	1.0	0.0	1.0	0.0	4.8	5.0	25.0	25.0	(0,6)	(0,6)
32-bit Filter	5.6	10.8	4.7	10.0	4.7	10.0	3.9	9.1	2.5	7.8	25.7	-9.0	34.6	-8.3	9.4	12.5	8.0	15.8	(6,6)	(6,6)
Geomean	**18.8**	**20.0**	**16.3**	**18.4**	**16.3**	**18.4**	**13.9**	**16.5**	**10.0**	**13.8**	**8.8**	**4.0**	**9.7**	**4.2**	**3.3**	**3.0**	**18.3**	**25.0**		

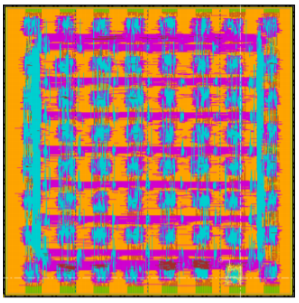
Circuit-set-2	LUT		Config Regs		Mux Count		CLB Area		CLB Power		Critical Path		Freq		Track count		TLC:LUT		Stages	
	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7
32-bit CRC	8.7	6.8	6.9	5.8	6.8	5.8	4.9	4.7	1.7	3.0	2.8	11.7	2.8	13.3	0.0	0.0	19.0	19.5	(1,0)	(1,0)
8-bit Encoder	16.7	20.0	11.4	17.8	11.3	17.8	5.7	15.3	-3.4	11.6	15.8	0.0	18.7	0.0	0.0	0.0	25.0	25.0	(1,0)	(1,0)
24-bit Logarithm	19.8	21.1	16.5	19.0	16.5	19.0	13.0	16.7	7.4	13.3	0.7	24.1	0.7	31.7	0.0	0.0	25.0	25.0	(6,0)	(6,0)
4-bit Cordic	22.4	19.9	20.0	18.8	19.9	18.8	17.4	17.6	13.3	15.7	1.4	3.5	1.4	3.6	0.0	0.0	25.0	24.8	(4,0)	(4,0)
Geomean	**17.0**	**17.1**	**13.8**	**15.5**	**13.8**	**15.5**	**10.4**	**13.7**	**5.0**	**11.1**	**5.4**	**10.3**	**5.7**	**11.5**	**0.0**	**0.0**	**25.0**	**25.0**		

Table 4.4: Percentage Improvements in TLFPGA as Compared to Standard FPGA in 40nm for OpenCores Circuits for LUT-6 (L-6) and LUT-7 (L-7) 50x50 TLFPGA; Results Are Extracted Using VPR. Models Used for VPR Are Based on the Tile Structures Placed and Routed Using Cadence Innovus®. Stages(x,y) Indicates That X Is the Original Number of Pipeline Stages in the Circuit, to Which Y Pipeline Stages Were Added. TLC:LUT Ratio Is Reported in percentage.

Circuit set 1	LUT-K		Config Regs		Mux Count		CLB Area		CLB Power		Critical Path		Freq		Track count		TLC:LUT		Stages	
	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7
64-bit Adder	18.1	18.1	15.2	15.2	15.1	15.1	9.8	9.8	13.1	57.4	8.3	8.3	9.0	9.0	0.0	0.0	25.0	25.0	(0,8)	(0,8)
64-bit Comparator	3.5	3.5	3.0	3.0	3.0	3.0	2.1	2.1	3.5	52.4	15.2	15.2	17.9	17.9	0.0	0.0	4.9	24.9	(0,10)	(0,10)
32-bit Multiplier	44.9	44.9	40.1	40.1	40.0	40.0	31.5	31.5	44.9	69.7	22.9	22.9	29.7	29.7	0.0	0.0	25.0	25.0	(0,10)	(0,10)
8-bit Hartley FFT	12.1	12.1	10.9	10.9	10.9	10.9	8.7	8.7	12.1	56.3	7.8	7.8	8.5	8.5	0.0	0.0	12.6	25.0	(0,2)	(0,2)
16-dit Divider	20.6	20.6	18.3	18.3	18.2	18.2	14.0	14.0	20.6	59.9	4.4	4.4	4.6	4.6	0.0	0.0	25.0	25.0	(0,6)	(0,6)
32-bit Filter	5.6	5.6	4.8	4.8	4.7	4.7	3.3	3.3	5.6	55.0	17.3	17.3	21.0	21.0	3.1	3.1	8.0	15.8	(6,6)	(6,6)
Geomean	**18.8**	**20.0**	**16.3**	**18.4**	**16.3**	**18.4**	**12.2**	**17.3**	**18.0**	**58.9**	**12.9**	**8.5**	**14.8**	**9.3**	**0.5**	**2.0**	**18.3**	**25.0**		

Circuit-set-2	LUT		Config Regs		Mux Count		CLB Area		CLB Power		Critical Path		Freq		Track count		TLC:LUT		Stages	
	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7	K=6	K=7
32-bit CRC	21.7	6.8	15.5	5.8	15.4	5.8	4.2	5.2	21.7	53.0	1.9	3.9	2.0	4.1	0.0	0.0	25.0	19.5	(1,0)	(1,0)
8-bit Encoder	16.7	20.0	11.4	17.8	11.3	17.8	1.7	16.4	16.7	59.6	3.3	0.0	3.5	0.0	0.0	0.0	25.0	25.0	(1,0)	(1,0)
24-bit Logarithm	19.8	21.1	16.5	19.0	16.5	19.0	10.6	17.7	19.8	60.1	6.7	9.6	7.2	10.7	0.0	3.8	25.0	25.0	(6,0)	(6,0)
4-bit Cordic	22.4	19.9	20.0	18.8	19.9	18.8	15.6	18.1	22.4	59.6	9.4	8.9	10.4	9.7	0.0	21.1	25.0	24.8	(4,0)	(4,0)
Geomean	**20.2**	**17.1**	**15.9**	**15.5**	**15.8**	**15.5**	**8.2**	**14.5**	**20.2**	**58.2**	**5.4**	**5.7**	**5.7**	**6.0**	**0.0**	**6.7**	**25.0**	**25.0**		

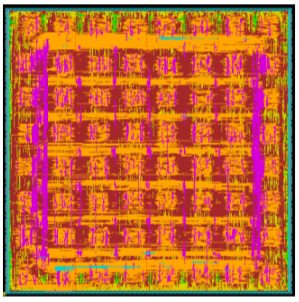
Table 4.5: Percentage Improvements in TLFPGA as Compared to Standard FPGA in 28nm for OpenCores Circuits for LUT-6 (L-6) and LUT-7 (L-7) 50x50 TLFPGA; Results Are Extracted Using VPR. Models Used for VPR Are Based on the Tile Structures Placed and Routed Using Cadence Innovus®. Stages(x,y) Indicates That X Is the Original Number of Pipeline Stages in the Circuit, to Which Y Pipeline Stages Were Added. TLC:LUT Ratio Is Reported in percentage.



(a)

Design	FPGA power (mW)	TLFPGA power (mW)	Improve (%)
C432	11.1	9.7	13.0 %
C499	12.6	11.2	11.1 %
C880	34.5	28.9	15.0 %

(b)



(c)

Design	FPGA power (mW)	TLFPGA power (mW)	Improve (%)
C432	5.9	5.01	15.1%
C499	6.4	5.63	11.9%
C880	15.6	13.7	12.18%

(d)

Figure 4.13: Physical Layout of an 8x8 Prototype TLFPGA in (A) 40nm and (C) 28nm Generated Using Cadence Innovus®; Verilog for the TLFPGA Was Generated Using Modified OpenFPGA Flow. Dynamic Power Reduction in Small Designs, When Mapped to the Physical Post-Layout Version of FPGA and TLFPGA in (B) 40nm and (D) 28nm.

Table 4.3 shows simultaneous improvements in track-count and frequency for a few circuits implemented in TLFPGA over FPGA, when the trade-offs are balanced using VPR. First, the circuits are mapped to an FPGA (baseline) and the TLFPGA (Solution 1) using the default settings in VPR. Solution 1 had a better track-count than the baseline but had a lower maximum frequency. In an attempt to balance the frequency to track-count trade-off, the circuits were re-mapped to the TLFPGA (Solution 2) by enabling all the timing-driven optimizations in VPR. Solution 2 is better than the baseline in terms of both frequency and track-count, which shows that the solutions generated by TLFPGA are superior to the FPGA, and are not a result of trade-offs.

To observe the number of ISCAS-85 circuits benefiting from the TLFPGA architecture, Figure 4.11 is plotted for both 40nm and 28nm technologies. It can be observed that as the number of pipeline stages increases, the number of circuits benefiting from the TLFPGA architecture also increases. This is because more pipeline stages create more opportunities to map the TLCs, subsequently allowing the circuit to draw the benefits of the TLFPGA architecture in a better way.

4.7.3 Results of Mapping Complex Circuits on TLFPGA

Although the ISCAS-85 circuits help us study the effect of pipelining on the TLFPGA, they may not properly represent the industrial designs that are currently available. Hence, benchmark circuits from OpenCores were additionally used to evaluate the TLFPGA. The benchmark circuits are divided into two groups: Circuit-set-1 and Circuit-set-2. Pipeline stages were added to the circuits in Circuit-set-1 so that their logic depth is in the range of 8-12 logic gates. This was done as most of the circuits in this group were combinational circuits. For circuits in Circuit-set-2, no additional pipeline stages were added. Tables 4.4 and 4.5 show the improvements in various parameters for the OpenCores circuits for both 40nm and 28nm respectively. This shows that the improvements gained from using the TLFPGA are consistent across technologies even for more practical circuits.

An additional experiment was also done to find the overhead in PPA due to the TLCs, if a circuit doesn't use any TLCs. The mean (geometric) degradation in the BLE area was found to be 2.7%, performance was 0.1%, BLE power was 0%, while the routing power improved by 1%. disabling the use of TLCs does not degrade the PPA in any appreciable degree. This shows that the overhead of TLCs is very small when not used but its benefits can be significant when used.

Parameter	FPGA Tile	TLFPGA Tile	% Reduction
Area	8001.3 μm^2	6935.3 μm^2	13.3%
Internal Power	17.9 μW	14.9μW	17.0%
Switching Power	32.1 μW	25.6μW	20.0%
Leakage Power	17.8 μW	14.8μW	16.8%
Total Power	53.5 μW	43.5μW	18.8%

(a) 40nm

Parameter	FPGA Tile	TLFPGA Tile	% Reduction
Area	1054.6 μm^2	870.5 μm^2	17.5%
Internal power	13.7 μW	11.2 μW	18.0%
Switching power	8.8 μW	7.4 μW	15.9%
Leakage power	10.4 μW	8.7 μW	16.6%
Total power	32.9 μW	27.3 μW	17.0%

(b) 28nm FDSOI

Table 4.6: Comparison of Area and Power of a Tile in 8x8 LUT-7 FPGA vs. a Tile in LUT-7 TLFPGA Tile with a Track-Width of 96 in X and Y Directions Using (A) 40nm and (B) 28nm Technology. Power Is Reported Using Static Power Analysis. Physical Layout of Tiles Include the Cells Required for Intra-Tile routing.

4.7.4 Validation of VPR Models by Physical Design

The results in this subsection are generated using the parasitic-extracted netlists from Cadence Innovus®. Physical layouts of 8x8 FPGA and TLFPGA were constructed using Cadence Genus® and Innovus® in both 40nm (Figure 4.13(a)) and 28nm (Figure 4.13(c)) technologies. The track-count for both the architectures was

Table 4.7: Comparison of Power (mW) for the Physical Layout of 8x8 LUT-7 FPGA vs. LUT-7 TLFPGA with a Track-Width of 96 in X and Y Directions Using (A) 40nm and (B) 28nm FDSOI Technology. Using TCM, Additional Power Reduction Is Expected. However, This Is Not Included During the Static Power analysis.

(a) 40nm	FPGA				TLFPGA				Drop %			
	Internal	Switching	Leak	Total	Internal	Switching	Leak	Total	Internal	Switching	Leak	Total
CLB	5.74	10.26	1.14	17.14	4.76	8.20	0.95	13.91	17.0%	20.0%	16.8%	18.8%
Switchbox	1.41	1.65	0.21	3.27	1.40	1.64	0.21	3.24	0.9%	0.6%	0.0%	0.7%
Crossbar	2.88	4.89	0.64	8.40	2.84	4.73	0.64	8.21	1.1%	3.1%	-0.1%	2.2%
IO Config	0.00	0.02	0.00	0.02	0.00	0.02	0.00	0.02	0.0%	0.0%	0.0%	0.0%
Total	10.02	16.82	1.98	28.82	9.00	14.60	1.79	25.39	10.2%	13.2%	9.6%	11.9%

(b) 28nm	FPGA				TLFPGA				Drop %			
	Internal	Switching	Leak	Total	Internal	Switching	Leak	Total	Internal	Switching	Leak	Total
CLB	1.72	5.47	0.75	7.94	1.32	4.73	0.62	6.67	23.4%	13.4%	17.9%	16.0%
Switchbox	1.12	3.56	0.50	5.19	1.08	3.44	0.50	5.02	3.9%	3.5%	0.1%	3.3%
Crossbar	5.00	2.79	2.38	10.17	4.99	2.68	2.38	10.06	0.1%	3.9%	0.0%	1.1%
IO Config	0.02	0.06	0.01	0.10	0.02	0.06	0.01	0.10	3.7%	-2.5%	-5.6%	-1.4%
Total	7.87	11.88	3.65	23.41	7.42	10.92	3.52	21.85	5.7%	8.1%	3.7%	6.6%

set to 96, and the cluster size was set to 10.

Tables 4.6a and 4.6b compare the power of a LUT-7 FPGA tile versus LUT-7 TLFPGA tile in both 40nm and 28nm respectively. Additionally, Tables 4.6a and 4.6b also compare the area in both the technologies. It can be observed that in both the technologies, the TLFPGA is substantially smaller than the FPGA tile, and consumes much lesser power. Note that these tables report the area and power numbers of the full tile, which includes inter-tile and intra-tile routing resources as well.

The improvements gained at the tile-level also extend to the full-scale architecture. Table 4.7 show the static power analysis comparison of both the FPGA and TLFPGA

	FPGA		TLFPGA	
	Count	Proportion	Count	Proportion
CLB	30800	12.2%	25816	10.5%
IO Config	5312	2.1%	5312	2.2%
Crossbar	172492	68.5%	172498	69.9%
Switchbox	43060	17.1%	43066	17.5%
Total	251664	100%	246692	100%

(a) 40nm

	FPGA		TLFPGA	
	Count	Proportion	Count	Proportion
CLB	51968	15.5%	40576	12.5%
IO Config	2688	0.8%	2688	0.8%
Crossbar	216896	64.5%	216896	66.8%
Switchbox	64507	19.2%	64478	19.9%
Total	336059	100.0%	324638	100.0%

Table 4.8: 28nm FDSOI

Table 4.9: Comparison of Instance Count for 8x8 LUT-7 FPGA vs. LUT-7 TLFPGA Physical Layout with a Track-Width of 96 in X and Y Directions Using (A) 40nm and (B) 28nm.

in both 40nm and 28nm respectively. These tables also indicate that the power consumed by a TLFPGA is lesser than the power consumed by an FPGA. This is because the components used to build the TLFPGA tiles consume lower power than their FPGA counterparts. Additional power drops, which are not part of the static power analysis, come through the use of the TCM algorithm, as described in Sec. 4.5.

	FPGA		TLFPGA	
	Area (μm^2)	Proportion	Area (μm^2)	Proportion
CLB	38970	8.0%	32352	6.8%
IO Config	14056	2.9%	14056	2.9%
Crossbar	380917	78.6%	380920	79.7%
Switchbox	50856	10.5%	50856	10.6%
Total	478099	100%	478184	100%

(a) 40nm

	FPGA		TLFPGA	
	Area (μm^2)	Proportion	Area (μm^2)	Proportion
CLB	66658	17.2%	53728	14.3%
IO Config	9651	2.5%	9651	2.6%
Crossbar	253620	65.4%	253620	67.6%
Switchbox	57985	14.9%	57966	15.5%
Total	387916	100.0%	374966	100.0%

(b) 28nm FDSOI

Table 4.10: Comparison of Area for 8x8 LUT-7 FPGA vs. LUT-7 TLFPGA Physical Layout with a Track-Width of 96 in X and Y Directions Using (A) 40nm and (B) 28nm technology.

Tables 4.8a and 4.8 show the instance count breakdown for the FPGA and the TLFPGA in both 40nm and 28nm respectively. Across both technologies, a clear drop in the instance count is observed. Note that the percentage improvement in overall

instance count may change depending on the number of instances that are present in the routing configuration. Furthermore, Tables 4.10a and 4.10b show the area breakdown for the FPGA and the TLFPGA in both 40nm and 28nm respectively. Observations from these tables show a clear improvement in the area at the full architecture level.

This evaluation was performed based on certain placement and routing settings. However, the results are subject to change based on the implementation strategy. This is because the area of the physical layout is dependent on both the cell type and the drive strengths of the cells. Therefore, if a certain architecture uses faster/slower interconnects or BLEs, the percentage improvements in the reported parameters will vary accordingly. These results are not meant to show the final area and power results, but instead to show the potential improvements that can be gained from using the TLFPGA architecture in a real environment. For the experimental results in this chapter, all the combinational loops that arise due to the interconnect structures of the FPGA and the TLFPGA were set as false paths during the placement and routing of these architectures. Minimum-sized buffers, inverters, and multiplexers were added for the interconnects to satisfy slew-rate constraints. Although this method is not ideal for a real FPGA design, it lets us generate and evaluate the FPGA architectures using a fully automated flow very quickly (Kim and Anderson (2015); Tang *et al.* (2019)). Due to the complexities in modeling accurate timing constraints and populating them in VPR models, the size of the FPGA and the TLFPGA was limited to 8x8. The small size of these architectures allowed us to avoid major timing violations. Due to the lack of timing constraints during the placement and routing of both these architectures, it is harder for the tool to set appropriate drive strengths for all the gates, and therefore the area is almost the same for both the FPGA and the TLFPGA post-layout.

A parasitic extracted Verilog netlist of an 8x8 FPGA and TLFPGA layout was used to map three of the ISCAS-85 circuits. A 300MHz clock was used to extract the power. Tables included in Figure 4.13(b) and 4.13(d) show that all the three circuits show improvement in overall dynamic power in both 40nm and 28nm respectively.

4.8 Conclusion

The architecture of an FPGA has remained almost the same since its inception. Therefore, for this chapter, radically different building blocks called TLCs were integrated alongside LUTs to enhance the area, power and performance of a traditional FPGA. TLCs are CMOS-compatible logic blocks that implement threshold functions. They have a negligible area footprint. When used in an FPGA, they offer significant PPA improvements, and yet incur negligible PPA degradation when not used. All the benefits in this chapter were evaluated using ISCAS circuits, as well as a few of the OpenCores circuits. Additionally, a few of the designs were also mapped to placed and routed versions of FPGA and TLFPGA to demonstrate real improvements. Since the PPA improvements depend on the amount of logic absorbed into a TLC to the amount of logic that is mapped to the LUTs, there may be scope for additional improvements if larger TLCs are used.

Chapter 5

QNN ASIC ACCELERATOR USING THRESHOLD GATES

5.1 Introduction

Deep neural networks (DNNs) have been remarkably successful in numerous applications of pattern recognition and data mining, including speech recognition, image classification, object recognition and detection, autonomous vehicles and robotics, recommendation systems, and many more. Consequently, they have become the dominant algorithmic framework in machine learning. DNNs are computationally and *energetically* intensive algorithms that perform billions of floating point multiply-accumulate operations on very large dimensional datasets, some involving tens of billions of parameters Fedus *et al.* (2022). Because *training* of large networks entails much greater computational effort and storage than inference, it is performed on high-performance servers with numerous CPU and GPU cores.

The energy cost and the environmental impact of training and inference of large DNNs are fast becoming unsustainable. Ref. Strubell *et al.* (2020) presents an analysis of the energy consumed in the training of several large NN models on various high-performance commercial multicore GPUs. For instance, training of the GPT-3 model with 175B parameters using NVidia's A100 with 1024 GPUs would consume 936 MWh of energy and take 34 days at a cost of $4.6M. Models even larger than the GPT-3 are being developed Fedus *et al.* (2022).

Improvements in energy efficiency of DNNs are not just limited to high-performance servers or desktop machines. The latest "*midrange*" and "*high-end*" mobile SoCs Kim *et al.* (2020); Liu *et al.* (2022) are being equipped with custom NN hardware accel-

erators to perform inference on mobile (e.g. mostly smartphones) and edge devices (e.g., IoT devices deployed in numerous spaces) for many of the above applications. The energy efficiency of inference on battery-powered devices is also of critical importance in terms of value to the customer and environmental impact. Given the rapid proliferation of ML techniques, several orders of magnitude improvement in energy efficiency over CPU-GPU implementations for training and inference of DNNs is needed for ML technology to be sustainable.

ASIC and FPGA are the two alternates to CPU-GPU implementations. Purely digital ASIC implementations are obtained by synthesis of custom logic blocks for specific operations such as 2-D convolution, inner product, matrix multiplication, and others Andri *et al.* (2017); Sze *et al.* (2017), each optimized for throughput and energy efficiency. Analog and mixed-signal solutions implement the inner product of fixed-weight matrices and input vectors by summing currents in crossbar arrays, where the weights are realized by various types of resistive elements (ReRAM Sun *et al.* (2018), MTJ Fan and Angizi (2017), Flash Guo *et al.* (2017b). Although orders of magnitude improvements in energy efficiency are reported, scalability and accuracy pose significant challenges.

Although ASICs have much greater energy efficiency and throughput than CPU-GPU implementations, they incur a very high cost because they are either restricted to specific NNs or due to the use of custom hardware blocks for each operation, the repertoire of operations implemented in hardware remains fixed. For both energy efficiency and throughput, FPGAs provide an intermediate solution to ASICs and CPU-GPU implementations Nurvitadhi *et al.* (2017). The reconfigurability of FPGAs, the availability of DSPs, and the fact that 90% of the operations are convolutions has made FPGAs ideal candidates for automatic mapping of NNs, with the goal of minimizing latency subject to energy constraints or vice versa Nurvitadhi

et al. (2017).

Regardless of whether it is an FPGA or ASIC implementation, throughput and energy efficiency can also be improved by modifying the structure of the NN. This includes tuning the hyper-parameters Wang *et al.* (2018, 2020), or modifying the network structure by removing the weights and connectionsLuo *et al.* (2017); Yang *et al.* (2017), or by altering the degree of quantizationWu *et al.* (2016); Wang *et al.* (2017). Another category of methods focuses on reducing the huge energy expenditure for moving data between the processor and off-chip memory, which is especially acute in NNs because of the large number of weights involved. The techniques to mitigate this include maximizing the reuse of data fetched from memory Yue *et al.* (2020); Luo *et al.* (2020), or transferring compressed data from the memory to the processor Ahanonu *et al.* (2018); Cheng *et al.* (2018).

Of the many available techniques for modifying NN structure, quantization remains the best way to achieve high energy efficiency and reduce computation time Trusov *et al.* (2021); Nagel *et al.* (2021), especially for energy-constrained systems. Quantization refers to using smaller bit-widths for the weights and/or the inputs during training, reducing them from 32-bit values to anywhere from 8-bit to 1-bit values. The term BNN refers to neural networks with 1-bit weights and inputs. Anything larger than that, but below full 32-bit precision is referred to as QNN. Quantization takes advantage of the fact that the accuracy of NNs is not very sensitive to substantial reductions in bit-widths until some critical value. Depending on the network, 4-bit to 1-bit QNNs for mobile applications provide an excellent tradeoff between energy efficiency and throughput versus accuracy Nagel *et al.* (2021).

This chapter presents a new ASIC design, called TULIP, for improving the energy efficiency of QNNs when performing inference. TULIP's unique features are summarized below.

1. TULIP is a scalable SIMD machine that consists of a collection of concurrently executing processing elements (PEs). The architecture of the TULIP-PE is radically different from PEs used in any other QNN accelerator Andri *et al.* (2017); Moons *et al.* (2018); Nakahara *et al.* (2016); Sun *et al.* (2018). It consists of a small network of *binary neurons*, referred to as *standard cell neuron* (SCN), each with a small, fixed fanin.

2. An SCN is a *clocked logic cell* that computes a threshold function of its inputs, on a clock edge. It is a *mixed-signal* circuit, whose inputs and outputs are logic signals but internally it computes the inner-product and threshold operation of a neuron, i.e. $f(x_1, \cdots, x_n | w_1, \cdots, w_n, T) = \sum_i^n w_i x_i \geq T$. Implemented as a standard cell, and after optimized for robustness and accounting for process variations, it is just a little larger than a conventional D-type flip-flop Wagle *et al.* (2022b). The SCNs in a TULIP-PE can be configured at run-time to execute all the operations of a QNN, namely the accumulation of partial sums, comparison, max-pooling, and RELU. Consequently, only a single processing element is required to implement all the operations in a QNN, and switching between operations is accomplished by supplying an appropriate set of logic signals to its inputs, which incurs no extra overhead in terms of area, power, or delay.

3. Unlike conventional PEs that are designed to operate at maximum bit-width (determined at design-time), the bit precision of TULIP-PEs can be changed within a single cycle without incurring a delay or energy penalty. This characteristic also enables making trade-offs between energy efficiency and accuracy at run-time.

4. Against the state-of-the-art MAC units used in QNN accelerators Andri *et al.*

(2017), the TULIP-PE is $\approx 16X$ smaller and consumes $125X$ less power. Although it is $9.6X$ slower, this is compensated by replicating 16 PEs and operating TULIP in a SIMD mode, executing multiple workloads in parallel that share inputs, which reduces the need to repeatedly fetch data from off-chip memory.

5. Since the SCNs in the TULIP-PE have limited fanin, much larger inner product calculations have to first be decomposed into smaller bit-width operations and then scheduled on the TULIP-PEs. For this, a novel *routing-aware* resource-constrained scheduling algorithm is presented that maps the nodes of a QNN onto TULIP-PEs.

6. The combined effect of the low area of TULIP-PE, the uniformity of the computation at the individual node and network levels, and the mapping algorithm results in an improvement of up to $50X$ in energy efficiency for QNNs over a MAC based design for the same area and performance of a MAC based design.

Note that a preliminary version of this work appears in Wagle *et al.* (2020a). This chapter includes substantial changes to architecture and extends support for a wide variety of QNNs while also significantly improving energy efficiency. In addition, extensive evaluations of TULIP on a wide variety of neural networks are included.

5.2 Overview of the Paper

Quantized Neural Networks (QNNs) are CNNs whose 32-bit input and weight values are converted into 8-bit to 1-bit values. In general, QNNs are directed graphs, where the vertices represent N-bit operations from a repertoire of operations: addition, comparison, multiplication, pooling, ReLU, and logic. We introduce a new SIMD architecture TULIP for accelerating QNNs. This architecture consists of a set of processing elements called TULIP-PEs, which are constructed using binary neu-

rons. Each TULIP-PE can execute a K-bit primitive operation (namely addition, comparison, or logic) at any given time. Since any QNN operation can be decomposed into a dataflow graph of primitive operations, the core of the chapter deals with scheduling a general directed dataflow graph of primitives on the TULIP-PE architecture. The first part of this chapter deals with the top-level architecture of the TULIP and the hardware architecture of its TULIP-PEs, while the second part of the chapter provides details of how computation is done on TULIP-PEs.

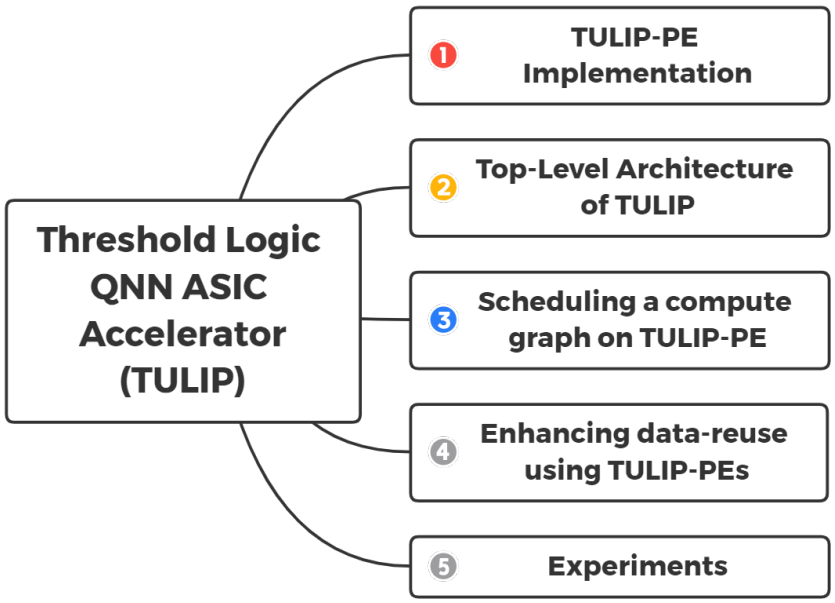


Figure 5.1: Organization of the Chapter

Sections 5.3, and 5.4 describe the architecture of the TULIP-PE and the top-level architecture of TULIP respectively. Section 5.5 then presents the scheduling algorithm needed to execute each node of the QNN on a TULIP-PE. Section 5.6 then describes how the small size of the TULIP-PEs enables us to deploy a number of them in the same space as a conventional processing unit, thereby enabling better weight reuse. Finally, Section 5.7 presents both quantitative and qualitative evalu-

ation of TULIP-PEs and the TULIP architecture against equivalent state-of-the-art architectures.

5.3 TULIP-PE Implementation

This section first describes the hardware architecture of TULIP-PE and the motivation behind its design. Then, it describes how primitive operations (K-bit addition, comparison, and logic) are mapped to TULIP-PE, and also how the larger operations of a QNN such as addition, multiplication, etc., can be decomposed into primitive operations.

5.3.1 Hardware Architecture of TULIP-PE

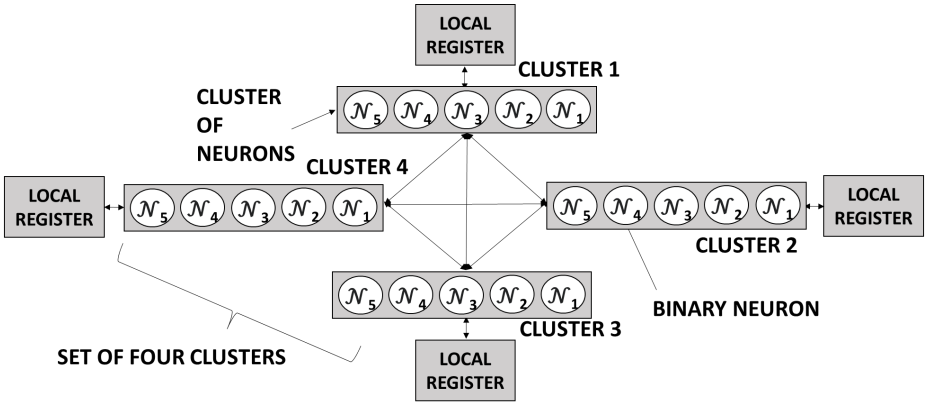


Figure 5.2: Architecture of a TULIP-PE, Consisting of Four Clusters and Four Local Registers. Each Cluster Contains K Neurons (K=5).

A QNN consists of several layers, usually, they are convolution, activation, pooling layers, etc. Conventionally when designing PEs, the focus is on efficiently accelerating the convolution layers as they make up for the largest share of the total computation time and energy. As a result, a typical PE in a QNN accelerator would consist of a multiply-and-accumulate (MAC) unit, that computes the weighted sum of inputs of

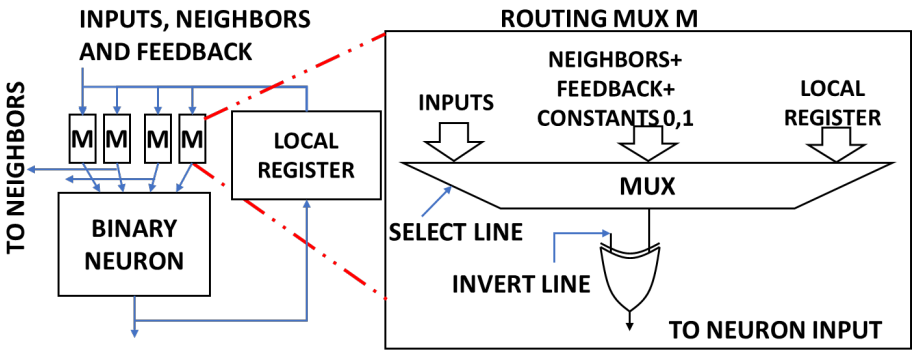


Figure 5.3: the Hardware Neuron and Its Connections with Inputs, Local Registers, and Other neurons.

a convolution layer.

The architecture of a TULIP-PE is designed much differently than a MAC unit, to enable the following features:

1. The TULIP-PE is designed to be over an order of magnitude smaller than a MAC unit, i.e. multiple TULIP-PEs can fit in the same area as a single MAC unit. When multiple TULIP-PEs run in parallel and share common inputs, they improve data reuse, and reduce off-chip memory usage.

2. It is designed to support all the layers of a QNN and not just the convolution layers. As a result, dedicated hardware blocks for other layers like pooling and activation are no longer needed in the hardware architecture.

3. Its design enables bit-level reconfigurability. This means that the bit-width of the hardware can be controlled depending on the operation, without any over-provisioning; unlike MAC units that are constructed to support maximum bit-width and avoid bit overflow of operations.

In order to understand how the TULIP-PE enables the above features, it is first necessary to describe its hardware architecture. A TULIP-PE (Figure 5.2) contains

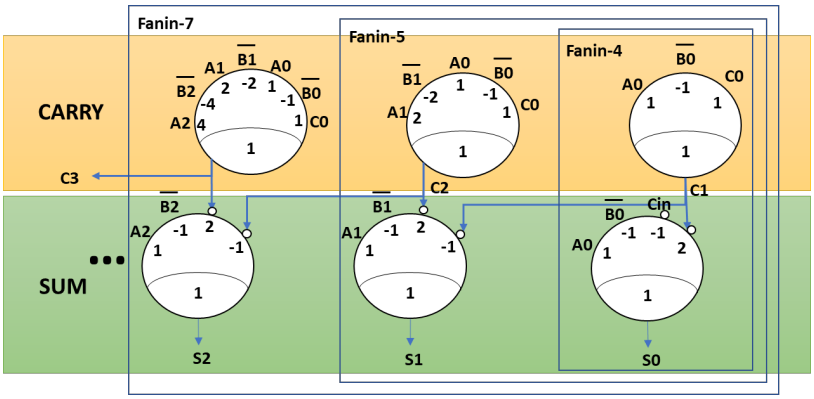


Figure 5.4: 3-Bit Carry Lookahead Adder Using Binary Neurons That Add Two 3-Bit Numbers A and B.

four clusters, each cluster containing K neurons each (Reasons explained in Section 5.3.2). The neurons in each cluster are labeled $\mathcal{N}_\kappa$, where κ is the index of the neuron in a cluster. As shown in Figure 5.3, each neuron is connected to external inputs, its 16-bit local register (latch-based), and its neighboring neurons using multiplexers. Inter-neuron communication is implemented using multiplexers. Each neuron shares its output and its inputs with other neurons.

5.3.2 Mapping Primitive Operations to TULIP-PE

The TULIP-PE supports all the layers in a QNN by decomposing the layer's operations (such as multiplication, ReLU, etc.) into primitive operations and then computing them sequentially. Therefore, the construction of TULIP-PE is based on optimizing energy efficiency and performance for primitive operations, which are in-turn implemented using threshold functions as follows.

1. **K-bit Addition ($ADDK(A, B, C_0)$)**

   Let C_{i+1} denote the carryout of stage i, $i \geq 0$. A carry *lookahead* of size i means that C_i is expressed as a function of A_j, B_j and C_0, $1 \leq j \leq i-1$. While the

carryout function is a threshold function regardless of the size of the lookahead, the sum function S_i is a threshold function of carryout C_{i+1} and carryin C_i. Hence the i^{th} carry bit and sum bits can be expressed as threshold functions as follows:

$$C_i \equiv C_0 + \sum_{j=0}^{i-1} 2^j A_j - \sum_{j=0}^{i-1} 2^j \overline{B_j} \geq 1 \ \forall i \in [1, .., K] \tag{5.1}$$

$$S_i \equiv A_i + \overline{2C_{i+1}} - \overline{B_i} - \overline{C_i} \geq 1 \ \forall i \in [0, .., K-1] \tag{5.2}$$

2. **K-bit Comparison ($COMPK(A, B, C)$)**

   Given two K-bit numbers A and B, the predicate $A > B$ can be computed using a K-bit comparator that uses threshold functions. If $A = B$, the value C (which represents the previous result; default = 0) propagates to the output. It can be represented using the following threshold function:

$$Y \equiv C + \sum_{i=0}^{K-1} 2^i A_i - \sum_{i=0}^{K-1} 2^i B_i \geq 1 \tag{5.3}$$

3. **K-bit Bitwise Logic operations ($LK(A, B) or LK(A)$)** Given two K-bit numbers A and B, a K-bit LK (=AND, OR, NAND or NOR) operation on A and B can be performed using K threshold functionsMuroga (1971b). For instance, each bit of the K-input AND gate is implemented as $Y_i \equiv A_i + B_i \geq 2 \ \forall i \in [0, .., K-1]$. Similarly, each bit of LK (=NOT) operation on A can be performed using the threshold function $Y_i \equiv 2\overline{A_i} \geq 1 \ \forall i \in [0, .., K-1]$. K-input LK (=XOR or XNOR) on A and B is implemented using a network of a two-input and a three-input threshold functionMuroga (1971b) for each output bit.

In a TULIP-PE, each cluster's i^{th} neuron ($i \in [1, K]$) computes the i^{th} significant bit of a primitive operation's output. Therefore, we construct a new threshold function Q_i for the i^{th} neuron, that contains the threshold functions of the i^{th} significant

bit of sum, carry, comparison, and logic operations (See Equation 5.1, 5.2 and 5.3). Q_i is defined as follows:

$$Q_i = Z_0 + \sum_{j=0}^{i-1} 2^j X_j \geq Z_1 + \sum_{j=0}^{i-1} 2^j Y_j, \quad 1 \leq i \leq K. \tag{5.4}$$

Primitive	i^{th} bit	Q	Z_0	X	Z_1	Y
ADDK	Carry	Q_i	C_0	$A_{i-1..0}$	0	$\overline{B}_{i-1..0}$
	Sum	Q_2	C_i	A_i	C_{i-1}	$\overline{B_i}$
COMPK	Compare	Q_i	C	$A_{i-1..0}$	0	$B_{i-1..0}$
LK	AND	Q_2	A_i	B_i	1	0
	OR	Q_2	A_i	B_i	0	0
	NAND	Q_2	$\overline{A_i}$	$\overline{B_i}$	0	0
	NOR	Q_2	$\overline{A_i}$	$\overline{B_i}$	1	0
	NOT	Q_2	$\overline{A_i}$	0	0	0

Table 5.1: Mapping Primitive Operations to Binary Neuron

A neuron can switch between its designated functions by appropriately assigning signals to Equation 5.4 and controlling their polarity, as shown in Table 5.1. Furthermore, a neuron indexed i can support all the functions supported by neurons indexed $i-1$ or lower. Note that unused signals are set to 0.

TULIP-PE requires a minimum of four clusters to ensure a single cycle delay between the launch of any two consecutive primitive operations. Considering that each primitive operation can be represented as a two-level (or one-level) computation of threshold functions, only two clusters are needed to perform the computation at any given time (compute mode), while the remaining two clusters are needed to read operands from their respective local registers and share them with the first two clusters (routing mode). The clusters switch between the compute and routing mode

depending on the local registers in which the operands are stored, and the local register in which the output must be written to.

Note that the number of bits that can be processed in each cycle increases as the number of neurons in each cluster (K) increases, i.e., better performance. However, as seen in Equation 5.4, as K increases, the fan-in of the binary neuron increases. Since there is a maximum fan-in limitation of the binary neuronWagle *et al.* (2022b), K is set to 5 for this paper.

Operations of a QNN that have a bitwidth N ($> K$), i.e. N-bit addition, accumulation, comparison, multiplication, pooling, ReLU, and logic operations, are decomposed into primitive operations, as follows: N-bit addition, comparison, or logic operations are decomposed into a cascade of $\lceil N/K \rceil$ primitives. N-bit multiplication of two operands A and B is done by first generating partial products (using N-bit AND operations), and then adding them using a $log_2(N)$-level adder tree (tree of addition operations) to generate the final result. ReLU can be expressed by first comparing the operand with 0, and then performing a bit-wise AND of the operand with the result of the comparison. The Maximum operation on two N-bit numbers A and B is realized by first comparing A and B to generate an output Y, after which an AND operation is used to generate intermediate results AY ($= A$ if $A > B$, else 0) and $B\overline{Y}$ ($= B$ if $A \leq B$, else 0). Finally, adding the two intermediate results yields the maximum value out of A and B. Max-pooling operation is realized by constructing a binary tree of Maximum operation.

5.4 Top Level Architecture of TULIP

This section presents the top-level hardware architecture of TULIP, which is used to accelerate QNNs. As shown in Figure 5.5, it consists of four major components: an image buffer (stores input pixels), a kernel buffer (stores weights), a R $\times$ C grid

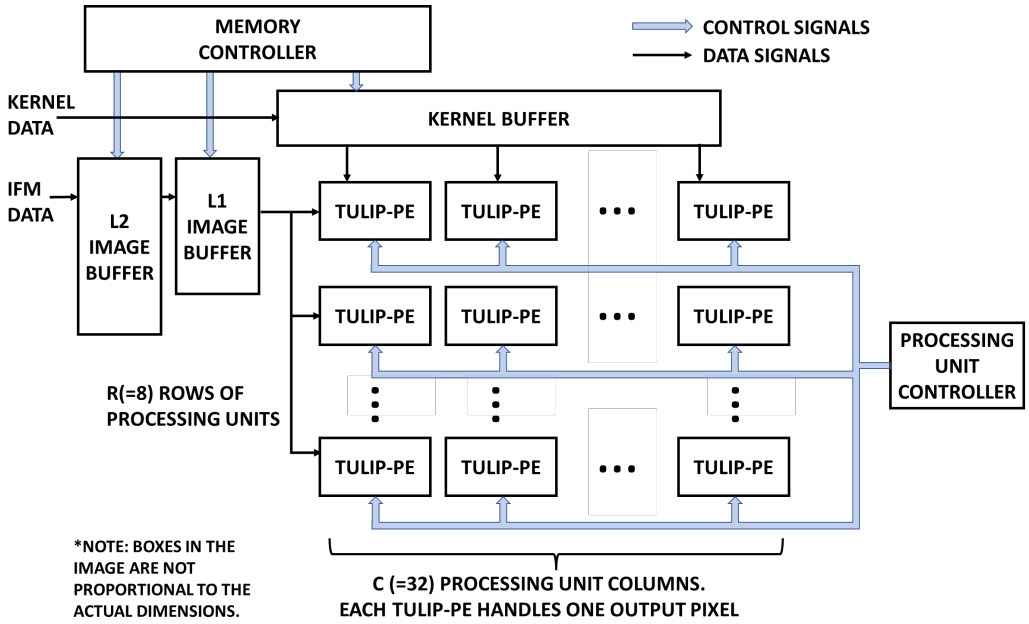


Figure 5.5: TULIP Top Level Architecture: Controller Configures the Processing Units. the Input Pixels and Weights Are Sent Through Image and Kernel Buffers. the Output of the Processing Units Is Collected in the Output Buffers Before Sending It Back to the memory.

of TULIP-PEs (performs computation), and a controller (sends necessary control signals). TULIP-PEs in the same row share input pixels, and TULIP-PEs in the same column share weights. In order to execute QNNs on this architecture, the following two problems must be solved.

The first problem entails scheduling the nodes of a QNN (Illustrated in Figure 5.7a)) onto the 2-D grid of processing elements (Figure 5.5). This scheduling depends on the arrangement of the processing elements and the size of the input and weight buffers. This schedule must also achieve high utilization of processing elements and minimize access to off-chip memory. This problem has been addressed by numerous other implementations, resulting in a collection of heuristic algorithms Chen *et al.* (2016); Andri *et al.* (2017); Ma *et al.* (2019b); Dave *et al.* (2019). Their heuristic

algorithms minimize data fetches from the external memory by exploiting the QNN's underlying structure that is based on 2D convolution, which is implemented as nested loops in the software. Consider the 2D convolution illustrated in Figure 5.6. The dimensions of the input image are (I, I, L), output image are (O, O, M) and weights are (K, K, L, M). For this convolution, the opportunities for data reuse are as follows:

1. Each input pixel can be reused $\lfloor K^2O^2/I^2 \rfloor$ times when computing one dimension of the output image.

2. Each kernel weight is reused O^2 times.

3. Each dimension of the input (L) is reused M times.

Existing heuristic algorithms can be used to assign the nodes of a given QNN to the TULIP-PEs, in a way that maximizes the reuse of input pixels and weights (listed above) and achieves high energy efficiency.

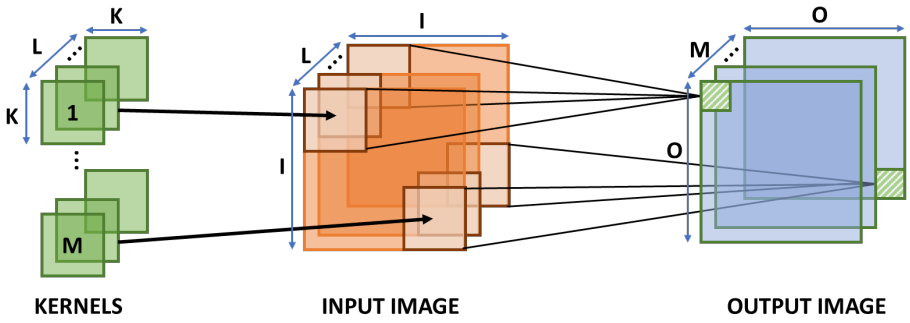


Figure 5.6: Data Reuse Opportunities in 2-D Convolution: Each Input Pixel Can Be Reused $\lfloor K^2O^2/I^2 \rfloor$ Times and Each Kernel Weight Is Reused O^2 Times for One Output Dimension. Each Dimension of the Input (L) Is Reused M times.

Each node of the QNN is a dataflow graph itself. Therefore, the second problem, which is the main focus of this paper, is to discuss how this dataflow graph is scheduled on the TULIP-PE, and is discussed in further detail in the following sections.

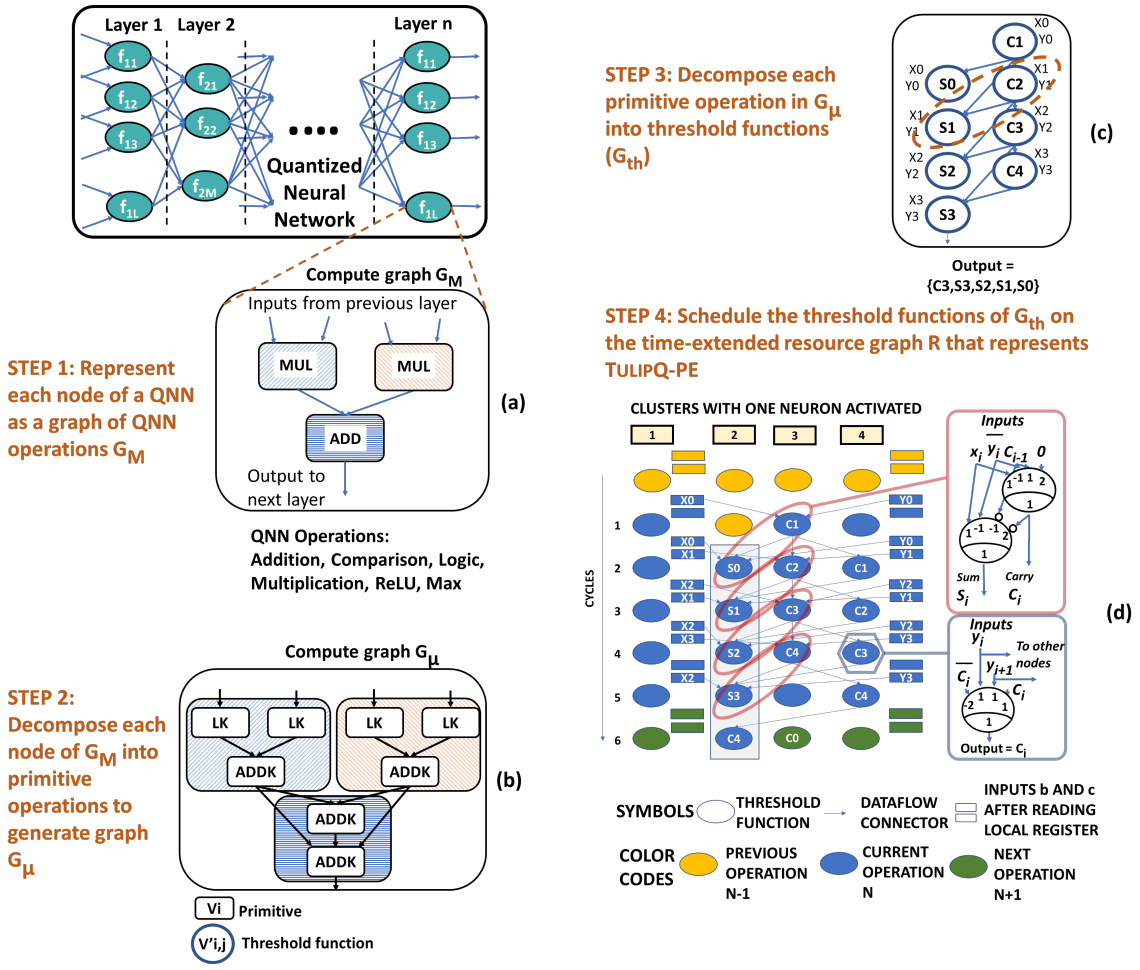


Figure 5.7: Mapping a Node of a QNN to TULIP-PE as an Equivalent Compute Graph M. for Illustration Purposes, K = 1, i.e. One Neuron Is Used in Each cluster.

Section 5.4 described how TULIP-PEs are integrated in the top-level TULIP architecture and how nodes of a QNN are assigned to them. Since each QNN node is a dataflow graph itself, this section addresses the problem of scheduling this dataflow graph on a TULIP-PE.

Let M be a graph that represents a node of a given QNN. Figure 5.7b shows an example of M. It is defined using Definition 5.5.1 as follows:

Definition 5.5.1 (Compute Graph $M(V_M, E_M)$)**.** It is a directed acyclic graph where each node $v \in V_M$ represents an operation of a QNN, such as addition, comparison, logic, multiplication, ReLU, or maximum. Each edge $e \in E_M$ represents a data dependency between the operations.

In order to schedule M on the TULIP-PE, it must undergo two levels of transformations; so that each node of the resulting graph is a threshold function that can be mapped to a binary neuron in the TULIP-PE. We first describe the necessary transformations on M, and then discuss the problem of scheduling the transformed graph to the TULIP-PE.

The first transformation of M involves the decomposition of each vertex $v \in V_M$ into an equivalent subgraph of primitive operations that computes the same function, as shown in Figure 5.7c. The resultant graph is called a primitive graph P and is defined using Definition 5.5.2 as follows:

Definition 5.5.2 (Primitive Graph $P(V_P, E_P)$)**.** It is a directed acyclic graph where each node $v \in V_P$ represents a K-input primitive operation, i.e. K-bit addition, comparison, or logic. Each edge $e \in E_P$ represents a data dependency between the primitive operations.

The second transformation involves further decomposition of each primitive operation $v \in V_P$ into an equivalent subgraph of threshold functions, as shown in Figure 5.7d. The resultant graph is called a threshold graph G and is defined using Definition 5.5.3 as follows:

Definition 5.5.3 (Threshold Graph $G(V_G, E_G)$)**.** It is a directed acyclic graph where each node $v \in V_G$ is a threshold function and each edge $e_{u,v} \in E_G$ represents the data dependency between two threshold functions u and v.

Finally, each threshold function of G is scheduled on the TULIP-PE architecture. Figure 5.7e illustrates how the threshold functions of the primitive $ADDK$ are scheduled to the binary neurons of TULIP-PE. Data stored in the local registers of clusters 1 and 4 are read by clusters 2 and 3 to generate the sum and carry bits. The problem of scheduling G to TULIP-PE is discussed in further detail below.

TULIP-PE is represented as a so-called time-extended resource graph in order to schedule G on it. It represents the TULIP-PE as a resource at various instances of time, and is described using Definition 5.5.4 as follows:

Definition 5.5.4 (Time Extended Resource Graph $R(V_R, E_R, T)$)**.** It is a directed acyclic graph where each node $v \in V_R$ is a resource that represents a tuple (v', t) of a neuron (or a local register) v' in TULIP-PE and time $t \in [0, T)$. An edge between two resources $u : (u', t)$ and $v : (v', t + 1)$ is represented as (u', v', t) and it indicates that the output of u' is given as an input to v' at time t. Edges don't exist between resources if they differ by more than 1 unit of time, or if there is no physical datapath between their associated neurons (or local registers).

Using the notation discussed above, the problem of scheduling of G to R is formulated as follows:

Problem 1 (Compute graph scheduling problem)**.** The problem is to construct a time extended resource graph R of minimum extension (smallest T) for which

1. there exists a surjective mapping $M : V_R^* \rightarrow V_G$, where $V_R^* \subset V_R$, such that for each neuron $v \in V_R^*$ can implement the threshold function $v' \in V_G$ mapped to it.

2. for every arc in $(u, v) \in E_G$, there is a path $P = (u', r_1, , ...r_n, v')$ such that u'

is a neuron associated u, v' is a neuron associated v, and $r_1, .., r_n$ are nodes of R that facilitate the datapath between u and v.

It must be noted that the TULIP-PE architecture is similar to a CGRA architecture. A CGRA consists of a systematic arrangement of compute units, such that each compute unit is connected to its neighboring compute unit through multiplexers. The TULIP-PE also consists of a systematic arrangement of binary neurons interconnected through multiplexers. Therefore, Problem 1 is similar to the problem of scheduling compute graphs on a CGRAs. It has already proven as an NP-complete Hamzeh *et al.* (2012) problem.

5.5.1 Existing Solutions and Their Limitations

Existing solutions that map arbitrary graphs to CGRA can also be used to map graphs to the TULIP-PE architecture. Several state-of-the-art CGRA scheduling algorithms in the literature Hamzeh *et al.* (2014); Chen *et al.* (2021); Canesche *et al.* (2021); Balasubramanian and Shrivastava (2020); Balasubramanian *et al.* (2018) typically involve the following three steps (Pseudo-code in Algorithm 1): First, the tuples are enumerated (Algorithm-1, line 2), where each tuple (o, p, t) represents a unique mapping of an operation o on a neuron p at a time t . Second, an edge between two nodes is added if they can coexist without any resource or data-dependency conflicts. The resultant graph is called a compatibility graph (Algorithm-1, line 3). Figure 5.8 shows an example of a compatibility graph C that is constructed using the G and R. Third, from this graph, the algorithm generates a maximum clique of tuples that are compatible (Algorithm-1, line 4). In Figure 5.8 (b), the tuples $(A, R1, 1)$ and $(B, R2, 2)$ are compatible, and form a clique. Since this clique contains a mapping for all the nodes of G (Algorithm-1, lines 5-6), this clique represents a feasible schedule, as shown in Figure 5.8 (c). If the above procedure does not result in a feasible schedul-

ing solution, then either G is transformed to make conflicting mappings compatible, or the total schedule time T is increased (Algorithm-1, line 9), before re-running the procedure again.

Algorithm 1 Mapping Compute Graph G to Resource Graph R

Input: G, R

Output: $\phi : V_G \rightarrow V_R$, $\xi : E_G \rightarrow E_R$

1: **for** *looper* = 0 to *max_limit* **do**
2: V_C = enumerate_mappings(V_G, V_R)
3: C = create_compatibility_graph(V_C)
4: M = maximal_clique (C)
5: **if** $|M| \equiv |G|$ **then**
6: ϕ, ξ = extract_mapping (M)
7: break
8: **else**
9: G = Transform (G)
10: **end if**
11: **end for**

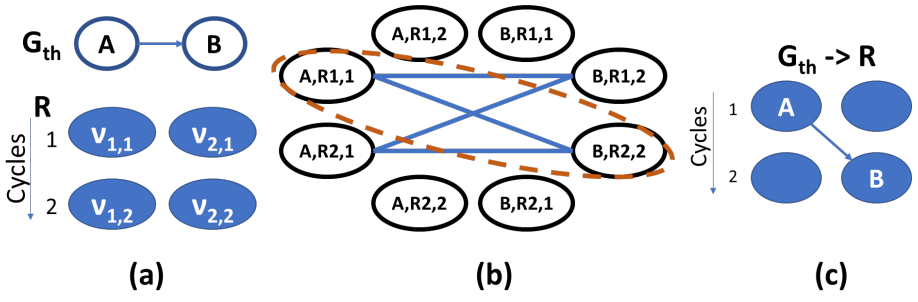


Figure 5.8: Scheduling Graphs of Threshold Functions on Binary Neurons A) Compute Graph G and Time-Extended Resource Graph R B) Compatibility Graph of G and R. C) Mapping Solution of G to R.

Algorithm-1 takes an impractically large amount of time when the nodes in G are over 50. This is because the size of the compatibility graph C increases by a factor of $W = (|V_G|)^2$ as the number of nodes in G increases and the time needed to find a maximal clique in C requires at-most $\mathcal{O}(W^8)$ steps Dave *et al.* (2018). However, Algorithm-1 is still required because it is the only algorithm that considers the constraints of the hardware while performing the scheduling. Therefore, there is a need to approach Problem 1 using a more sophisticated algorithm that allows us to run Algorithm-1 on G part by part (partitioning G), instead of processing all the nodes of G at once. This approach is made possible by the proposed algorithm, which is discussed later in Section 5.5.2. By partitioning G, the overall execution time drops significantly. For analysis sake, if we assume that G is partitioned into D parts, then the size of the compatibility graph C_i for each partition i now only increases by a factor of $W' = (\sum_i |V_G|/C_i)^2 = W/D^2$. As a result, the time needed to find a maximal clique of all C_i requires at-most $\mathcal{O}(W^8/D^{15})$.

Note that traditional high-level scheduling algorithms (used for task allocation to CPUsKu and De Micheli (1991); Landwehr *et al.* (1994)) do not apply when performing scheduling on TULIP-PE. This is because, unlike high-level scheduling algorithms, routing-aware scheduling algorithms used for CGRAs generate a valid schedule while also honoring the routing constraints that arise due to the physical limitations (bandwidth, connectivity, etc.) of the hardware.

5.5.2 Decomposing the Compute Graph Scheduling Problem

The previous subsection showed that the compute graph scheduling problem can be solved using Algorithm-1, but takes an impractically large amount of time when the number of nodes in G is high. Instead of using Algorithm-1 naively, this chapter proposes a different approach. Before transforming the primitive graph P to G, we

first identify the order in which the primitives are executed on TULIP-PE and how their outputs are stored in the local registers. Once this is known, each primitive can then be scheduled separately using Algorithm-1, within its designated time slot. As a result, the time complexity of mapping $G \rightarrow R$ reduces substantially, because the above procedure is equivalent to partitioning G into smaller subgraphs that represent primitives. This two-step process is represented using the following two sub-problems:

Problem 2 (Primitive scheduling problem)**.** Given a primitive graph P, the problem is to identify the order in which the nodes of V_P must be scheduled on a TULIP-PE, such that

1. The execution time of P on a TULIP-PE is minimized.
2. A local register is allocated for storing the output of each node in V_P, in a way that does not exceed the maximum capacity of the local registers, at any time in the schedule.
3. The data-dependency of each node in V_P is satisfied in the schedule.
4. The schedule adheres to the hardware constraints of TULIP-PE.

Problem 3 (Primitive mapping problem)**.** For each primitive $v_i \in V_P$, let $v_i \rightarrow G_i = (V_i, E_i)$ be an equivalent graph of threshold functions. Then the problem is to find a feasible mapping $G_i \rightarrow R$, given the local registers from which the input data is fetched and output data is stored.

The primitive scheduling problem (Problem 2) is an NP-Complete problem. Its proof is discussed later in this section. It can be solved by representing it as an integer linear programming (ILP) formulation, given below. This formulation is based on the high-level scheduling algorithm presented in Landwehr *et al.* (1994), but the proposed

Notation	Description
s_v	Time at which storage of node v begins
e_v	Time at which storage of node v ends
B	Size of each local register
L	Total number of local registers
T	Maximum time required to execute all the primitives on R; $T = 2\|V_P\|$.
Y	Positive constant greater than T, for ILP purposes only
S	Positive constant less than 1, for ILP purposes only
J	$[0, .., L-1]$
π	$[0, .., T-1]$
$u \prec v$	u is the immediate predecessor of v
E	Makespan (execution time) of P on TULIP-PE

Table 5.2: Notation for ILP Used to Solve the Primitive Scheduling Problem

ILP also adds routing constraints (explained later). Table 5.2 lists the notation used in the proposed ILP. The primitive scheduling problem has to establish bindings between operations v, time steps t, and resources (local registers) r. Such bindings can be represented using triple-indexed binary decision variables, shown in Equation 5.5.

$$\chi_{v,t,r} = \begin{cases} 1 & \text{if } v \text{ is mapped to } r \text{ at time } t \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

Using the above equation, two additional binding variables are derived: $\rho_{v,r}$, which represents the mapping of v with local register r, and $\tau_{v,t}$ which represents the map-

ping of v with time t. These variables are used to express resource and time-specific constraints respectively.

$$\rho_{v,r} = \begin{cases} 1 & \text{if } \exists t,\ \chi_{v,t,r} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

$$\tau_{v,t} = \begin{cases} 1 & \text{if } \exists r,\ \chi_{v,t,r} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

With the goal of minimizing the makespan (execution time) of P on TULIP-PE, the following constraints are needed to define the set of feasible solutions:

$$\textbf{Minimize } E \text{ such that} \tag{5.8}$$

I. Resource availability constraints: These constraints are added to ensure that the local registers are not over-utilized. The first constraint (Equation 5.9) ensures that the storage used by a local register r at any time t must never exceed the maximum capacity B.

$$\forall r \in J, \forall t \in \pi : \qquad \sum_{\forall v \in V_P} \chi_{v,j,t} \leq B \tag{5.9}$$

The second constraint (Equation 5.10) expresses the relationship between the variables ρ and χ.

$$\forall v \in V_P,\ \forall r \in J, \forall t \in \pi : \qquad \chi_{v,j,t} \leq r_{v,j} \leq \sum_{\forall t \in \pi} \chi_{v,j,t} \tag{5.10}$$

The third constraint (Equation 5.11) ensures that each primitive's output is stored in only one local register.

$$\forall v \in V_P : \qquad \sum_{\forall r \in J} \rho_{v,r} = 1 \tag{5.11}$$

2. Precedence constraints: Constraint in Equation 5.12 is added to ensure that the data dependency due to the precedence relationship between any two primitives u and v is satisfied in the schedule.

$$\forall u, v \in V_P \text{ such that } u \prec v : \qquad s_u < s_v \leq e_u \tag{5.12}$$

For the example schedule shown in Figure 5.9, $s_u = 1$, $e_u = 2$, and $s_v = 2$.

3. Timing validity constraints:

These constraints ensure that the start and end times of all the nodes are valid and feasible (Equation 5.13), and that the start times of any two nodes are not equal (Equation 5.14). In Equation 5.14, the temporary variable $x = 1$ if the start time $s_u > s_v$ and $x = 0$ if $s_u < s_v$.

$$\forall v \in V_P : \qquad 0 \leq s_v < e_v \leq T \tag{5.13}$$

$$\forall u, v \in V_P, \quad u \neq v : \qquad Y - S \geq Y \times x + s_v - s_u \geq S \tag{5.14}$$

The following constraints are added to set the value of $\tau_{v,t}$. This variable indicates if the output of node v is stored at time t. To set τ, two additional variables θ and λ are introduced. The variable $\theta_{v,t} = 1$ if t is greater (or equal) than the start time of node v, and is 0 otherwise (Equation 5.15). Similarly, the variable $\lambda_{v,t} = 1$ if t is less (or equal) than the start time of node v, and is 0 otherwise (Equation 5.16). Finally, $\tau_{v,t} = 1$ if both $\theta_{v,t} = \lambda_{v,t} = 1$, and is 0 otherwise (Equation 5.17).

$$\forall v \in V_P, \quad \forall t \in \pi : \qquad t + 1 \leq s_v + Y \times \theta_{v,t} \leq t + Y \tag{5.15}$$

$$t - 1 \leq e_v - Y \times \lambda_{v,t} \leq t - Y \tag{5.16}$$

$$\theta_{v,t} + \lambda_{v,t} - 1 \leq \tau_{v,t} \leq \theta_{v,t}, \lambda_{v,t} \tag{5.17}$$

Constraint in Equation 5.18 is added to identify the end time of the last primitive

that will be scheduled on TULIP-PE; so that it can be minimized in the objective function. Minimizing this end time (E) is equivalent to minimizing the makespan (execution time) of P on TULIP-PE.

$$\forall v \in V_P : \qquad e_v \leq E \tag{5.18}$$

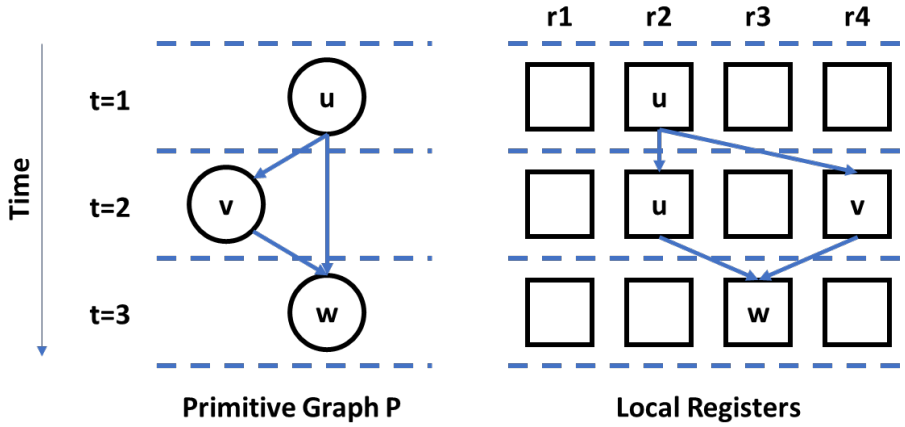


Figure 5.9: Example to Illustrate Primitive Scheduling Problem. the Output of Each Node in the Primitive Graph P Is Stored in the Local Registers of TULIP-PE.

4. Routing constraints:

The following constraints ensure that the data-routing capabilities of local registers are not violated, and are explained as follows. A local register can perform either a read or a write operation at any given time, but not both simultaneously. Therefore, two nodes that share an edge cannot be assigned to the same local register. In Figure 5.9, since u is the immediate predecessor of v, the output of u is stored in different local register than v. While the output of u is read from a local register, the output of v is simultaneously written to a different local register (Equation 5.19). Furthermore, two sibling nodes cannot be assigned the same local register. As shown in Figure 5.9, u and v are immediate predecessors of w. Therefore, u and v cannot have the same

local register. This constraint is because the local registers supply only one operand to each primitive in TULIP-PE. As a result, we need two separate local registers to provide two operands (Equation 5.20).

$$\forall u, v \in V_P, \quad \forall r \in J, \quad u \prec v : \qquad \rho_{u,r} + \rho_{v,r} \leq 1 \tag{5.19}$$

$$\forall u, v, w \in V_P, \quad \forall r \in J, \quad u, v \prec w : \qquad \rho_{u,r} + \rho_{v,r} \leq 1 \tag{5.20}$$

Neuron inputs	Parameters	Time (Sec)	Local Reg Size (B)
64	6.88E+03	0.04	2
128	2.81E+04	0.12	2
256	1.14E+05	0.49	3
512	4.56E+05	2.14	4
1024	1.83E+06	9.32	4
2048	7.33E+06	41.82	4
4096	2.93E+07	216.46	5

Table 5.3: Number of ILP Parameters and Run-Time for Solving Primitive Scheduling Problem, for Compute Graphs That Represent Neurons with Varying Number of inputs.

Table 5.3 shows the number of parameters that are generated when running the ILP for compute graphs that represent neurons with a varying number of inputs. For even the largest size neurons, the time required to run the ILP is very less.

Proof of NP Completeness: In the primitive scheduling problem, routing constraints specify that if two primitive nodes that share an edge, or if two sibling primitive nodes exist, they cannot be assigned the same local register. Since there are four local

registers, the problem of allocating registers to primitive operations is equivalent to a k-coloring problem (where k equals 4) (NP CompleteAppel and Haken (1978)). Here, the primitives represent the nodes of the graph to be colored, and the local registers represent the colors that need to be assigned. Therefore, the primitive scheduling problem is also an NP-complete problem.

Algorithm to map primitive operations to R: The problem of mapping the threshold functions of each primitive G_i to the resource graph R, i.e. $G_i \rightarrow R$ can be solved using Algorithm 1.

Finally, the solutions from both the algorithms discussed in this sub-section are combined to form a single schedule that maps G to R.

The algorithm described above enables TULIP-PE to modify its schedule depending on the number of neurons enabled in each cluster. Figure 5.10 a) and b) show how the schedule of addition operation can be varied based on the available neurons (denoted by K). For example, assume that we need to execute an addition operation of two 4-bit numbers, X and Y. TULIP-PE uses five cycles (4 cycles before the next primitive can be launched) to finish its addition operation if only one neuron (K=1) is enabled in each cluster. However, if the number of neurons in each cluster is doubled (K=2), the schedule can be re-adjusted to finish the addition operation in four cycles (3 cycles before the next primitive can be launched). If all five neurons are enabled in each cluster, then TULIP-PE would only require two cycles (1 cycle before the next primitive can be launched) to finish the addition operation. This critical feature enables a run-time trade-off between delay and energy efficiency on the TULIP-PE. Furthermore, if some neurons in the manufactured chip stop working, those neurons can be bypassed by modifying the schedule.

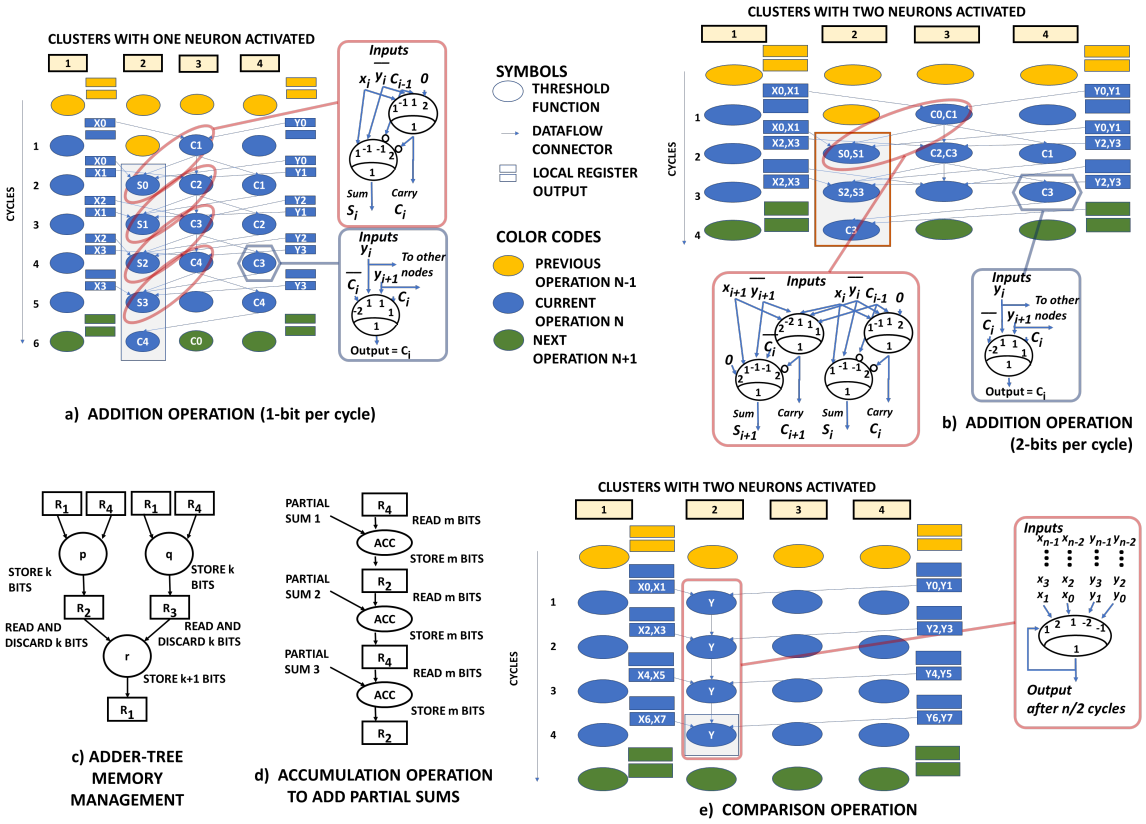


Figure 5.10: Addition Operation, Adder-Tree, Accumulation, and Comparison Using TULIP-PE Architecture. Depending on the Number of Neurons Available in Each Cluster, the Scheduler Can Automatically Tune the Schedule for the Best performance.

5.6 Enhancing Data-reuse Using TULIP-PEs

This section provides the theoretical basis of how the use of TULIP-PEs enhances data-reuse, as compared to a MAC unit. This is done by comparing the delay and area complexity of both the processing elements as follows: Let m and n be the number of bits needed to represent inputs and weights respectively. If both the processing elements need to multiply and accumulate N such pairs, then the area complexity of the MAC unitGarland and Gregg (2018) and the TULIP-PE is $\mathcal{O}(mn)$ and $\mathcal{O}(1)$ respectively. The area complexity of TULIP-PE is a constant because the

hardware does not change irrespective of the workload. The delay complexity of the MAC unitGarland and Gregg (2018) and the TULIP-PE is $\mathcal{O}(N)$ and $\mathcal{O}(mnN)$ respectively. TULIP-PE has a worse delay complexity than a MAC unit because it processes each operation K bits at a time. Although the TULIP-PE is smaller, it is much slower than a MAC unit. Note that these trade-offs change when MACs and TULIP-PEs are analyzed in a SIMD architecture.

	Gate complexity	Delay complexity
1 MAC Unit	$\mathcal{O}(mn)$	$\mathcal{O}(N)$
1 TULIP-PE	$\mathcal{O}(1)$	$\mathcal{O}(mnN)$
Row of C MAC units	$\mathcal{O}(Cmn)$	$\mathcal{O}(N/C)$
Grid of $R \times C$ TULIP-PEs	$\mathcal{O}(CR)$	$\mathcal{O}(mnN/CR)$

Table 5.4: Gate and Delay Complexity of MAC Units and TULIP-PEs. TULIP-PEs Match the Delay and Gate Complexity of MAC Units When $R = mn$. However, Since There Are Now R TULIP-PEs for Every MAC Unit, the Increased Parallelism Promotes Data Sharing, Thereby Improving Data Reuse by a Factor of R.

Consider the following two SIMD architectures. First is the baseline architecture, which consists of a row of C MAC units. We use this architecture as a reference. Second is the TULIP architecture, with a grid of $R \times C$ grid of TULIP-PEs. The baseline has a gate complexity of $\mathcal{O}(Cmn)$ and a delay complexity of $\mathcal{O}(N/C)$. Similarly, TULIP has a gate complexity of $\mathcal{O}(CR)$ and a delay complexity of $\mathcal{O}(mnN/CR)$. TULIP can match the area and delay complexity of the baseline by setting $R = mn$. However, TULIP is still better than the baseline. This is because the grid arrangement provides higher opportunities for weight reuse. If we assume that a workload of $R \times C$ graphs will be processed by both the architectures, then the baseline would

fetch each weight R times whereas the TULIP would fetch each weight just once. As a result, significant energy-efficiency improvements are observed by enhancing the data-reuse. The complexity analysis discussed above is summarized in Table 5.4.

Note that the concept discussed above has already been used in other design settings. For instance, processor designers often choose to use several slower cores instead of using fewer faster cores, to enhance the energy-efficiency without compromising on throughput. The work presented in this chapter also uses this concept, but at the level of processing elements. TULIP replaces the traditionally used MAC units with slower but more energy-efficient TULIP-PEs.

5.7 Experimental Results

5.7.1 Experimental Setup

TULIP architecture was evaluated using TSMC 40nm LP library. Synthesis was done using Cadence Genus, and then the placement and routing were done using Cadence Innovus. Timing checks were performed using cross-corner analysis at {SS, 125C, 0.81V}, {TT, 25C, 0.9V} and {FF, 0C, 0.99V}.

Table 5.5 demonstrates that a binary hardware neuron is substantially better than its conventional CMOS standard cell equivalent in terms of area, power, and delay. This advantage is significant since TULIP uses this hardware neuron for all operations (Computation of partial sums, comparison, RELU, and max pool). TULIP was synthesized and placed using TSMC 40nm-LP standard cells with Cadence Genus and Innovus (Figure 5.11). The VCD file generated using real QNN workloads was used for power analysis to model switching activity accurately.

For comparison against an equivalent baseline architecture, we chose a recent 1-bit QNN accelerator named YodaNN Andri *et al.* (2017), designed in 65nm UMC

	Binary neuron		Logic equivalent		Improvement (X)	
κ	Area (um2)	Power (uW)	Area (um2)	Power (uW)	Area	Power
1	15.6	6.93	33	10.23	2.12	1.47
2	15.6	7.79	44	11.92	2.82	1.53
3	15.6	8.82	88	22.31	5.64	2.53
4	16.7	9.38	104	36.35	6.67	3.88
5	16.7	9.68	216	48.12	13.85	4.97

Table 5.5: Binary Neuron Comparison: Hardware Neuron Versus Standard Cell Neuron, Operating at 434MHz (Time Period:2300ps). κ Indicates the Index of Neuron in a cluster.

technology. Then, we modified the same architecture to add support for 2 to 4-bit QNNs. To make a fair comparison, we implemented the entire baseline design in the same technology as TULIP (40nm-LP from TSMC) and synthesized, placed, and routed both the designs. TULIP and Baseline were both designed for up to 12-bit inputs and 4-bit weights. The 12-bit inputs are only needed for handling integer layers of a QNN. Therefore, we added clock gating appropriately whenever fewer input bits were used. Both the baseline as well as TULIP architecture support kernel sizes of 3, 5, and 7.

Other ASIC architectures are available in the literature, such as XNORBIN Al Bahou *et al.* (2018), which use more advanced memory techniques to improve energy efficiency. However, these architectures do not support integer layers and are unsuitable for comparison. Although the original paper of YodaNNAndri *et al.* (2017) does not report the throughput and energy efficiency for fully connected layers, we estimate the throughput and power by performing an element-wise matrix multiplication using the MAC units present in their architecture. There are a few other ASIC archi-

tectures Knag *et al.* (2020); Moons *et al.* (2018) that deliver higher energy-efficiency than the TULIP architecture. However, these architectures can only support kernels of size 2 and only 1-bit QNNs. Furthermore, these architectures have never been demonstrated for processing large neural networks such as the ones used for Imagenet Classification, using the reported energy-efficiency. Meanwhile, TULIP delivers high-energy efficiency, while also delivering support for all the kernel sizes available in the common neural networks.

5.7.2 Evaluation of TULIP-PE Against MAC

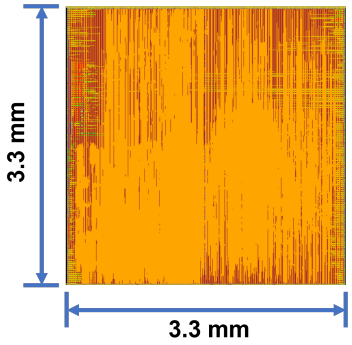
In Table 5.6, the baseline 18-bit reconfigurable MAC unit based on the design present in YodaNNAndri *et al.* (2017) is compared against the TULIP-PE module with five neurons in each cluster. Note that both the MAC unit and TULIP-PE can compute integer as well as quantized layers. In large QNN architectures such as AlexnetRastegari *et al.* (2016), the initial layers are integer layers, while the rest are quantized. Since the computation technique between Baseline and TULIP significantly differs for quantized layers, the comparison of the MAC and the TULIP-PE is made for the quantized layers. Both modules perform the weighted sum for quantized activations and quantized weights. The MAC unit realizes accumulation by multiplying and accumulating one window in each cycle. Meanwhile, TULIP-PE treats the weighted sum as a compute graph consisting of multiplication operations connected to an adder-tree. This is important because TULIP realizes adders, multipliers, etc., of custom bit widths, thereby eliminating the loss of energy incurred by MAC unit that uses max-width addition and multiplication operation in every cycle. Based on Table 5.6, we note that the TULIP-PE is 15.8X smaller than the MAC unit and consumes up to 125X less power. However, it consumes 9.5X more time than the MAC unit since it performs bit-level addition. As a result, the power delay product

	Bit width	1	2	3	4
Tulip PE	Power (mW)	0.18	0.18	0.18	0.18
	Cycles (#)	155	170	227	307
	Time (ns)	356.5	391.0	522.1	706.1
	PDP (pJ)	65.5	71.8	95.9	129.7
MAC Unit	Power (mW)	2.6	8.0	16.2	22.6
	Cycles (#)	32	32	32	32
	Time (ns)	73.6	73.6	73.6	73.6
	PDP (pJ)	189.4	590.3	1193.8	1666.3
Ratio (X=B/T)	PDP	2.9	8.2	12.4	12.8

Table 5.6: Comparison of Fully Reconfigurable MAC Unit Based on YodaNN Architecture Andri *et al.* (2017), with a TULIP-PE (K=5), for Computing a 288 Input Weighted Sum (32 Input Channels, Kernel =3x3). TULIP-PE Is 15.8X Smaller than the MAC Unit. PDP: Power Delay Product

of a TULIP-PE is up to 5.8X lower than the MAC unit while at the same time being 15.8X smaller than the MAC.

Furthermore, since a MAC unit cannot compute operations such as comparison, max-pooling, etc., the data is sent to other parts of the chip for these operations in Andri *et al.* (2017). However, the TULIP-PE can preserve the data locality and perform the comparison and max-pooling operations internally without moving the data to other modules, saving additional energy.



Technology	TSMC 40LP
Area	10.6 mm^2
L2/L1/ Kernel Area	693K/327K/ 2203K μm^2
Processing Unit. Area	2677K μm^2
Controller Area	10K μm^2
# Std. Cells	2958K
# Nets	1584K
Wirelength (m)	88.7
# Metal Layers	6

Figure 5.11: Layout of TULIP Architecture in TSMC 40nm-LP

5.7.3 Evaluation of the TULIP Architecture

For this paper, TULIP was designed with 256 TULIP-PEs to ensure that the chip area of TULIP matches that of the baseline architecture. However, the number of processing units in TULIP can be scaled to suit the application, if needed.

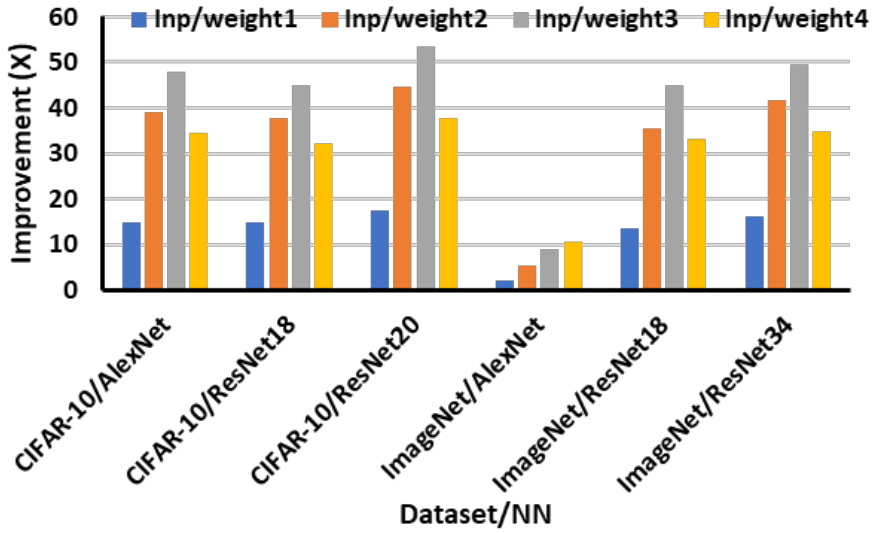
Tables 5.7 and 5.8 show the energy-efficiency and throughput values for various neural networks (at varying bit-precisions), accelerated using both the TULIP and baseline architecture. For TULIP, two sets of results are presented: 1) TULIP tuned for best energy-efficiency, and 2) TULIP tuned for best throughput. Here, tuning

I/W bits	Benchmark		TULIP-Q for En. Eff.			TULIP-Q for Perf		
	En. Eff.	Perf.	En. Eff.	Perf.	K	En. Eff.	Perf.	K
(a) AlexNet								
1	3.1	66.6	63.6 (20.9X)	73.2 (1.1X)	3	45.1 (14.8X)	74.5 (1.1X)	5
2	1.0	66.6	45.7 (46.2X)	67.7 (1.0X)	3	38.7 (39.1X)	72.6 (1.1X)	5
3	0.5	66.6	23.6 (48.2X)	67.6 (1.0X)	5	23.6 (48.2X)	67.6 (1.0X)	5
4	0.4	66.6	12.2 (34.8X)	61.6 (0.9X)	5	12.2 (34.8X)	61.6 (0.9X)	5
(b) ResNet18								
1	2.4	31.6	51.3 (21.2X)	36.2 (1.1X)	3	35.4 (14.6X)	36.8 (1.2X)	5
2	0.8	31.6	35.3 (44.7X)	33.1 (1.0X)	3	29.7 (37.6X)	35.3 (1.1X)	5
3	0.4	31.6	17.7 (45.4X)	32.5 (1.0X)	5	17.7 (45.4X)	32.5 (1.0X)	5
4	0.3	31.6	9.0 (32.2X)	29.5 (0.9X)	5	9.0 (32.2X)	29.5 (0.9X)	5
(c) ResNet20								
1	2.3	53.3	57.6 (25.3X)	75.1 (1.4X)	3	39.6 (17.4X)	77.6 (1.5X)	5
2	0.7	53.3	39.4 (53.2X)	62.9 (1.2X)	3	33.1 (44.8X)	71.3 (1.3X)	5
3	0.4	53.3	19.7 (54.6X)	60.8 (1.1X)	5	19.7 (54.7X)	60.8 (1.1X)	5
4	0.3	53.3	10.0 (38.5X)	51.1 (1.0X)	5	10.0 (38.5X)	51.1 (1.0X)	5

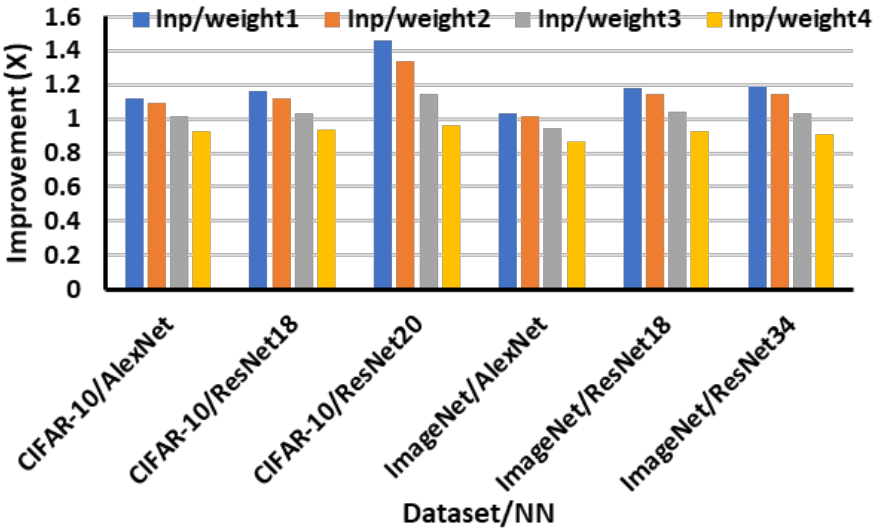
Table 5.7: Energy Efficiency (En. Eff (TOPS/J)) and Throughput (Perf. (GOPS/s)) of TULIP and an Equivalent MAC-Unit Based Benchmark Architecture for CIFAR-10 Classification. K Indicates the Number of Neurons Used in Each Cluster. Two Variants of TULIP Are Shown: One Tuned for Energy Efficiency, While the Other Is Tuned for performance.

I/W bits	Benchmark		TULIP-Q for En. Eff.			TULIP-Q for Perf		
	En. Eff.	Perf.	En. Eff.	Perf.	K	En. Eff.	Perf.	K
(a) AlexNet								
1	4.3	123.2	9.3 (2.2X)	125.4 (1.0X)	3	9.0 (2.1X)	127.5 (1.0X)	5
2	1.7	123.2	9.0 (5.2X)	116.3 (0.9X)	3	8.8 (5.1X)	124.7 (1.0X)	5
3	0.9	123.2	8.1 (9.0X)	116.3 (0.9X)	5	8.1 (9.0X)	116.3 (0.9X)	5
4	0.7	123.2	6.9 (10.4X)	106.5 (0.9X)	5	6.9 (10.4X)	106.5 (0.9X)	5
(b) ResNet18								
1	3.2	84.6	57.5 (18.1X)	97.9 (1.2X)	3	42.5 (13.4X)	100.1 (1.2X)	5
2	1.0	84.6	42.8 (41.2X)	88.3 (1.0X)	3	36.9 (35.5X)	96.6 (1.1X)	5
3	0.5	84.6	23.3 (44.8X)	87.9 (1.0X)	5	23.3 (44.8X)	87.9 (1.0X)	5
4	0.4	84.6	12.4 (33.5X)	78.3 (0.9X)	5	12.3 (33.4X)	78.3 (0.9X)	5
(c) ResNet34								
1	3.1	88.2	73.9 (23.8X)	102.3 (1.2X)	3	49.9 (16.1X)	104.8 (1.2X)	5
2	1.0	88.2	50.5 (50.5X)	91.4 (1.0X)	3	42.1 (42.1X)	100.8 (1.1X)	5
3	0.5	88.2	24.8 (49.5X)	91.0 (1.0X)	5	24.8 (49.5X)	91.0 (1.0X)	5
4	0.4	88.2	12.4 (35.5X)	80.4 (0.9X)	5	12.4 (35.5X)	80.4 (0.9X)	5

Table 5.8: Energy Efficiency (En. Eff (TOPS/J)) and Throughput (Perf. (GOPS/s)) of TULIP and an Equivalent MAC-Unit Based Benchmark Architecture for ImageNet Classification. K Indicates the Number of Neurons Used in Each Cluster. Two Variants of TULIP Are Shown: One Tuned for Energy Efficiency, While the Other Is Tuned for performance.



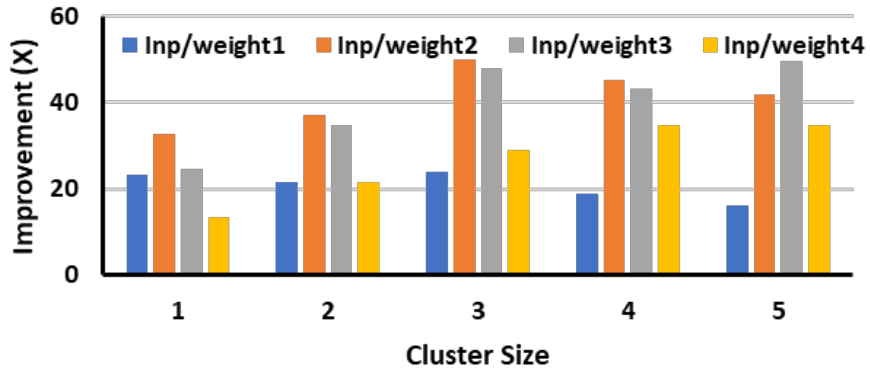
(a) Improvements in Energy Efficiency



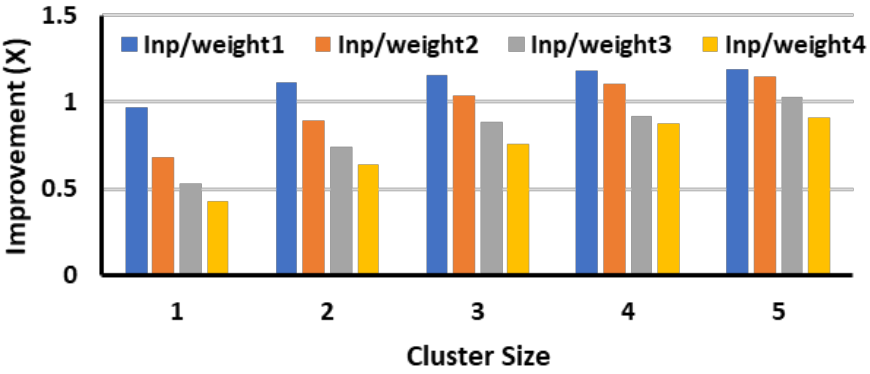
(b) Improvements in Throughput (Perf. (GOPS/s))

Figure 5.12: Improvements of TULIP-PE (Using Five Neurons per Cluster) over Equivalent MAC-Unit Based Benchmark Circuit, for Various Neural networks.

is done by changing the number of active neurons in each cluster. Based on the results presented in both the tables, TULIP shows consistent improvement in energy-efficiency over the baseline for all the neural networks.



(a) Improvements in Energy Efficiency



(b) Improvements in Throughput (Perf. (GOPS/s))

Figure 5.13: Improvements of TULIP-PE (With Varying Number of Active Neurons in Each Cluster) over Equivalent MAC-Unit Based Benchmark Circuit, for ImageNet Classification Using ResNet-34.

As shown in Figure 5.12a, TULIP consistently demonstrates an order of magnitude higher improvements in energy-efficiency for all variants of the neural networks. This is primarily attributed to the fact that TULIP realizes adders, multipliers, etc., of custom bit widths, thereby eliminating the waste incurred by conventional accumulation methods that use max-width operators in every cycle. This, coupled with the improved weight-reuse, significantly enhances the overall energy-efficiency against the baseline architecture. The corresponding throughput improvements are shown in

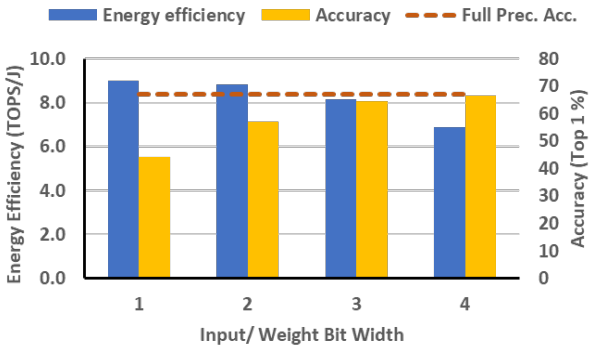
Figure 5.12b. As the bit-precision of the neural network increases, the throughput decreases. The drop in throughput is because an increase in the bit-precision increases the number of bits that need to be processed for each operation.

Figure 5.13b shows that the throughput increases as the number of neurons in each cluster increases. Although this graph is restricted to the inference of Imagenet classification using ResNet-34, this trend applies to other neural networks as well. The increase in the throughput is because the increase in the number of neurons allows each operation to execute faster on the TULIP-PE. By appropriately choosing the right configuration, it is possible to match the throughput of the Baseline (or even improve it) while gaining significant energy-efficiency improvements. The corresponding energy-efficiency improvements are shown in Figure 5.13a.

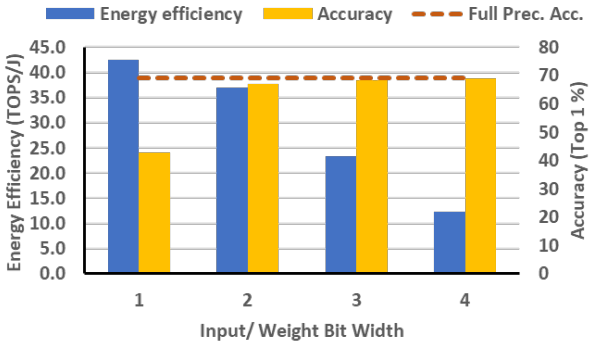
Figure 5.14 demonstrates how the TULIP architecture can be used to trade-off energy-efficiency and accuracy at runtime for neural networks used for ImageNet classification task. As the bit-precision increases, the energy-efficiency decreases but accuracy increases. Consequently, depending on the application, we can tweak the accuracy at run-time to optimize energy-efficiency and throughput. For instance, in standby mode, we can operate TULIP using 1-bit precision to get the best energy-efficiency for tolerable accuracy. Meanwhile, in high-accuracy mode improves the accuracy for a short duration at the cost of energy-efficiency.

5.8 Conclusion

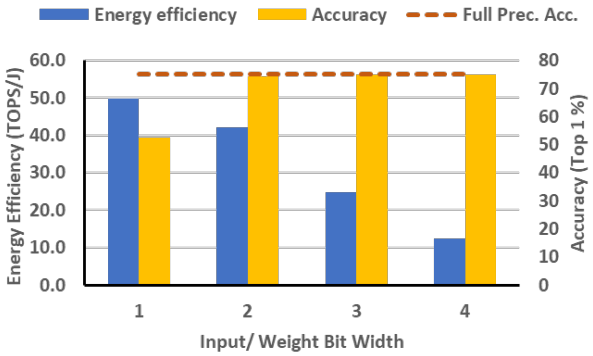
This chapter is the first implementation of TULIP, a QNN accelerator that uses current-mode binary neurons. TULIP demonstrates up to 30X-50X improvement in energy efficiency against a state-of-the-art QNN hardware accelerator without using the standard low power techniques such as voltage scaling and approximate computing. The TULIP design uses the same area as the baseline QNN architecture. It gains



(a) ImageNet/AlexNet



(b) ImageNet/ResNet 18



(c) ImageNet/ResNet 34

Figure 5.14: Trading off Energy Efficiency (En. Eff (TOPS/J)) with Accuracy (%) for ImageNet Classification Using TULIP Architecture. Full Prec. Acc. Indicates Top the 1% Accuracy When Using 32-Bit Integers and weights.

energy-efficiency because it uses optimal bit-width operators instead of max-width operators like in an accumulator. The improvements are further boosted since the processing elements (TULIP-PEs) are built using a unique arrangement of hardware neurons. These neurons enable function-reconfigurability without sacrificing performance or energy-efficiency. Due to this feature, TULIP-PEs can switch between the various operations of a QNN, such as multiplication, ReLU, etc. while preserving data locality. They have extremely low area and power footprint compared to the existing realizations of the same function. As a result, many TULIP-PEs can be deployed in the same area as an equivalent MAC unit, and run several operations in parallel. Increased parallelism enhances data reuse significantly, further improving the energy efficiency.

Chapter 6

CONCLUSION AND EXTENSIONS

The research presented in this dissertation shows promising results in terms of substantially improving the performance, power, and area of ASICs as well as FPGAs. It warrants further study of the use of perceptrons in building energy-efficient architectures. This chapter outlines some open problems and new ideas not covered in this dissertation.

6.1 Extensions

6.1.1 Threshold Logic Processing in Memory (PIM) Architectures

The processing in the memory computing paradigm has been very effective in removing the memory wall barrier in the traditional Von-Neumann architectures. The data transfers from the external main memory to the processor are two orders of magnitude slower and consumes three orders of magnitude more energy. The PIM architectures substantially reduce the data transfers on the memory channel, thereby, increasing the throughput and energy efficiency of several data-intensive applications. The data-intensive applications include large-scale encryption/decryption programs Myers (1999), large-scale graph processing, bio-informatics, machine learning, etc.

The data-intensive applications are mostly based on a basic set of operations such as bit-wise logic operation, addition, comparison, multiplications, etc. The FTL cell and TULIP-PE can perform these operations with a small area footprint and consume substantially less power than the CMOS implementations. The FTL cell

can be programmed to perform multiple threshold functions, which can be selected during runtime using control bits. A threshold logic processing element (TLPE) design using FTL cells is shown in the Figure 6.1.

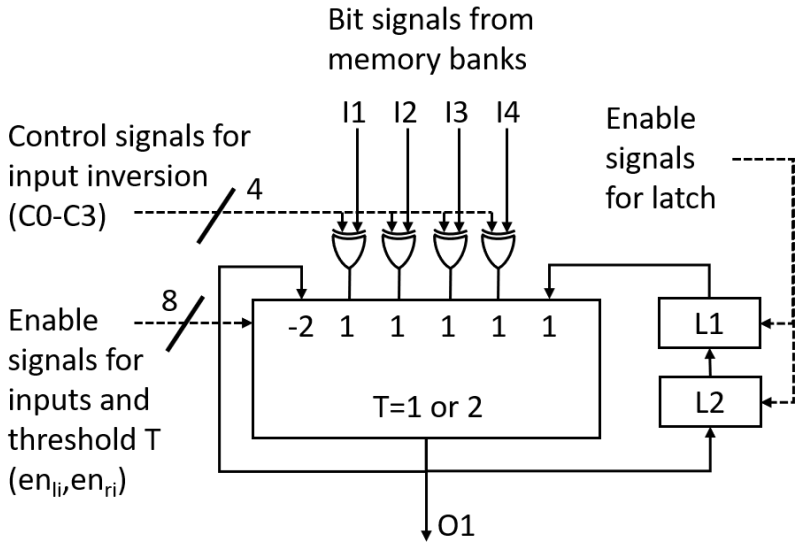


Figure 6.1: Architecture of Threshold Logic Processing element.

A TLPE consists of one FTL cell to perform computations, two latches L1 and L2 to temporarily store the output, and four XOR gates to invert the signals from the banks. The FTL cells is designed to implement a subset of the threshold function [-2,1,1,1,1,1;T], where T is selected during operation to be 1 or 2. The inputs to the processing element can be inverted using control signals *C0-C3* or optionally disabled using enable signals en_{li}. The threshold and the remaining two inputs of the FTL cell are enabled or disabled by signals en_{ri}. The TLPE is designed to perform basic logic operations (NOT, (N)OR, (N)AND, X(N)OR) and addition operation of 3 bits.

Figure 6.2 shows the integration of the threshold logic processing element (TLPE) within the DRAM memory chip. An array of TLPE (TLPEA) of size **N** is connected to four banks in the memory chip, where **N** is the number of bits in a row of the bank latched into the sense amplifiers (BLSA). There is one TLPEA for a set of **four**

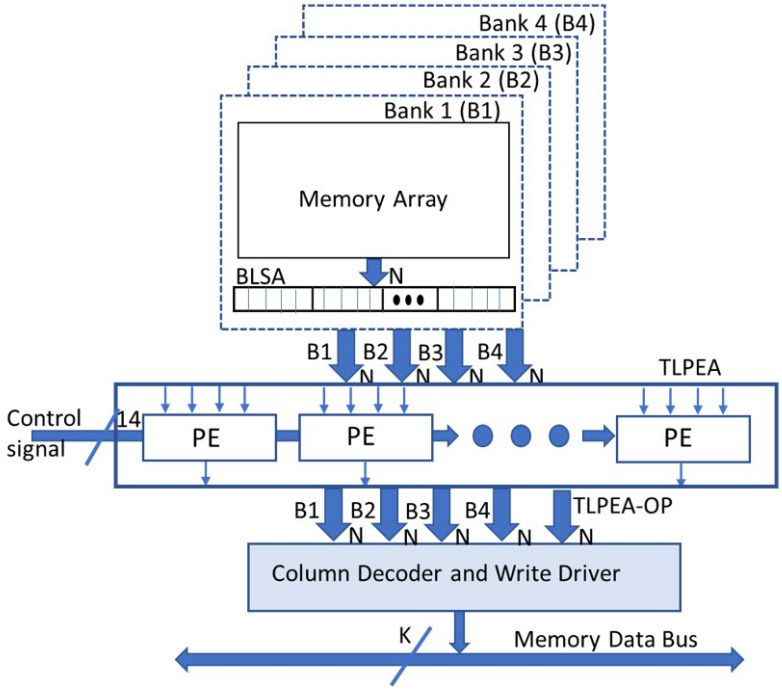


Figure 6.2: Threshold Logic Processing Element Array (TLPEA) Connected to Banks in a DRAM device.

banks in one DRAM chip. A TLPEA accepts N-bit input vectors $B1$, $B2$, $B3$ and $B4$ from all the four banks as shown in Figure 6.2. For bitwise operation in this work, at most two out of four banks are activated using a four-bank activation window (t_{FAW}) to get the operands. Consequently, only two out of four inputs to the TLPE are enabled by external control signals. The output of the TLPE array *TLPEA-OP* is connected to the column decoder and write driver block as shown in Figure 6.2. Using the control signals and the write drivers driven by the *TLPEA-OP*, the result of the computations is written back to one of the four banks. During the computation phase, the column decoder selects all the bitlines of the selected bank.

The evaluation of TLPE is carried on several practical applications such as AES, Graph Matching Index problem, and DNA sequence mapping algorithm Myers (1999). Their latency and energy comparison against the benchmark architectures is shown

Table 6.1: Latency and Energy Comparison for Executing Graph Matching Index Problem and DNA Sequence Mapping AlgorithmMyers (1999) on Different Platforms Normalized to CIDAN. Graph Matching Index Problem Is Carried out on Three Data Sets; Facebook, Amazon, Dblp

	Latency (CIDAN =1)		**Energy (CIDAN=1)**	
Workload	**Graph Matching Index**	**DNA sequence Mapping**	**Graph Matching Index**	**DNA sequence Mapping**
ReDRAM	3.24	3.14	1.96	2.12
Ambit	4.32	4.35	2.61	2.88

in Table 6.1.

The TULIP-PE is also shown to perform various operations of the artificial neural networks (binary and quantized) with a substantially smaller area footprint and power than a conventional MAC unit. Due to these advantages, the TULIP-PE can be seamlessly integrated with the DRAM memory banks. The resulting processing in memory (PIM) architecture achieves an average throughput improvement of 72X/5.4X and energy efficiency improvement of 244X/29X over CPU/GPU.

Figure 6.3(A) shows the throughput, and Figure 6.3(B) shows the energy efficiency of CIDAN-XE (PIM architecture with TULIP-PEs) in different evaluation modes versus the other architectures using the ALEXNET workload with an input image size of 224x224x3. DRISA has the highest throughput using the binary-weighted (BW) implementation of ALEXNET. CIDAN-XE in the 8-bit BW mode has a comparable throughput to DRISA with a **5.6X** higher energy efficiency. DrAcc operating in 8-bit ternary weight (TW) mode has the highest energy efficiency among the prior PIM architectures which is comparable to CIDAN-XE 8-bit TW mode. CIDAN-XE 8-bit TW has about **1.2X** higher throughput when DrAcc is operated in its high throughput configuration. In this configuration, the latency of DrAcc is 386.4s, whereas, the latency of CIDAN-XE 8-bit TW is 9.7ms which is three orders of magnitude less

than the DrAcc. In high-speed mode, DrAcc has a latency of 275ms which is still **28X** more than the CIDAN-XE 8-bit TW while the DrAcc's throughput drops to 3.63 Frames/s. CIDAN-XE 8-bit TW has a throughput of 102 Frames/s. Hence, CIDAN-XE is a high throughput and highly energy-efficient architecture and shows considerable improvements over the prior PIM architectures.

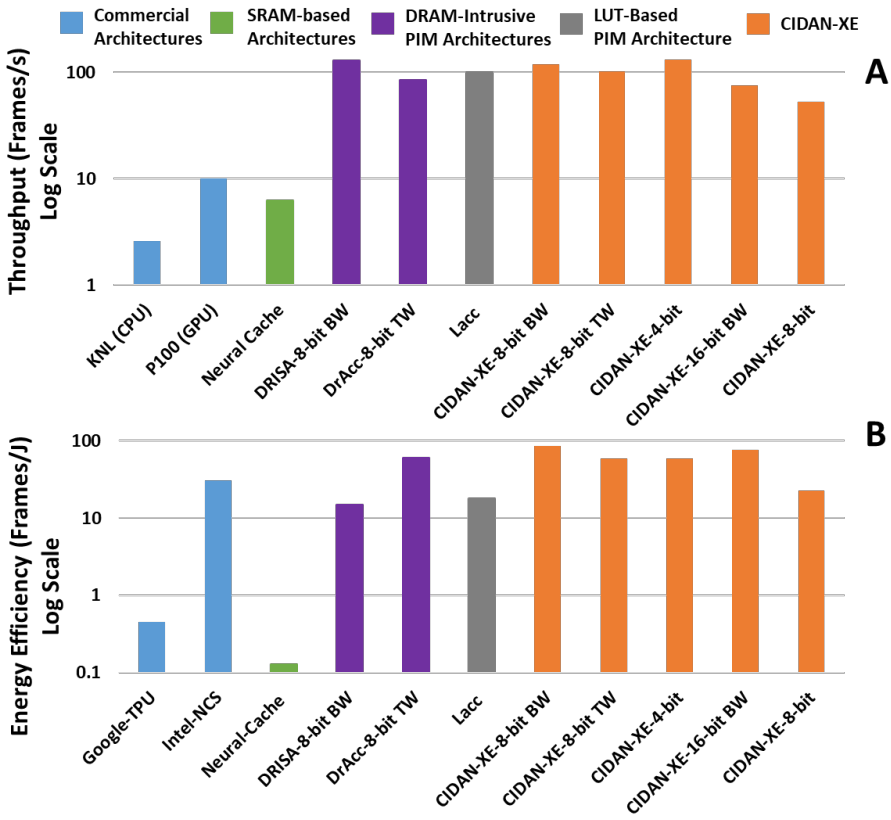


Figure 6.3: Throughput (A) and Energy-Efficiency (B) Comparison of CIDAN-XE Against the State-of-the-Art Architectures When Computing ALEXNET.

6.1.2 Circuit Obfuscation

Counterfeit integrated circuits (ICs) pose a major threat to national security and can cause significant financial losses for the US semiconductor industry, potentially reaching billions of dollars. IC counterfeiting has been occurring for many years

and has traditionally involved relatively simple techniques like recycling, relabeling, and repackaging, which do not require access to advanced manufacturing. However, there are now opportunities for attackers to infiltrate the IC supply chain at the manufacturing stage by modifying designs using advanced tools and technologies. These opportunities arose because the vast majority of ICs that are used in the US and sold by US companies globally are manufactured outside the US. In the absence of any form of protection, access to the layout database removes any and all barriers to counterfeiting or reverse engineering.

In the most recent and comprehensive survey of IC counterfeiting , there are seven types of counterfeiting: (1) recycling, (2) relabeling and repackaging, (3) illegal manufacturing, (4) low-spec components, (5) cloning and reverse engineering, (6) forgery, and (7) structural modifications. A variety of techniques have been developed to detect and prevent each of these types of counterfeiting. Among these counterfeiting methods, (3) and (7) require access to foundry and testing facilities, and (5) requires specialized equipment to deconstruct the design after fabrication through a process called delayering.

An attacker may attempt to study the design of a circuit to understand how it functions, potentially with the goal of stealing the design and/or the manufactured circuit or selling the design and/or the manufactured circuit illegally. To accomplish these objectives, the attacker must overcome any IP protection measures that may have been implemented in the ASIC. It is assumed that the attacker has access to design information, including (a) the protected netlist and knowledge of the defense techniques used in the ASIC to protect the design, as well as information about the inputs, registers, and/or memory elements that implement these defense techniques, but does not know the target design or how it is implemented in the ASIC architecture. The attacker also has access to reverse engineering tools to transform the target

layout to any desired level of abstraction. (b) Any parametric operating bounds of the IC that may be revealed from the process design kits provided by the foundry, as well as information about standard cell designs and their delay, power, area, capacitance values, leakage, design rules, simulation models, and layout. (c) A legitimate operating chip that the attacker may obtain from the market or through theft.

One way to prevent counterfeiting by a foundry or unauthorized access is through a technique known as logic locking. Logic locking is a technique to secure integrated circuits (ICs) by inserting logic gates with additional control inputs into the circuit. The set of all these control inputs serves as a secret key that must be set to a specific value for the circuit to operate correctly. The key must be stored in tamper-proof memory. Logic locking can help prevent counterfeiting or unauthorized access to the IC. However, it can also come with drawbacks, such as increased PPA requirements. Earlier logic locking techniques were found to be vulnerable to a type of attack called Boolean SATisfiability (SAT), and keys could be discovered within minutes using this method. As a result, several logic-locking techniques resistant to SAT attacks have been developed. Although these techniques make it exponentially more time-consuming to perform a SAT attack by increasing the search space for the keys, the PPA overhead is still there. Another way to secure ICs is through split manufacturing, where the untrusted foundry produces all but the top metal layers of the IC, and the fully trusted foundry produces the top metal layers. This approach can increase production time and cost and also degrade performance, power usage, and area requirements. Instead of logic locking, an alternate method is proposed in this subsection that involves the use of FTL cells, which obfuscates not only the circuit but also improves the area, power, and performance in the process. Further details are provided below.

A 5-input FTL cell can be programmed in 94,570 ways (with some input vari-

ations not counted). Its function is only determined when its flash transistors are programmed with their specific threshold voltages by the user. Therefore, the attacker (or even the foundry itself) does not know the gate's function at the time of manufacturing. For an attacker, it is nearly impossible to determine all the possible ways to program the FTL cells of a digital block, with even as few as 30 FTL cells. This is because the number of possibilities is extremely large (in the order of 10^{120}).

Another important security feature of designs that use FTL cells is the ability to include a kill switch that can render the circuit inoperable. The charge stored in the flash transistors of an FTL cell can be disrupted to disable the circuit until it is returned to the user. Once returned, the circuit can be reprogrammed to restore its original functionality. Due to the clear advantages of an FTL cell's use in circuit obfuscation, this topic warrants further study.

6.1.3 Threshold Logic Tamper-proof Mechanism for Scan-chain Locking

The function of an FTL cell can be changed post-fabrication. Using this property, we construct a state machine using a combination of FTL cells that acts as a lock for scan chains, as shown in Figure 6.4. An authenticated user unlocks this mechanism by programming each FTL cell in the state machine with the correct functionality, initializing the state, and then running the state machine for a pre-determined number of clock cycles. The final state acts as the key that unlocks the testing scan chains. Once the circuit is unlocked, the authenticated user erases the functionality of the state machine by erasing the flash transistors, thereby making the state-machine tamper-proof. For an unauthenticated user, who requires access to the circuit's test mechanisms for reverse engineering, the path to unlocking the chip is practically infeasible. This is because unlocking the chip would require identifying the correct functionality of each FTL cell, and then determining the correct initial state for the

state machine.

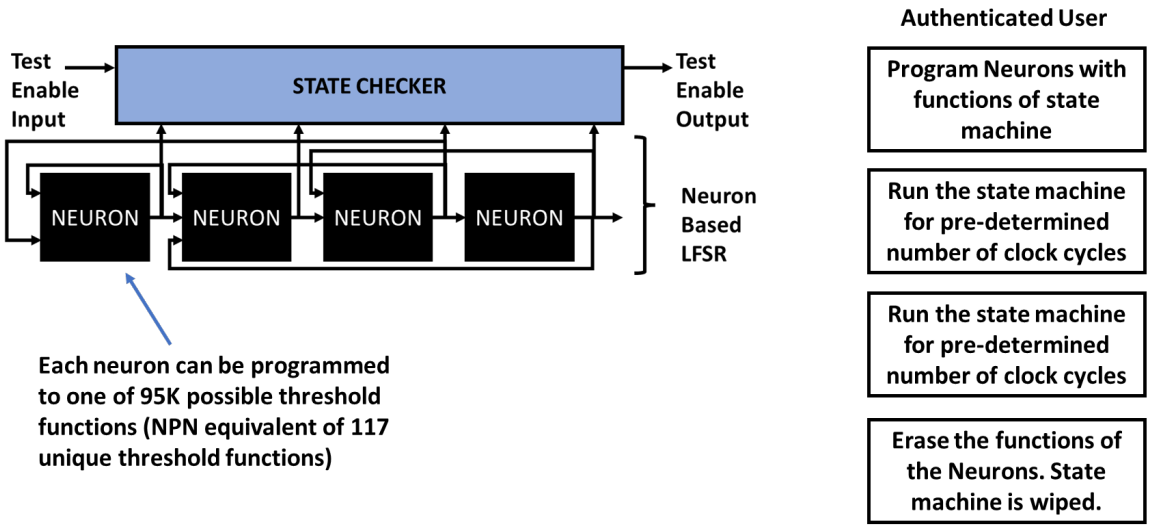


Figure 6.4: Scan Chain Locking Mechanism Using FTL Cells (Neurons)

REFERENCES

"A Highly Reliable 2-Bits/Cell Split-Gate Flash Memory Cell With a New Program-Disturbs Immune Array Configuration", IEEE Transactions on Electron Devices **61**, 7, 2350–2356, URL `http://ieeexplore.ieee.org/document/6828772/` (2014).

""The Apple M1 is the first ARM-based chipset for Macs with the fastest CPU cores and top iGPU". GSMArena.com. Retrieved 2020-11-11", (2020), URL `https://www.gsmarena.com/`.

Kao, C.-C. and Y.-T. Lai, "A routability and performance driven technology mapping algorithm for LUT based FPGA designs", in "ISCAS'99. Proceedings of the 1999 IEEE International Symposium on Circuits and Systems VLSI (Cat. No.99CH36349)", vol. 1, pp. 474–477 (IEEE, 1999), URL `http://ieeexplore.ieee.org/document/777928/`.

Kim, J. H. and J. H. Anderson, "Synthesizable FPGA fabrics targetable by the Verilog-to-Routing (VTR) CAD flow", in "2015 25th International Conference on Field Programmable Logic and Applications (FPL)", pp. 1–8 (IEEE, 2015), URL `http://ieeexplore.ieee.org/document/7293955/`.

Li, H., W.-K. Mak and S. Katkoori, "LUT-based FPGA technology mapping for power minimization with optimal depth", in "Proceedings IEEE Computer Society Workshop on VLSI 2001. Emerging Technologies for VLSI Systems", pp. 123–128 (IEEE Computer Society, 2001), URL `http://ieeexplore.ieee.org/document/923150/`.

Pan, P. and C. Liu, "Technology Mapping of Sequential Circuits for LUT-Based FPGAs for Performance", in "Fourth International ACM Symposium on Field-Programmable Gate Arrays", pp. 58–64 (IEEE, 1996), URL `http://ieeexplore.ieee.org/document/1377287/`.

Abusultan, M. and S. Khatri, "Implementing low power digital circuits using flash devices", in "2016 IEEE 34th International Conference on Computer Design (ICCD)", pp. 109–116 (2016a).

Abusultan, M. 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 (IEEE, Scottsdale, AZ, USA, 2016b), URL `http://ieeexplore.ieee.org/document/7753317/`.

Abusultan, M. and S. P. Khatri, "A Ternary-Valued, Floating Gate Transistor-Based Circuit Design Approach", in "2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)", pp. 719–724 (IEEE, Pittsburgh, PA, USA, 2016c), URL `http://ieeexplore.ieee.org/document/7560286/`.

Agrawal, V., V. Prabhakar, K. Ramkumar, L. Hinh, S. Saha, S. Samanta and R. Kapre, "In-Memory Computing array using 40nm multibit SONOS achieving 100 TOPS/W energy efficiency for Deep Neural Network Edge Inference Accelerators", in "2020 IEEE International Memory Workshop (IMW)", pp. 1–4 (2020), iSSN: 2573-7503.

Ahanonu, E., M. Marcellin and A. Bilgin, "Lossless image compression using reversible integer wavelet transforms and convolutional neural networks", in "2018 Data Compression Conference", pp. 395–395 (2018).

Ahmed, E. and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density", IEEE Transactions on Very Large Scale Integration (VLSI) Systems **12**, 3, 288–298, URL `http://ieeexplore.ieee.org/document/1281800/` (2004).

Akopyan, F., J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **34**, 10, 1537–1557 (2015).

Al Bahou, A., G. Karunaratne, R. Andri, L. Cavigelli and L. Benini, "XNORBIN: A 95 TOp/s/W hardware accelerator for binary convolutional neural networks", in "2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)", pp. 1–3 (2018).

Anderson, J. H., Q. Wang and C. Ravishankar, "Raising FPGA Logic Density Through Synthesis-Inspired Architecture", IEEE Transactions on Very Large Scale Integration (VLSI) Systems **20**, 3, 537–550, URL `http://ieeexplore.ieee.org/document/5711708/` (2012).

Andri, R., L. Cavigelli, D. Rossi and L. Benini, "YodaNN: An Architecture for Ultralow Power Binary-Weight CNN Acceleration", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **PP**, 1–1 (2017).

Annampedu, V. and M. Wagh, "Decomposition of threshold functions into bounded fan-in threshold functions", Information and Computation **227**, 84–101 (2013a).

Annampedu, V. and M. D. Wagh, "Decomposition of threshold functions into bounded fan-in threshold functions", Information and Computation **227**, 84–101 (2013b).

Appel, K. and W. Haken, *The Four-Color Problem*, pp. 153–180 (Springer New York, New York, NY, 1978), URL `https://doi.org/10.1007/978-1-4613-9435-8_7`.

Azari, E., A. Wagle, S. Khatri and S. Vrudhula, "A statistical methodology for post-fabrication weight tuning in a binary perceptron", in "2020 21st International Symposium on Quality Electronic Design (ISQED)", pp. 141–148 (IEEE, 2020).

Balasubramanian, M., S. Dave, A. Shrivastava and R. Jeyapaul, "Laser: A hardware/software approach to accelerate complicated loops on cgras", in "2018 Design, Automation and Test in Europe Conference and Exhibition (DATE)", pp. 1069–1074 (2018).

Balasubramanian, M. and A. Shrivastava, "Crimson: Compute-intensive loop acceleration by randomized iterative modulo scheduling and optimized mapping on cgras", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**, 11, 3300–3310 (2020).

Bartoli, J., V. Della Marca, J. Delalleau, A. Regnier, S. Niel, F. La Rosa, J. Postel-Pellerin and F. Lalande, "A new non-volatile memory cell based on the flash architecture for embedded low energy applications: ATW (Asymmetrical Tunnel Window)", in "2014 International Semiconductor Conference (CAS)", pp. 117–120 (IEEE, Sinaia, 2014), URL `https://ieeexplore.ieee.org/document/6966409/`.

Bayat, F. M., X. Guo, M. Klachko, N. Do, K. Likharev and D. Strukov, "Model-based high-precision tuning of NOR flash memory cells for analog computing applications", in "2016 74th Annual Device Research Conference (DRC)", pp. 1–2 (IEEE, Newark, DE, USA, 2016), URL `http://ieeexplore.ieee.org/document/7548449/`.

Beiu, V., "A survey of perceptron circuit complexity results", in "Proceedings of the International Joint Conference on Neural Networks, 2003.", vol. 2, pp. 989–994 (IEEE, 2003), URL `http://ieeexplore.ieee.org/document/1223825/`.

Beiu, V., J. Quintana and M. Avedillo, "VLSI implementations of threshold logic- a comprehensive survey", IEEE Transactions on Neural Networks **14**, 5, 1217–1243 (2003).

Berezowski, K. and S. Vrudhula, "Automatic design of binary and multiple-valued logic gates on RTD series", in "8th Euromicro Conference on Digital System Design (DSD'05)", p. 139–142 (2005).

Bersuker, G., Y. Jeon and H. Huff, "Degradation of thin oxides during electrical stress", Microelectronics Reliability **41**, 12, 1923s–1931 (2001).

Betz, V. and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research", in "Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications", FPL '97, pp. 213–222 (Springer-Verlag, 1997).

Bez, R., E. Camerlenghi, A. Modelli and A. Visconti, "Introduction to flash memory", Proceedings of the IEEE **91**, 4, 489–502 (2003).

Bhattacharjee, D., R. Devadoss and A. Chattopadhyay, "Revamp: Reram based vliw architecture for in-memory computing", in "Design, Automation Test in Europe Conference Exhibition (DATE), 2017", pp. 782–787 (IEEE, USA, 2017).

Bobba, S. and I. Hajj, "Current-mode threshold logic gates", in "Proceedings 2000 International Conference on Computer Design", p. 235–240 (IEEE Comput. Soc, 2000), URL `http://ieeexplore.ieee.org/document/878291/`.

Boboila, S. 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 (2010).

Bohossian, V., P. Hasler and J. Bruck, "Programmable neural logic", IEEE Transactions on Components, Packaging, and Manufacturing Technology: Part B **21**, 4, 346–351 (1998).

Brayton, R. and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool", in "Computer Aided Verification", edited by T. Touili, B. Cook and P. Jackson, Lecture Notes in Computer Science, pp. 24–40 (Springer, 2010).

Cai, Y., E. F. Haratsch, O. Mutlu and K. Mai, "Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling", in "2013 Design, Automation Test in Europe Conference Exhibition (DATE)", pp. 1285–1290 (2013a).

Cai, Y., E. F. Haratsch, O. Mutlu and K. Mai, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling", in "Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013", pp. 1285–1290 (IEEE Conference Publications, Grenoble, France, 2013b), URL `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6513712`.

Cai, Y., 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", p. 438–449 (IEEE, 2015), URL `https://ieeexplore.ieee.org/document/7266871`.

Canesche, M., M. Menezes, W. Carvalho, F. S. Torres, P. Jamieson, J. A. Nacif and R. Ferreira, "Traversal: A fast and adaptive graph-based placement and routing for cgras", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **40**, 8, 1600–1612 (2021).

Celinski, P., S. D. Cotofana, J. Lopez, S. F. Al-Sarawi and D. Abbott, "State of the art in CMOS threshold logic VLSI gate implementations and applications", in "VLSI Circuits and Systems", vol. 5117, pp. 53–64 (SPIE, 2003), URL `https://doi.org/10.1117/12.497792`.

Chang, K. K., A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan and O. Mutlu, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms", Proc. ACM Meas. Anal. Comput. Syst. **1**, 1, 50–92 (2017).

Chen, D., J. Cong and P. Pan, "FPGA Design Automation: A Survey", Foundations and Trends® in Electronic Design Automation **1**, 3, 195–334, URL `http://www.nowpublishers.com/article/Details/EDA-003` (2006).

Chen, E., D. Lottis, A. Driskill-Smith, D. Druist, V. Nikitin, S. Watts, X. Tang and D. Apalkov, "Non-volatile spin-transfer torque ram (stt-ram)", in "68th Device Research Conference", pp. 249–252 (IEEE, USA, 2010).

Chen, Y., J. Emer and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks", in "2016 ACM/IEEE ISCA", pp. 367–379 (2016).

Chen, Y., Z. Zhao, J. Jiang, G. He, Z. Mao and W. Sheng, "Reducing memory access conflicts with loop transformation and data reuse on coarse-grained reconfigurable architecture", in "2021 Design, Automation and Test in Europe Conference and Exhibition (DATE)", pp. 124–129 (2021).

Chen, Y.-H., J. Emer and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks", in "2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)", pp. 367–379 (IEEE, USA, 2016).

Cheng, Y., D. Wang, P. Zhou and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges", IEEE Signal Processing Magazine **35**, 1, 126–136 (2018).

Cho, I. W., B. R. Lim, J. Kim, S. S. Kim, K. C. Kim, B. J. Lee, G. J. Bae, N. I. Lee, S. H. Kim, K. W. Koh, H. Kang, M. K. Seo, S. W. Kim, S. H. Hwang, D. Y. Lee, M. C. Kim, S. D. Chae, S. A. Seo and C. W. Kim, "Full integration and characterization of Localized ONO Memory (LONOM) for embedded flash technology", in "Digest of Technical Papers. 2004 Symposium on VLSI Technology, 2004.", pp. 240–241 (2004).

Chu, Y. S., Y. H. Wang, C. Y. Wang, Y. H. Lee, A. C. Kang, R. Ranjan, W. T. Chu, T. C. Ong, H. W. Chin and K. Wu, "Split-gate flash memory for automotive embedded applications", in "2011 International Reliability Physics Symposium", pp. 6B.1.1–6B.1.5 (2011), iSSN: 1938-1891.

Chung, C.-P. and K.-S. Chang-Liao, "A Highly Scalable Single Poly-Silicon Embedded Electrically Erasable Programmable Read Only Memory With Tungsten Control Gate by Full CMOS Process", IEEE Electron Device Letters **36**, 4, 336–338, URL `http://ieeexplore.ieee.org/document/7047219/` (2015).

Cong, J. and Y. Ding, "On nominal delay minimization in LUT-based FPGA technology mapping", Integration **18**, 1, 73–94, URL `https://linkinghub.elsevier.com/retrieve/pii/0167926094900124` (1994).

Cong, J. and B. Xiao, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration", in "2011 IEEE/ACM International Symposium on Nanoscale Architectures", pp. 1–8 (2011).

Cong, J. and Yuzheng Ding, "FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **13**, 1, 1–12, URL `http://ieeexplore.ieee.org/document/273754/` (1994).

Corporation, I., "Lakefield: Hybrid cpu with foveros technology", `https://newsroom.intel.com/press-kits/lakefield/` (2019).

Cortadella, J. and S. Sapatnekar, *Static timing analysis*, pp. 133–154 (CRC Press, 2017).

Cui, Z.-Y., M.-H. Choi, Y.-S. Kim, H.-G. Lee, K.-W. Kim and N.-S. Kim, "Single poly-EEPROM with stacked MIM and n-well capacitor", Electronics Letters **45**, 3, 185 (2009).

Cuppens, R., C. Hartgring, J. Verwey and H. Peek, "An EEPROM for microprocessors and custom logic", in "1984 IEEE International Solid-State Circuits Conference. Digest of Technical Papers", vol. XXVII, pp. 268–269 (1984).

Dagan, H., A. Teman, A. Fish, E. Pikhay, V. Dayan and Y. Roizin, "A low-cost low-power non-volatile memory for RFID applications", in "2012 IEEE International Symposium on Circuits and Systems (ISCAS)", pp. 1827–1830 (2012), iSSN: 2158-1525.

Dara, C. B., T. Haniotakis and S. Tragoudas, "Delay analysis for an n-input current mode threshold logic gate", in "2012 IEEE Computer Society Annual Symposium on VLSI", p. 344–349 (2012).

Dave, S., M. Balasubramanian and A. Shrivastava, "Ramp: Resource-aware mapping for cgras", in "2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)", pp. 1–6 (2018).

Dave, S., Y. Kim, S. Avancha, K. Lee and A. Shrivastava, "Dmazerunner: Executing perfectly nested loops on dataflow accelerators", ACM Trans. Embed. Comput. Syst. **18**, 5s, URL `https://doi.org/10.1145/3358198` (2019).

Davies, M., N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning", IEEE Micro **38**, 1, 82–99 (2018).

Dechu, S., M. Goparaju and S. Tragoudas, "A Metric of Tolerance for the Manufacturing Defects of Threshold Logic Gates", in "2006 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems", p. 318–326 (2006).

Degraeve, R., B. Kaczer and G. Groeseneken, "Degradation and breakdown in thin oxide layers: mechanisms, models and reliability prediction", Microelectronics Reliability **39**, 10, 1445–1460 (1999).

Dube, A., A. Wagle, G. Singh and S. Vrudhula, "Tunable precision control for approximate image filtering in an in-memory architecture with embedded neurons", in "Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design", pp. 1–9 (2022).

Fan, D. and S. Angizi, "Energy efficient in-memory binary deep neural network accelerator with dual-mode sot-mram", in "2017 IEEE International Conference on Computer Design (ICCD)", pp. 609–612 (2017).

Farabet, C., B. Martini, B. Corda, P. Akselrod, E. Culurciello and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision", in "Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on", pp. 109–116, IEEE (IEEE, USA, 2011).

Farooq, U., Z. Marrakchi and H. Mehrez, *FPGA Architectures: An Overview* (Springer New York, 2012).

Fedorov, V. V., M. Abusultan and S. P. Khatri, "An area-efficient Ternary CAM design using floating gate transistors", in "2014 IEEE 32nd International Conference on Computer Design (ICCD)", pp. 55–60 (IEEE, Seoul, South Korea, 2014), URL `http://ieeexplore.ieee.org/document/6974662/`.

Fedorov, V. V., M. Abusultan and S. P. Khatri, "FTCAM: An Area-Efficient Flash-Based Ternary CAM Design", IEEE Transactions on Computers **65**, 8, 2652–2658, URL `http://ieeexplore.ieee.org/document/7303918/` (2016).

Fedus, W., B. Zoph and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity", Journal of Machine Learning Research **23**, 120, 1–39, URL `http://jmlr.org/papers/v23/21-0998.html` (2022).

Feng, W., J. Greene and A. Mishchenko, "Improving FPGA Performance with a S44 LUT Structure", in "Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 61—66 (ACM, 2018).

Fisher, J. A., "Trace scheduling: A technique for global microcode compaction", IEEE Trans. Comput. **30**, 7, 478–490 (1981).

Fohl, W. and D. Hemmer, "An FPGA-Based Virtual Reality Audio System", Journal of The Audio Engineering Society (2015).

Fowler, R. and L. Nordheim, "Electron Emission in Intense Electric Fields", Proc. Royal Soc. of London. Series A **119**, 781 (1928).

Furber, S., "Spinnnaker: The world's biggest noc", in "Networks-on-Chip (NoCS), 2014 Eighth IEEE/ACM International Symposium on", pp. ii–ii, IEEE (IEEE, USA, 2014).

Gao, M., J. Pu, X. Yang, M. Horowitz and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory", in "Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems", pp. 751–764, ACM (Association for Computer Machinery, USA, 2017).

Garland, J. and D. Gregg, "Low complexity multiply-accumulate units for convolutional neural networks with weight-sharing", ACM Trans. Archit. Code Optim. **15**, 3, URL `https://doi.org/10.1145/3233300` (2018).

Gogl, D., G. Burbach, H. Fiedler, M. Verbeck and C. Zimmermann, "A single-poly EEPROM cell in SIMOX technology for high-temperature applications up to 250/spl deg/C", IEEE Electron Device Letters **18**, 11, 541–543 (1997).

Goncalves, O., G. Prenat, G. Di Pendina and B. Dieny, "Non-volatile FPGAs based on spintronic devices", in "Proceedings of the 50th Annual Design Automation Conference on - DAC '13", p. 1 (ACM Press, 2013), URL http://dl.acm.org/citation.cfm?doid=2463209.2488889.

Guimarães, G. F., J. P. S. M. Lima, J. M. X. N. Teixeira, G. D. Silva, V. Teichrieb and J. Kelner, "FPGA infrastructure for the development of augmented reality applications", in "Proceedings of the 20th annual conference on Integrated circuits and systems design - SBCCI '07", p. 336 (ACM Press, 2007), URL http://dl.acm.org/citation.cfm?doid=1284480.1284568.

Guo, X., F. M. Bayat, M. Bavandpour, M. Klachko, M. R. Mahmoodi, M. Prezioso, K. K. Likharev and D. B. Strukov, "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR flash memory technology", in "2017 IEEE International Electron Devices Meeting (IEDM)", pp. 6.5.1–6.5.4 (IEEE, San Francisco, CA, USA, 2017a), URL http://ieeexplore.ieee.org/document/8268341/.

Guo, X., F. M. Bayat, M. Bavandpour, M. Klachko, M. M.R., M. Prezioso, K. Likharev and D. Strukov, "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR flash memory technology", in "2017 IEEE International Electron Devices Meeting (IEDM)", pp. 6.5.1–6.5.4 (2017b).

Gupta, P. and N. Jha, "An Algorithm for Nanopipelining of RTD-Based Circuits and Architectures", IEEE Transactions On Nanotechnology **4**, 2, 159–167 (2005).

Ha, H., "Understanding and improving the energy efficiency of dram, phd thesis, stanford university", URL https://searchworks.stanford.edu/view/12819402 (2018).

Ha, H., A. Pedram, S. Richardson, S. Kvatinsky and M. Horowitz, "Improving energy efficiency of dram by exploiting half page row access", in "The 49th Annual IEEE/ACM International Symposium on Microarchitecture", MICRO-49 (IEEE Press, USA, 2016).

Haj-Yahya, J., E. Rotem, A. Mendelson and A. Chattopadhyay, "A comprehensive evaluation of power delivery schemes for modern microprocessors", in "20th International Symposium on Quality Electronic Design (ISQED)", pp. 123–130 (IEEE, USA, 2019).

Hamzeh, M., A. Shrivastava and S. Vrudhula, "Epimap: Using epimorphism to map applications on cgras", in "Proceedings of the 49 Design Automation Conference (DAC)", (San Diego, CA, 2012).

Hamzeh, M., A. Shrivastava and S. Vrudhula, "Branch-aware loop mapping on cgras", in "2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)", pp. 1–6 (2014).

Hang-Ting Lue, Tzu-Hsuan Hsu, Szu-Yu Wang, Erh-Kun Lai, Kuang-Yeu Hsieh, Rich Liu and Chih-Yuan Lu, "study of incremental step pulse programming (ISPP) and STI edge effect of BE-SONOS NAND Flash", in "2008 IEEE International Reliability Physics Symposium", pp. 693–694 (2008), iSSN: 1938-1891.

He, K., X. Zhang, S. Ren and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification", 2015 IEEE International Conference on Computer Vision (ICCV) pp. 1026–1034 (2015).

Henry, D., B. Kuszmaul and V. Viswanath, "The ultrascalar processor-an asymptotically scalable superscalar microarchitecture", in "Proceedings 20th Anniversary Conference on Advanced Research in VLSI", pp. 256–273 (IEEE, USA, 1999).

Hinton, G., L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury and T. Sainath, "Deep neural networks for acoustic modeling in speech recognition", IEEE Signal Processing Magazine **29**, 82–97 (2012).

Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations", J. Mach. Learn. Res. **18**, 1, 6869–6898 (2017).

Hunt, K., D. Sbarbaro, R. Zbikowski and P. Gawthrop, "Neural networks for control systems survey", Automatica **28**, 6, 1083 – 1112, URL `http://www.sciencedirect.com/science/article/pii/000510989290053I` (1992).

Jouppi, N. P., C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit", in "Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on", pp. 1–12, IEEE (IEEE, USA, 2017).

Jung, S.-G., K.-W. Lee, K.-S. Kim, S.-W. Shin, S.-S. Lee, J.-C. Om, G.-H. Bae and J.-H. Lee, "Modeling of $V_{\rm th}$ Shift in nand Flash-Memory Cell Device Considering Crosstalk and Short-Channel Effects", IEEE Transactions on Electron Devices **55**, 4, 1020–1026, URL `http://ieeexplore.ieee.org/document/4475401/` (2008).

Kaya, S., H. Hamed, D. Ting and G. Creech, "Reconfigurable threshold logic gates with nanoscale DG-MOSFETs", Solid-State Electronics **51**, 10, 1301–1307, URL `https://linkinghub.elsevier.com/retrieve/pii/S0038110107002857` (2007a).

Kaya, S., H. F. Hamed, D. T. Ting and G. Creech, "Reconfigurable threshold logic gates with nanoscale dg-mosfets", Solid-State Electronics **51**, 10, 1301–1307 (2007b).

Khan, F., M. S. Han, D. Moy, R. Katz, L. Jiang, E. Banghart, N. Robson, T. Kirihata, J. C. S. Woo and S. S. Iyer, "Design Optimization and Modeling of Charge Trap Transistors (CTTs) in 14 nm FinFET Technologies", IEEE Electron Device Letters **40**, 7, 1100–1103, URL `https://ieeexplore.ieee.org/document/8726301/` (2019a).

Khan, F., D. Moy, D. Anand, E. Schroeder, R. Katz, L. Jiang, E. Banghart, N. Robson and T. Kirihata, "Turning Logic Transistors into Secure, Multi-Time Programmable, Embedded Non-Volatile Memory Elements for 14 nm FINFET Technologies and Beyond", in "2019 Symposium on VLSI Technology", pp. T116—T117 (IEEE, 2019b), URL https://ieeexplore.ieee.org/document/8776510/.

Khatri, S. P., S. Vrudhula, M. Abusultan, K. Bharathi, S.-W. Chu, C.-Y. Lee, K. R. Scott, G. Singh and A. Wagle, "Flash: A "forgotten" technology in vlsi design", in "Frontiers of Quality Electronic Design (QED)", pp. 67–136 (Springer, Cham, 2023).

Kim, M., J. Kim, G. Park, L. Everson, H. Kim, S. Song, S. Lee and C. H. Kim, "A 68 Parallel Row Access Neuromorphic Core with 22K Multi-Level Synapses Based on Logic-Compatible Embedded Flash Memory Technology", in "2018 IEEE International Electron Devices Meeting (IEDM)", pp. 15.4.1–15.4.4 (IEEE, San Francisco, CA, 2018), URL https://ieeexplore.ieee.org/document/8614599/.

Kim, S., J. Lee, S. Kang, J. Lee and H.-J. Yoo, "A power-efficient cnn accelerator with similar feature skipping for face recognition in mobile devices", IEEE Transactions on Circuits and Systems I: Regular Papers **67**, 4, 1181–1193 (2020).

Knag, P. C., G. K. Chen, H. E. Sumbul, R. Kumar, M. A. Anders, H. Kaul, S. K. Hsu, A. Agarwal, M. Kar, S. Kim and et al., "A 617 tops/w all digital binary neural network accelerator in 10nm finfet cmos", in "2020 IEEE Symposium on VLSI Circuits", p. 1–2 (IEEE, 2020), URL https://ieeexplore.ieee.org/document/9162949/.

Kono, T., T. Ito, T. Tsuruda, T. Nishiyama, T. Nagasawa, T. Ogawa, Y. Kawashima, H. Hidaka and T. Yamauchi, "40-nm Embedded Split-Gate MONOS (SG-MONOS) Flash Macros for Automotive With 160-MHz Random Access for Code and Endurance Over 10 M Cycles for Data at the Junction Temperature of 170 $ˆ{\circ}$C", IEEE Journal of Solid-State Circuits **49**, 1, 154–166, URL http://ieeexplore.ieee.org/document/6612753/ (2014).

Krizhevsky, A., I. Sutskever and G. Hinton, "Imagenet classification with deep convolutional neural networks", in "Advances in Neural Information Processing Systems 25", edited by F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, pp. 1097–1105 (Curran Associates, Inc., 2012).

Ku, D. and G. De Micheli, "Relative scheduling under timing constraints", in "Proceedings of the 27th ACM/IEEE Design Automation Conference", DAC '90, p. 59–64 (Association for Computing Machinery, New York, NY, USA, 1991), URL https://doi.org/10.1145/123186.123227.

Kulkarni, N. and S. Vrudhula, "Efficient Enumeration of Unidirectional Cuts for Technology Mapping of Boolean Networks", (2016).

Kulkarni, N., J. Yang, J. Seo and S. Vrudhula, "Reducing Power, Leakage, and Area of Standard-Cell ASICs Using Threshold Logic Flip-Flops", IEEE Transactions on Very Large Scale Integration (VLSI) Systems **24**, 9, 2873–2886, URL http://ieeexplore.ieee.org/document/7430367/ (2016a).

Kulkarni, N., J. Yang, J.-S. Seo and S. Vrudhula, "Reducing power, leakage, and area of standard-cell asics using threshold logic flip-flops", IEEE Transactions on Very Large Scale Integration (VLSI) Systems **24**, 9, 2873–2886 (2016b).

Kulkarni, N., J. Yang and S. Vrudhula, "A fast, energy efficient, field programmable threshold-logic array", in "2014 International Conference on Field-Programmable Technology (FPT)", pp. 300–305 (IEEE, 2014), URL `http://ieeexplore.ieee.org/document/7082804/`.

Kumar, T. N., H. A. F. Almurib and F. Lombardi, "A novel design of a memristor-based look-up table (LUT) for FPGA", in "2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)", pp. 703–706 (IEEE, 2014), URL `http://ieeexplore.ieee.org/document/7032878/`.

Kuon, I., R. Tessier and J. Rose, *FPGA architecture: survey and challenges*, no. 2,2 in Foundations and trends in electronic design automation (Now Publ, 2008).

Lageweg, C. R., S. D. Cotofana and S. Vassiliadis, "A full adder implementation using set based linear threshold gates", in "Proceedings 9th IEEE International conference on electronics, circuits and systems - ICECS 2002", pp. 665–669 (2002).

Landwehr, B., P. Marwedel and R. Dömer, "Oscar: optimum simultaneous scheduling, allocation and resource binding based on integer programming", pp. 90–95 (1994).

Lee, B. C., E. Ipek, O. Mutlu and D. Burger, "Phase change memory architecture and the quest for scalability", Commun. ACM **53**, 7, 99–106 (2010).

Lee, W., "Tutorial: Design and optimization of power delivery networks", IEIE Transactions on Smart Processing and Computing **5**, 349–357 (2016).

Lee, Y. K., B. Seo, T.-K. Yu, B. Lee, E. Kim, C. Jeon, W. Park, Y. Kim, D. Lee, H. Lee and S. Cho, "A 45-nm logic compatible 4Mb-split-gate embedded flash with 1M-cycling-endurance", in "2014 IEEE 6th International Memory Workshop (IMW)", pp. 1–4 (IEEE, Taipei, Taiwan, 2014), URL `http://ieeexplore.ieee.org/document/6849369/`.

Li, J., G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu and X. Li, "Smartshuttle: Optimizing off-chip memory accesses for deep learning accelerators", in "2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)", pp. 343–348, IEEE (IEEE, USA, 2018).

Li, P. and Y. Luo, "P4gpu: Accelerate packet processing of a p4 program with a cpu-gpu heterogeneous architecture", in "2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)", pp. 125–126 (IEEE, USA, 2016).

Liang, M. and X. Hu, "Recurrent convolutional neural network for object recognition", in "The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", (2015).

Lin, J. Y., D. Chen and J. Cong, "Optimal simultaneous mapping and clustering for FPGA delay optimization", in "Proceedings of the 43rd annual conference on Design automation - DAC '06", p. 472 (ACM Press, 2006), URL http://portal.acm.org/citation.cfm?doid=1146909.1147035.

Lipatov, I. A. and I. V. Tiunov, "Performance-driven technology mapping for XC5510 family FPGAs", in "2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)", pp. 477–479 (IEEE, 2017), URL http://ieeexplore.ieee.org/document/7910595/.

Liu, H. J., "Archipelago - An Open Source FPGA with Toolflow Support", in "Thesis", (UC Berkeley, 2014), URL https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-43.pdf.

Liu, J., B. Jaiyen, R. Veras and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh", in "Proceedings of the 39th Annual International Symposium on Computer Architecture", ISCA '12, p. 1–12 (IEEE Computer Society, USA, 2012).

Liu, Z., P. N. Whatmough, Y. Zhu and M. Mattina, "S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration", in "2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)", pp. 573–586 (IEEE Computer Society, Los Alamitos, CA, USA, 2022), URL https://doi.ieeecomputersociety.org/10.1109/HPCA53966.2022.00049.

Luo, C., J. Diao and C. Chen, "Fullreuse: A novel reram-based cnn accelerator reusing data in multiple levels", in "2020 IEEE 5th International Conference on Integrated Circuits and Microsystems (ICICM)", pp. 177–183 (2020).

Luo, J.-H., J. Wu and W. Lin, "Thinet: A filter level pruning method for deep neural network compression", in "2017 IEEE International Conference on Computer Vision (ICCV)", pp. 5068–5076 (2017).

Luo, L. Q., Z. Q. Teo, Y. J. Kong, F. X. Deng, J. Q. Liu, F. Zhang, X. S. Cai, K. M. Tan, K. Y. Lim, P. Khoo, S. M. Jung, S. Y. Siah, D. Shum, C. M. Wang, J. C. Xing, G. Y. Liu, Y. Diao, G. M. Lin, L. Tee, S. M. Lemke, P. Ghazavi, X. Liu, N. Do, K. L. Pey and K. Shubhakar, "Functionality Demonstration of a High-Density 2.5V Self-Aligned Split-Gate NVM Cell Embedded into 40nm CMOS Logic Process for Automotive Microcontrollers", in "2016 IEEE 8th International Memory Workshop (IMW)", pp. 1–4 (IEEE, Paris, France, 2016), URL http://ieeexplore.ieee.org/document/7495271/.

López-García, J., J. Fernández-Ramos and A. Gago-Bohórquez, "A balanced capacitive threshold-logic gate", Analog Integrated Circuits and Signal Processing **40**, 1, 61–69 (2004).

Ma, S., M. Donato, S. K. Lee, D. Brooks and G.-Y. Wei, "Fully-CMOS Multi-Level Embedded Non-Volatile Memory Devices With Reliable Long-Term Retention for Efficient Storage of Neural Network Weights", IEEE Electron Device Letters **40**, 9, 1403–1406, URL https://ieeexplore.ieee.org/document/8767952/ (2019a).

Ma, Y., Y. Cao, S. Vrudhula and J.-s. Seo, "An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks", in "2017 27th International Conference on Field Programmable Logic and Applications (FPL)", pp. 1–8 (IEEE, 2017), URL http://ieeexplore.ieee.org/document/8056824/.

Ma, Y., Y. Cao, S. Vrudhula and J.-s. Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA", IEEE Transactions on Very Large Scale Integration (VLSI) Systems **26**, 7, 1354–1367, URL https://ieeexplore.ieee.org/document/8330049/ (2018).

Ma, Y., Y. Cao, S. Vrudhula and J.-S. Seo, "Performance Modeling for CNN Inference Accelerators on FPGA", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems TCAD (2019b).

McCulloch, W. S. and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", in "Neurocomputing: Foundations of Research", edited by J. A. Anderson and E. Rosenfeld (MIT Press, 1988), URL http://dl.acm.org/citation.cfm?id=65669.104377.

Mehonic, A., A. Sebastian, B. Rajendran, O. Simeone, E. Vasilaki and A. J. Kenyon, "Memristors – from in-memory computing, deep learning acceleration, spiking neural networks, to the future of neuromorphic and bio-inspired computing", (2020).

Mehri, M. and N. Masoumi, "A thorough investigation into active and passive shielding methods for nano-VLSI interconnects against EMI and crosstalk", AEU - International Journal of Electronics and Communications **69**, 9, 1199–1207 (2015).

Menshawy, R. S., A. H. Yousef and A. Salem, "Code smells and detection techniques: A survey", in "2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)", pp. 78–83 (2021).

Minnick, R. C., "Linear-input logic", IRE Trans. Electron. Comput. **10**, 6–16 (1961).

Mitani, H., K. Matsubara, H. Yoshida, T. Hashimoto, H. Yamakoshi, S. Abe, T. Kono, Y. Taito, T. Ito, T. Krafuji, K. Noguchi, H. Hidaka and T. Yamauchi, "7.6 A 90nm embedded 1T-MONOS flash macro for automotive applications with 0.07mJ/8kB rewrite energy and endurance over 100M cycles under Tj of 175°C", in "2016 IEEE International Solid-State Circuits Conference (ISSCC)", pp. 140–141 (IEEE, San Francisco, CA, USA, 2016), URL http://ieeexplore.ieee.org/document/7417946/.

Moolchandani, D., A. Kumar and S. R. Sarangi, "Accelerating CNN Inference on ASICs: A Survey", Journal of Systems Architecture **113**, 101887, URL https://linkinghub.elsevier.com/retrieve/pii/S1383762120301612 (2021).

Moons, B., D. Bankman, L. Yang, B. Murmann and M. Verhelst, "Binareye: An always-on energy-accuracy-scalable binary cnn processor with all memory on chip in 28nm cmos", in "2018 IEEE Custom Integrated Circuits Conference (CICC)", pp. 1–4 (2018).

Mozaffari, S. and S. Tragoudas, "Maximizing the Number of Threshold Logic Functions Using Resistive Memory", IEEE Transactions on Nanotechnology **17**, 5, 897–905, URL https://ieeexplore.ieee.org/document/8329555/ (2018).

Mozaffari, S., S. Tragoudas and T. Haniotakis, "A Generalized Approach to Implement Efficient CMOS-Based Threshold Logic Functions", IEEE Transactions on Circuits and Systems I: Regular Papers **65**, 3, 946–959, URL http://ieeexplore.ieee.org/document/8118301/ (2018).

Mullin, A. A., "Threshold Logic: A Synthesis Approach", SIAM Review **8**, 3, 405–406 (1966).

Muroga, S., *Threshold Logic and its Applications* (Wiley-Interscience New York, 1971a).

Muroga, S., *Threshold Logic and its Applications* (1971b).

Muroga, S., *Threshold Logic and its Applications* (John Wiley & Sons, 1971c).

Mutlu, O., S. Ghose, J. Gómez-Luna and R. Ausavarungnirun, "A modern primer on processing in memory", CoRR **abs/2012.03112**, URL https://arxiv.org/abs/2012.03112 (2020).

Myers, G., "A fast bit-vector algorithm for approximate string matching based on dynamic programming", J. ACM **46**, 3, 395–415, URL https://doi.org/10.1145/316542.316550 (1999).

Nagel, M., M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen and T. Blankevoort, "A white paper on neural network quantization", URL https://arxiv.org/abs/2106.08295 (2021).

Nakahara, H., H. Yonekawa, T. Sasao, H. Iwamoto and M. Motomura, "A memory-based realization of a binarized deep convolutional neural network", in "2016 International Conference on Field-Programmable Technology (FPT)", pp. 277–280 (2016).

Neutzling, A., J. Matos, A. Mishchenko, A. Reis and R. Ribas, "Effective Logic Synthesis for Threshold Logic Circuit Design", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**, 5, 926–937, URL https://ieeexplore.ieee.org/document/8355999/ (2019).

Nii, K., Y. Taniguchi and K. Okuyama, "A Cost-Effective Embedded Nonvolatile Memory with Scalable LEE Flash®-G2 SONOS for Secure IoT and Computing-in-Memory (CiM) Applications", in "2020 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)", pp. 1–4 (2020).

Nukala, N. S., N. Kulkarni and S. Vrudhula, "Spintronic threshold logic array (STLA) - a compact, low leakage, non-volatile gate array architecture", in "Proceedings of the 2012 IEEE/ACM International Symposium on Nanoscale Architectures - NANOARCH '12", pp. 188–195 (ACM Press, 2012), URL http://dl.acm.org/citation.cfm?doid=2765491.2765525.

Nurvitadhi, E., G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra and G. Boudoukh, "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?", in "Proceedings of the 2017 ACM/SIGDA", FPGA '17, pp. 5–14 (ACM, New York, NY, USA, 2017), URL http://doi.acm.org/10.1145/3020078.3021740.

Ogura, T., N. Ogura, M. Kirihara, Ki Tae Park, Y. Baba, M. Sekine and K. Shimeno, "Embedded twin MONOS flash memories with 4 ns and 15 ns fast access times", in "2003 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.03CH37408)", pp. 207–210 (Japan Soc. Appl. Phys, Kyoto, Japan, 2003), URL http://ieeexplore.ieee.org/document/1221204/.

Ozdemir, H., A. Kepkep, B. Pamir, Y. Leblebici and U. Cilingiroglu, "A capacitive threshold-logic gate", IEEE Journal of Solid-State Circuits **31**, 8, 1141–1150 (1996).

Padure, M., S. Cotofana, C. Dan, M. Bodea and S. Vassiliadis, "A new latch-based threshold logic family", in "2001 International Semiconductor Conference. CAS 2001 Proceedings (Cat. No.01TH8547)", p. 531–534 (IEEE, 2001), URL http://ieeexplore.ieee.org/document/967522/.

Padure, M., S. Cotofana, C. Dan, S. Vassiliadis and M. Bodea, "Compact delay modeling of latch-based threshold logic gates", in "Proceedings. International Semiconductor Conference", vol. 2, p. 317–320 vol.2 (2002).

Padure, M., S. Cotofana and S. Vassiliadis, "Design and experimental results of a cmos flip-flop featuring embedded threshold logic", in "Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.", vol. 5, pp. V–253–V–256 (IEEE, 2003), URL http://ieeexplore.ieee.org/document/1206246/.

Palacharla, S., N. Jouppi and J. Smith, "Complexity-effective superscalar processors", in "Conference Proceedings. The 24th Annual International Symposium on Computer Architecture", pp. 206–218 (IEEE, USA, 1997).

Park, H., D. Kim, J. Ahn and S. Yoo, "Zero and data reuse-aware fast convolution for deep neural networks on gpu", in "2016 International Conference on Hardware/-Software Codesign and System Synthesis (CODES+ISSS)", pp. 1–10 (2016).

Park, S., S. Kim and B. Lee, "Development of 2T-SONOS Cell Using a Contamination-Free Process Integration for a Highly Reliable Code Storage eNVM", IEEE Transactions on Electron Devices **67**, 3, 922–928 (2020).

Park, S.-K., H.-M. Song, N.-Y. Kim, I.-W. Cho and K.-D. Yoo, "Novel Select Gate Lateral Coupling Single Poly eNVM for an HVC-MOS Process", IEEE Electron Device Letters **35**, 3, 351–353, URL http://ieeexplore.ieee.org/document/6728616/ (2014).

Perricone, R., I. Ahmed, Z. Liang, M. G. Mankalale, X. S. Hu, C. H. Kim, M. Niemier, S. S. Sapatnekar and J. Wang, "Advanced spintronic memory and logic for non-volatile processors", in "Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017", pp. 972–977 (2017), URL http://ieeexplore.ieee.org/document/7927132/.

Pisarchyk, Y. and J. Lee, "Efficient memory management for deep neural net inference", (2020).

Rajendran, J., H. Manem, R. Karri and G. S. Rose, "Memristor based programmable threshold logic array", in "2010 IEEE/ACM International Symposium on Nanoscale Architectures", p. 5–10 (2010).

Ramos, J. F., J. L. García, S. C. Martín and A. G. Bohórquez, "A new theorem in threshold logic and its application to multioperand binary adders", International Journal of Computer Mathematics **80**, 11, 1363–1372 (2003).

Raoux, S., G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung and C. H. Lam, "Phase-change random access memory: A scalable technology", IBM Journal of Research and Development **52**, 4.5, 465–479 (2008).

Rastegari, M., V. Ordonez, J. Redmon and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks", vol. 9908, pp. 525–542 (2016).

Raszka, J., M. Advani, V. Tiwari, L. Varisco, N. Hacobian, A. Mittal, M. Han, A. Shirdel and A. Shubat, "Embedded flash memory for security applications in a 0.13 um CMOS logic process", in "2004 IEEE International Solid-State Circuits Conference (IEEE Cat. No.04CH37519)", pp. 46–512 (IEEE, San Francisco, CA, USA, 2004), URL `http://ieeexplore.ieee.org/document/1332586/`.

Ren, S., K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks", in "Advances in Neural Information Processing Systems 28", edited by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett, pp. 91–99 (Curran Associates, Inc., 2015).

Research, I., "Iedm 2020: Advances in memory, analog ai and interconnects point to the future of hybrid cloud and ai", (2020).

Richter, D., *Fundamentals of Non-Volatile Memories*, p. 5–110, Springer Series in Advanced Microelectronics (Springer Netherlands, 2014).

Rodriguez-Villegas, E., J. M. Quintana, M. J. Avedillo and A. Rueda, "High-speed low-power logic gates using floating gates", in "2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)", vol. 5, p. V–V (2002).

Roizin, Y., E. Aloni, A. Birman, V. Dayan, A. Fenigstein, D. Nahmad, E. Pikhay and D. Zfira, "C-Flash: An Ultra-Low Power Single Poly Logic NVM", in "2008 Joint Non-Volatile Semiconductor Memory Workshop and International Conference on Memory Technology and Design", pp. 90–92 (2008), iSSN: 2159-4864.

Rosenblatt, F., "The perceptron: A probabilistic model for information storage and organization in the brain.", Psychological Review (1958).

Rothenbuhler, A., T. Tran, E. Smith, V. Saxena and K. Campbell, "Reconfigurable threshold logic gates using memristive devices", Journal of Low Power Electronics and Applications **3**, 2, 174–193 (2013).

Sampath, M., P. S. Mane and C. K. Ramesha, "Hybrid CMOS-memristor based FPGA architecture", in "2015 International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA)", pp. 1–6 (IEEE, 2015), URL http://ieeexplore.ieee.org/document/7050461/.

Scott, K. R. and S. P. Khatri, "A flash-based digital to analog converter for low power applications", in "2022 IEEE 40th International Conference on Computer Design (ICCD)", pp. 1–8 (2022).

Scott, K. R. and S. P. Khatri, "An extremely low-voltage floating gate artificial neuron", in "2023 IEEE International Symposium on Circuits and Systems (ISCAS)", pp. 1–4 (2023).

Serrano, G., "High performance analog circuit design using floating-gate techniques", (2007).

Shi, F., Z. Yan and M. Wagh, "An enhanced multiway sorting network based on n-sorters", in "2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)", p. 60–64 (IEEE, 2014), URL http://ieeexplore.ieee.org/document/7032078/.

Shukuri, S., N. Ajika, M. Mihara, K. Kobayashi, T. Endoh and M. Nakashima, "A 60nm NOR Flash Memory Cell Technology Utilizing Back Bias Assisted Band-to-Band Tunneling Induced Hot-Electron Injection (B4-Flash)", in "2006 Symposium on VLSI Technology, 2006. Digest of Technical Papers.", pp. 15–16 (IEEE, Honolulu, HI, USA, 2006), URL http://ieeexplore.ieee.org/document/1705194/.

Simon Tam, Ping-Keung Ko and Chenming Hu, "Lucky-electron model of channel hot-electron injection in mosfet's", IEEE Transactions on Electron Devices **31**, 9, 1116–1125 (1984).

Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", URL https://arxiv.org/abs/1409.1556 (2014).

Singh, G., A. Wagle, S. Khatri and S. Vrudhula, "Cidan-xe: Computing in dram with artificial neurons", vol. 3, p. 834146 (Frontiers Media SA, 2022).

Singh, G., A. Wagle, S. Vrudhula and S. Khatri, "Cidan: Computing in dram with artificial neurons", in "2021 IEEE 39th International Conference on Computer Design (ICCD)", pp. 349–356 (IEEE, 2021).

Siu, K., V. Roychowdhury and T. Kailath, "Depth-size tradeoffs for neural computation", IEEE Transactions on Computers **40**, 12, 1402–1412 (1991).

Siu, K., V. Roychowdhury and T. Kailath, *Discrete Neural Computation: A Theoretical Foundation* (Prentice-Hall, Inc., 1995).

Sohi, G. S., S. E. Breach and T. N. Vijaykumar, "Multiscalar processors", in "Proceedings of the 22nd Annual International Symposium on Computer Architecture", ISCA '95, p. 414–425 (Association for Computing Machinery, New York, NY, USA, 1995), URL https://doi.org/10.1145/223982.224451.

Song, S., K. C. Chun and C. H. Kim, "A logic-compatible embedded flash memory featuring a multi-story high voltage switch and a selective refresh scheme", in "2012 Symposium on VLSI Circuits (VLSIC)", pp. 130–131 (2012), iSSN: 2158-5636.

Song, S.-H., K. C. Chun and C. H. Kim, "A Logic-Compatible Embedded Flash Memory for Zero-Standby Power System-on-Chips Featuring a Multi-Story High Voltage Switch and a Selective Refresh Scheme", IEEE Journal of Solid-State Circuits **48**, 5, 1302–1314, URL `http://ieeexplore.ieee.org/document/6472737/` (2013).

Srinivasan, V., G. J. Serrano, J. Gray and P. Hasler, "A Precision CMOS Amplifier Using Floating-Gate Transistors for Offset Cancellation", IEEE Journal of Solid-State Circuits **42**, 2, 280–291, URL `http://ieeexplore.ieee.org/document/4077175/` (2007).

Strubell, E., A. Ganesh and A. McCallum, "Energy and policy considerations for modern deep learning research", Proceedings of the AAAI Conference on Artificial Intelligence **34**, 09, 13693–13696, URL `https://ojs.aaai.org/index.php/AAAI/article/view/7123` (2020).

Sun, X., X. Peng, P. Chen, R. Liu, J. Seo and S. Yu, "Fully parallel rram synaptic array for implementing binary neural network with (+1, -1) weights and (+1, 0) neurons", in "2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)", (2018).

Sze, V., Y.-H. Chen, T.-J. Yang and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey", Proceedings of the IEEE **105**, 12, 2295–2329 (2017).

Taito, Y., M. Nakano, H. Okimoto, D. Okada, T. Ito, T. Kono, K. Noguchi, H. Hidaka and T. Yamauchi, "7.3 A 28nm embedded SG-MONOS flash macro for automotive achieving 200MHz read operation and 2.0MB/S write throughput at Ti, of 170°C", in "2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers", pp. 1–3 (IEEE, San Francisco, CA, USA, 2015), URL `http://ieeexplore.ieee.org/document/7062961/`.

Tang, W.-C., W.-H. Lo, Y.-L. Wu and S.-C. Chang, "FPGA Technology Mapping Optimization by Rewiring Algorithms", in "2005 IEEE International Symposium on Circuits and Systems", pp. 5653–5656 (IEEE, 2005), URL `http://ieeexplore.ieee.org/document/1465920/`.

Tang, X., E. Giacomin, A. Alacchi, B. Chauviere and P.-E. Gaillardon, "OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs", in "2019 29th International Conference on Field Programmable Logic and Applications (FPL)", pp. 367–374 (IEEE, 2019), URL `https://ieeexplore.ieee.org/document/8892171/`.

Tang, X., G. Kim, P.-E. Gaillardon and G. De Micheli, "A Study on the Programming Structures for RRAM-Based FPGA Architectures", IEEE Transactions on Circuits and Systems I: Regular Papers **63**, 4, 503–516, URL `http://ieeexplore.ieee.org/document/7430310/` (2016).

Tech, A., "The iPhone XS and XS Max Review: Unveiling the Silicon Secrets", (2018a).

Tech, I., "Power vr series 3nx neural network accelerator", https://www.imgtec.com/vision-ai/powervr-series3nx/ (2018b).

Tehrani, S., J. Pak, M. Randolph, Y. Sun, S. Haddad, E. Maayan and Y. Betser, "Advancement in Charge-Trap Flash memory technology", in "2013 5th IEEE International Memory Workshop", pp. 9–12 (2013), iSSN: 2159-4864.

Tomasulo, R. M., "An efficient algorithm for exploiting multiple arithmetic units", IBM Journal of Research and Development **11**, 1, 25–33 (1967).

Torricelli, F., L. Milani, A. Richelli, L. Colalongo, M. Pasotti and Z. M. Kovacs-Vajna, "Half-MOS Single-Poly EEPROM Cell in Standard CMOS Process", IEEE Transactions on Electron Devices **60**, 6, 1892–1897, URL http://ieeexplore.ieee.org/document/6515639/ (2013).

Trusov, A., E. Limonova, D. Slugin, D. Nikolaev and V. V. Arlazarov, "Fast implementation of 4-bit convolutional neural networks for mobile devices", in "2020 25th International Conference on Pattern Recognition (ICPR)", pp. 9897–9903 (2021).

Tsuda, S., Y. Kawashima, K. Sonoda, A. Yoshitomi, T. Mihara, S. Narumi, M. Inoue, S. Muranaka, T. Maruyama, T. Yamashita and et al., "First demonstration of FinFET split-gate MONOS for high-speed and highly-reliable embedded flash in 16/14nm-node and beyond", in "2016 IEEE International Electron Devices Meeting (IEDM)", pp. 11.1.1–11.1.4 (IEEE, 2016), URL http://ieeexplore.ieee.org/document/7838393/.

Vasilakis, E., I. Sourdis, V. Papaefstathiou, A. Psathakis and M. G. H. Katevenis, "Modeling energy-performance tradeoffs in arm big.little architectures", in "2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)", pp. 1–8 (2017).

Viraraghavan, J., D. Leu, B. Jayaraman, A. Cestero, R. Kilker, Ming Yin, J. Golz, R. R. Tummuru, R. Raghavan, D. Moy, T. Kempanna, F. Khan, T. Kirihata and S. Iyer, "80Kb 10ns read cycle logic Embedded High-K charge trap Multi-Time-Programmable Memory scalable to 14nm FIN with no added process complexity", in "2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)", pp. 1–2 (IEEE, Honolulu, HI, USA, 2016), URL http://ieeexplore.ieee.org/document/7573462/.

Vrudhula, S., S. Khatri and A. Wagle, "Threshold logic gates using flash transistors", WO Patent WO2021011394A1 (2021).

Vrudhula, S. and A. Wagle, "Fpga with reconfigurable threshold logic gates for improved performance, power, and area", US Patent 11,356,100 (2022).

Wagle, A., E. Azari and S. Vrudhula, "Embedding binary perceptrons in fpga to improve area, power and performance", in "2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)", pp. 1–8 (IEEE, 2019a).

Wagle, A., S. Khatri and S. Vrudhula, "A configurable bnn asic using a network of programmable threshold logic standard cells", in "2020 IEEE 38th International Conference on Computer Design (ICCD)", pp. 433–440 (2020a).

Wagle, A., S. Khatri and S. Vrudhula, "A configurable bnn asic using a network of programmable threshold logic standard cells", in "2020 IEEE 38th International Conference on Computer Design (ICCD)", pp. 433–440 (IEEE, 2020b).

Wagle, A., G. Singh, S. Khatri and S. Vrudhula, "A novel asic design flow using weight-tunable binary neurons as standard cells", (IEEE, 2022a).

Wagle, A., G. Singh, S. Khatri and S. Vrudhula, "A novel asic design flow using weight-tunable binary neurons as standard cells", IEEE Transactions on Circuits and Systems I: Regular Papers **69**, 7, 2968–2981 (2022b).

Wagle, A., G. Singh, J. Yang, S. Khatri and S. Vrudhula, "Threshold logic in a flash", in "2019 IEEE 37th International Conference on Computer Design (ICCD)", pp. 550–558 (IEEE, 2019b).

Wagle, A. and S. Vrudhula, "Heterogeneous fpga architecture using threshold logic gates for improved area, power, and performance", vol. 41, pp. 1855–1867 (IEEE, 2021).

Wagle, A., S. Vrudhula and S. Khatri, "Configurable bnn asic using a network of programmable threshold logic standard cells", US Patent App. 17/504,279 (2022c).

Wagle, A., J. Yang, A. Dengi and S. Vrudhula, "FPGAs with Reconfigurable Threshold Logic Gates for Improved Performance, Power and Area", in "2018 28th International Conference on Field Programmable Logic and Applications (FPL)", pp. 256–2563 (IEEE, 2018a), URL `https://ieeexplore.ieee.org/document/8533505/`.

Wagle, A., J. Yang, A. Dengi and S. Vrudhula, "Fpgas with reconfigurable threshold logic gates for improved performance, power and area", in "2018 28th International Conference on Field Programmable Logic and Applications (FPL)", pp. 256–2563 (IEEE, 2018b).

Wang, Y., H. Shen and D. Duan, "On stabilization of quantized sampled-data neural-network-based control systems", IEEE Transactions on Cybernetics **47**, 10, 3124–3135 (2017).

Wang, Y., Y. Wang, H. Li, Z. Cai, X. Tang and Y. Yang, "Cnn hyperparameter optimization based on cnn visualization and perception hash algorithm", in "2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)", pp. 78–82 (2020).

Wang, Z., M. Agung, R. Egawa, R. Suda and H. Takizawa, "Automatic hyperparameter tuning of machine learning models under time constraints", in "2018 IEEE International Conference on Big Data (Big Data)", pp. 4967–4973 (2018).

Wong, H.-S. P., H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen and M.-J. Tsai, "Metal–oxide rram", Proceedings of the IEEE **100**, 6, 1951–1970 (2012).

Wong, H.-S. P., S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi and K. E. Goodson, "Phase change memory", Proceedings of the IEEE **98**, 12, 2201–2227 (2010).

Wu, J., C. Leng, Y. Wang, Q. Hu and J. Cheng, "Quantized convolutional neural networks for mobile devices", in "2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", pp. 4820–4828 (2016).

Yang, J., J. Davis, N. Kulkarni, J. Seo and S. Vrudhula, "Dynamic and leakage power reduction of ASICs using configurable threshold logic gates", in "2015 IEEE Custom Integrated Circuits Conference (CICC)", pp. 1–4 (IEEE, 2015), URL `http://ieeexplore.ieee.org/document/7338369/`.

Yang, J., N. Kulkarni, S. Yu and S. Vrudhula, "Integration of threshold logic gates with rram devices for energy efficient and robust operation", in "2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)", p. 39–44 (IEEE, 2014a), URL `http://ieeexplore.ieee.org/document/6880500/`.

Yang, J., N. Kulkarni, S. Yu and S. Vrudhula, "Integration of threshold logic gates with RRAM devices for energy efficient and robust operation", in "2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)", pp. 39–44 (IEEE, 2014b), URL `http://ieeexplore.ieee.org/document/6880500/`.

Yang, T.-J., Y.-H. Chen and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning", in "2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", pp. 6071–6079 (2017).

Yue, J., Z. Yuan, X. Feng, Y. He, Z. Zhang, X. Si, R. Liu, M.-F. Chang, X. Li, H. Yang and Y. Liu, "14.3 a 65nm computing-in-memory-based cnn processor with 2.9-to-35.8tops/w system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse", in "2020 IEEE International Solid- State Circuits Conference - (ISSCC)", pp. 234–236 (2020).

Zhang, Q., T. Wang, Y. Tian, F. Yuan and Q. Xu, "Approxann: An approximate computing framework for artificial neural network", in "2015 Design, Automation Test in Europe Conference Exhibition (DATE)", pp. 701–706 (2015).

Zhang, R., P. Gupta, L. Zhong and N. Jha, "Threshold Network Synthesis and Optimization and Its Application to Nanotechnologies", IEEE Transaction On Computer-Aided Design Of Integrated Circuits And Systems **24**, 107–118 (2005).

Zhou, Y. and J. Jiang, "An FPGA-based accelerator implementation for deep convolutional neural networks", in "2015 4th International Conference on Computer Science and Network Technology (ICCSNT)", pp. 829–832 (IEEE, 2015), URL `http://ieeexplore.ieee.org/document/7490869/`.

APPENDIX A

CONVERGENCE PROOF:
PERCEPTRON LEARNING ALGORITHM

Theorem 1. Let there be a threshold voltage assignment $Vt*$ for the FTL circuit, such that the FTL circuit gives the correct response for all the input vectors. The modified perceptron convergence theorem will converge to such an assignment, if possible, within a finite number of steps.

Let the threshold function be defined as follows, for a weight vector $w*$:

1. $w*^T \cdot p > 0$ for every input vector $p \in C1$
2. $w*^T \cdot p < 0$ for every input vector $p \in C1$

The goal is to find the upper bound on the number of iterations needed for the Perceptron Learning Algorithm (PLA) to realize this threshold function on the FTL cell.

Before starting the proof, the following assumptions are made:

1. A solution threshold voltage assignment $Vt*$ exists, for the algorithm to find. This assumption is necessary for having any hope of finding a solution using the algorithm.
2. We assume that the weight vector of the threshold function w is proportional to the threshold voltage vector Vt through some unate, and invertible function f. The Vt vector can be mapped to the actual threshold voltage Vt_{actual}.
3. Let δ be the minimum voltage change that can be done to the threshold voltage of a flash transistor using programming logic.

Proof. The first objective is to find some weight vector w such that the perceptron gives the correct response for all the input vectors. The modified perceptron learning algorithm modifies the Vt vector when there is an incorrect classification/response.

If the k^{th} member of the training set $p(k)$ is incorrectly classified in the k^{th} iteration, then the weights are updated using the following equations:

$$Vt(k+1) = Vt(k) - \delta p(k) \text{ if } w^T \cdot p(k) > 0 \text{ and } p(k) \in C2 \tag{A.1}$$

$$Vt(k+1) = Vt(k) + \delta p(k) \text{ if } w^T \cdot p(k) < 0 \text{ and } p(k) \in C1 \tag{A.2}$$

For correct response, however, there is no change in the Vt vector.

$$Vt(k+1) = Vt(k) \text{ if } w^T \cdot p(k) > 0 \text{ and } p(k) \in C1 \tag{A.3}$$

$$Vt(k+1) = Vt(k) \text{ if } w^T \cdot p(k) < 0 \text{ and } p(k) \in C2 \tag{A.4}$$

Without the loss of generality, the above update equations can be combined if we assume that the minterms are re-assigned sign as follows:

$$m(k) = p(k) \text{ if } p(k) \in C1 \tag{A.5}$$

$$m(k) = -p(k) \text{ if } p(k) \in C2 \tag{A.6}$$

Let the new set associated with the sign-reassigned minterms be X. Then, by combining equations A.1 and A.2, we get:

$$Vt(k+1) = Vt(k) + \delta m(k) \ni m(k) \in X \tag{A.7}$$

We can rewrite the above equation in terms of f and w as:

$$f^{-1}(w(k+1)) = f^{-1}(w(k)) + \delta m(k) \ni m(k) \in X \tag{A.8}$$

$f^{-1}(w(k))$ can be unrolled over all k to produce the following equation:

$$f^{-1}(w(k+1)) = \sum_{n=1}^{k-1} \delta m(n) + \delta m(k) \ni m(k) \in X \tag{A.9}$$

$$f^{-1}(w(k+1)) = \sum_{n=1}^{k} \delta m(n) \ni m(k) \in X \tag{A.10}$$

If we take the dot product on both sides of the equation using Vt^*, we get

$$Vt^{*T} \cdot f^{-1}(w(k+1)) = Vt^{*T} \cdot \sum_{n=1}^{k} \delta m(n) \ni m(k) \in X \tag{A.11}$$

$$Vt^{*T} \cdot f^{-1}(w(k+1)) = \delta Vt^{*T} \cdot \sum_{n=1}^{k} m(n) \ni m(k) \in X \tag{A.12}$$

Let $\alpha = min(Vt^{*T} \cdot m(n)); \forall n \in [1, k])$. Then Equation A.12 can be transformed into the following inequality:

$$Vt^{*T} \cdot f^{-1}(w(k+1)) \geq \delta k \alpha \tag{A.13}$$

Using Cauchy-Schwarz inequality for the dot product $Vt^{*T} \cdot f^{-1}(w(k+1))$, we get

$$||Vt^{*T}||^2 ||f^{-1}(w(k+1))||^2 >= ||Vt^{*T} \cdot f^{-1}(w(k+1))||^2 \tag{A.14}$$

Combining Equations A.13 and A.14, we get:

$$||Vt^{*T}||^2 ||f^{-1}(w(k+1))||^2 >= \delta^2 k^2 \alpha^2 \tag{A.15}$$

$$||f^{-1}(w(k+1))||^2 >= \delta^2 k^2 \alpha^2 / ||Vt^{*T}||^2 \tag{A.16}$$

Equation A.16 represents the upper bound on $||f^{-1}(w(k+1))||^2$, which is the lower bound for the squared Euclidian norm of the threshold voltage vector.

For finding the lower bound on $||f^{-1}(w(k+1))||^2$, we will use the following steps:

$$f^{-1}(w(k+1)) = f^{-1}(w(k)) + \delta m(k) \ni m(k) \in X \tag{A.17}$$

$$||f^{-1}(w(k+1))||^2 = ||f^{-1}(w(k))||^2 + ||\delta m(k)||^2 + 2 f^{-1}(w(k)) \cdot \delta m(k) \ni m(k) \in X \tag{A.18}$$

$$||f^{-1}(w(k+1))||^2 = ||f^{-1}(w(k))||^2 + ||\delta m(k)||^2 + 2(\sum_{i=1}^{k-1} \delta m(i)) \cdot \delta m(k) \ni m(k) \in X \tag{A.19}$$

In the above equation, unrolling the term $f^{-1}(w(k))$ leads to the following equation:

$$||f^{-1}(w(k+1))||^2 = ||\sum_{i=1}^{k-1} \delta m(i)||^2 + ||\delta m(k)||^2 + 2(\sum_{i=1}^{k-1} \delta m(i)) \cdot \delta m(k) \ni m(k) \in X \tag{A.20}$$

Since $||\delta m(k)||^2 = \delta m(k) \cdot \delta m(k)$, the above equation can be simplified as follows:

$$||f^{-1}(w(k+1))||^2 = ||\sum_{i=1}^{k-1} \delta m(i)||^2 + ||\delta m(k)||^2 + 2(\sum_{i=1}^{k-1} \delta m(i)) \cdot \delta m(k) \ni m(k) \in X \tag{A.21}$$

We replace each $m(i)$ with a $\theta \in X$: a vector that has the maximum absolute value out of all $m(i)$. We also increase the upper bound on the summations from k-1 to k.

$$||f^{-1}(w(k+1))||^2 \leq ||\sum_{i=1}^{k} \delta\theta||^2 + ||\delta\theta||^2 + 2(\sum_{i=1}^{k} \delta\theta) \cdot \delta\theta \ni \theta \in X \tag{A.22}$$

We introduce two additional terms β and γ, which are defined as follows:

$$\beta = ||\sum_{i=1}^{k} \delta\theta||^2 \tag{A.23}$$

$$\gamma = ||\delta\theta||^2 + 2(\sum_{i=1}^{k} \delta\theta) \cdot \delta\theta \tag{A.24}$$

For an n-input FTL cell, we have n+1 flash transistors. Using this fact, we simplify

the above equations further, as follows:

$$\beta = ||(k)\delta\theta||^2 \tag{A.25}$$

$$\beta = k^2\delta^2(n+1) \tag{A.26}$$

$$\gamma = \delta^2(n+1) + 2(\sum_{i=1}^{k} \delta\theta) \cdot \delta\theta \tag{A.27}$$

$$\gamma = \delta^2(n+1) + 2k\delta^2(n+1) \tag{A.28}$$

$$\gamma \approx 2k\delta^2(n+1) \tag{A.29}$$

Using the above two equations, Equation A.22 can be represented as follows:

$$||f^{-1}(w(k+1))||^2 \leq \beta + \gamma \tag{A.30}$$

Equation A.30 represents the upper bound on $||f^{-1}(w(k+1))||^2$.

Combining Equation A.16 and A.30, we get the following equation:

$$\delta^2k^2\alpha^2/||Vt^{*T}||^2 \leq \beta + \gamma \tag{A.31}$$

$$\delta^2k^2\alpha^2/||Vt^{*T}||^2 \leq k^2\delta^2(n+1) + 2k\delta^2(n+1) \tag{A.32}$$

Upon simplifying for k, we get:

$$k \leq 2\delta^2(n+1)||Vt^{*T}||^2/\alpha^2\delta^2 \tag{A.33}$$

Minimum value of α is δ. By substituting α, we get:

$$k \leq 2(n+1)||Vt^{*T}||^2/\delta^2 \tag{A.34}$$

Since the value of k has an upper bound defined using constants, it is guaranteed to converge to a solution.

□

APPENDIX B

ALL 5-INPUT THRESHOLD FUNCTIONS

Table B.1: List of All 117 Unique 5-Input Threshold Functions

Index	Weights	Threshold Functions (Sum of Products)
#	[a,b,c,d,e;T]	
0	[1,0,0,0,0;1]	a
1	[1,1,0,0,0;2]	ab
2	[1,1,0,0,0;1]	a | b
3	[1,1,1,0,0;3]	abc
4	[1,1,1,0,0;1]	a | b | c
5	[1,1,1,0,0;2]	ab | bc | ca
6	[2,1,1,0,0;3]	ab | ac
7	[2,1,1,0,0;2]	a | bc
8	[1,1,1,1,0;4]	abcd
9	[1,1,1,1,0;1]	a | b | c | d
10	[2,2,1,1,0;5]	abc | abd
11	[2,2,1,1,0;2]	a | b | cd
12	[1,1,1,1,0;3]	abc | abd | acd | bcd
13	[1,1,1,1,0;2]	ab | ac | ad | bc | bd | cd
14	[2,1,1,1,0;4]	abc | abd | acd
15	[2,1,1,1,0;2]	a | bc | bd | cd
16	[2,2,1,1,0;4]	ab | acd | bcd
17	[2,2,1,1,0;3]	ab | bc | ac | ad | bd
18	[3,2,1,1,0;5]	ab | acd
19	[3,2,1,1,0;3]	a | bc | bd
20	[3,2,2,1,0;5]	ab | ac | bcd

21	[3,2,2,1,0;4]	ab | ac | bc | ad
22	[2,1,1,1,0;3]	ab | ac | ad | bcd
23	[3,1,1,1,0;4]	ab | ac | ad
24	[3,1,1,1,0;3]	a | bcd
25	[1,1,1,1,1;5]	abcde
26	[1,1,1,1,1;1]	a | b | c | d | e
27	[2,2,2,1,1;7]	abcd | abce
28	[2,2,2,1,1;2]	a | b | c | de
29	[1,1,1,1,1;4]	abcd | abce | abde | acde | bcde
30	[1,1,1,1,1;2]	ab | ac | bc | ad | bd | cd | ae | be | ce | de
31	[2,2,1,1,1;6]	abcd | abce | abde
32	[2,2,1,1,1;2]	a | b | cd | ce | de
33	[2,1,1,1,1;5]	abcd | abce | abde | acde
34	[2,1,1,1,1;2]	a | bc | bd | cd | be | ce | de
35	[3,3,2,1,1;8]	abc | abde
36	[3,3,2,1,1;3]	a | b | cd | ce
37	[2,2,2,1,1;6]	abc | abde | acde
38	[2,2,2,1,1;3]	ab | ac | bc | ad | bd | cd | ae | be | ce
39	[3,2,2,1,1;7]	abc | abde | acde
40	[3,2,2,1,1;3]	a | bc | bd | cd | be | ce
41	[3,3,2,2,1;8]	abc | abd | acde | bcde
42	[3,3,2,2,1;4]	ab | ac | bc | ad | bd | cd | ae | be
43	[1,1,1,1,1;3]	abc | abd | acd | bcd | abe | ace | bce | ade | bde | cde
44	[4,3,2,2,1;9]	abc | abd | acde

45	[4,3,2,2,1;4]	a | bc | bd | cd | be
46	[3,2,2,2,1;7]	abc | abd | acd | bcde
47	[3,2,2,2,1;4]	ab | ac | bc | ad | bd | cd | ae
48	[3,3,1,1,1;7]	abc | abd | abe
49	[3,3,1,1,1;3]	a | b | cde
50	[2,2,1,1,1;5]	abc | abd | abe | acde | bcde
51	[2,2,1,1,1;3]	ab | ac | bc | ad | bd | ae | be | cde
52	[3,3,2,2,1;7]	abc | abd | acd | bcd | abe
53	[3,3,2,2,1;5]	ab | ac | bc | ad | bd | cde
54	[3,3,2,2,2;7]	abc | abd | acd | bcd | abe | ace | bce | ade | bde
55	[3,3,2,2,2;6]	ab | acd | bcd | ace | bce | ade | bde | cde
56	[2,2,2,1,1;5]	abc | abd | acd | bcd | abe | ace | bce
57	[2,2,2,1,1;4]	ab | ac | bc | ade | bde | cde
58	[3,2,1,1,1;6]	abc | abd | abe | acde
59	[3,2,1,1,1;3]	a | bc | bd | cde
60	[4,3,2,2,1;8]	abc | abd | acd | abe | bcde
61	[4,3,2,2,1;5]	ab | ac | bc |
62	[4,3,3,2,1;8]	abc | abd | abe | ace | acd | bcd
63	[4,3,3,2,1;6]	ab | ac | a | d | bc | bcd | cde
64	[4,3,3,2,2;8]	abc | abd | abe | acd | ace | ade | bcd | bce
65	[4,3,3,2,2;7]	ab | ac | ade | bcd | bce | bde | cde
66	[3,3,1,1,1;6]	ab | acde | bcde
67	[3,3,1,1,1;4]	ab | bc | ac | ad | bd | ae | be
68	[2,2,1,1,1;4]	ab | acd | bcd | ace | bce | ade | bde

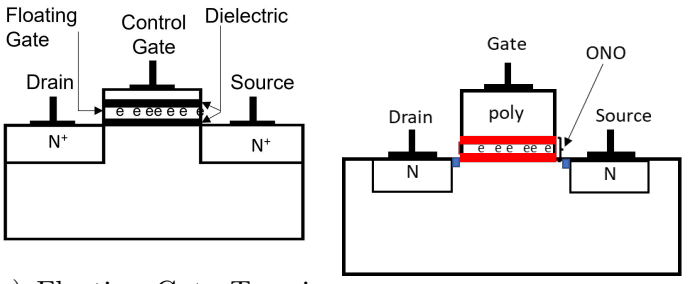
69	[3,3,2,1,1;6]	ab | acd | bcd | ace | bce
70	[3,3,2,1,1;5]	ab | ac | bc | ade | bde
71	[5,3,2,2,1;9]	abc | abd | ace | acd
72	[5,3,2,2,1;5]	a | bc | bd | cde
73	[3,2,2,1,1;6]	abc | abd | acd | abe | ace | bcde
74	[3,2,2,1,1;4]	ab | bc | ca | ad | ae | bde | cde
75	[3,2,2,2,1;6]	abc | add | acd | bcd | abe | ace | ade
76	[3,2,2,2,1;5]	ab | ac | ad | bcd | bce | bde | cde
77	[4,3,1,1,1;7]	ab | acde | bcde
78	[4,3,1,1,1;4]	a | bc | bd | be
79	[5,4,2,2,1;9]	ab | acd | bcde
80	[5,4,2,2,1;6]	ab | ac | ad | ae | bc | bd
81	[5,4,3,2,1;9]	ab | acd | ace | bcd
82	[5,4,3,2,1;7]	ab | ac | ad | bc | bde
83	[5,4,3,2,2;9]	ab | acd | ade | ace | bcd | bce
84	[5,4,3,2,2;8]	ab | ac | ade | bcd | bde | bce
85	[4,3,3,1,1;7]	ab | ac | bcd | bce
86	[4,3,3,1,1;6]	ab | bc | ac | ade
87	[4,2,2,1,1;7]	abc | abd | acd | abe | ace
88	[4,2,2,1,1;4]	a | bc | bde | cde
89	[2,1,1,1,1;4]	abc | abd | acd | abe | ace | ade | bcde
90	[2,1,1,1,1;3]	ab | ac | ad | bcd | ae | bce | bde | cde
91	[4,3,2,1,1;7]	ab | acd | ace | bcde
92	[4,3,2,1,1;5]	ab | ac | bc | ad | ae | bde

93	[4,3,2,2,1;7]	ab | acd | bcd | ace | ade
94	[4,3,2,2,1;6]	ab | ac | ad | bcd | bce | bde
95	[3,2,2,1,1;5]	ab | ac | bcd | bce | ade
96	[3,1,1,1,1;5]	abc | abd | acd | abe | ace | ade
97	[3,1,1,1,1;3]	a | bcd | bce | bde | cde
98	[5,3,2,1,1;3]	ab | acd | ace
99	[5,3,2,1,1;5]	a | bc | bde
100	[3,2,1,1,1;5]	ab | acd | ace | ade | bcde
101	[3,2,1,1,1;4]	ab | ac | ad | bcd | ae | bce | bde
102	[5,3,3,1,1;8]	ab | ac | bcde
103	[5,3,3,1,1;6]	ab | ac | ad | ae | bc
104	[5,3,3,2,1;8]	ab | ac | ade | bcd
105	[5,3,3,2,1;7]	ab | ac | ad | bcd | bce
106	[4,2,1,1,1;6]	ab | acd | ace | ade
107	[4,2,1,1,1;4]	a | bcd | bce | bde
108	[4,2,2,1,1;6]	ab | ac | ade | bcde
109	[4,2,2,1,1;5]	ab | ac | ad | bcd | ae | bce
110	[5,2,2,1,1;7]	ab | ac | ade
111	[5,2,2,1,1;5]	a | bcd | bce
112	[5,2,2,2,1;7]	ab | ac | ad | bcde
113	[5,2,2,2,1;6]	ab | ac | ad | bcd | ae
114	[3,1,1,1,1;4]	ab | ac | ad | ae | bcde
115	[4,1,1,1,1;5]	ab | ac | ad | ae
116	[4,1,1,1,1;4]	a | bcde

APPENDIX C

FLASH TRANSISTORS

A flash transistor is functionally similar to a field effect (FET) transistor, except that it is made to have an additional layer of charge between the control gate and the channel as a means to adjust the threshold voltage (VT) of the transistor. Programming and erasing these devices correspond to increasing and decreasing VT, and this is achieved by electrons tunneling into or out of the charge layer via Fowler-Nordheim (FN) tunneling Fowler and Nordheim (1928). In general, programming is performed by applying a sequence of high voltage pulses (the duration and magnitude of which determines VT) Richter (2014) to the control gate and holding the bulk, source, and drain terminals at $0V$. Erasure is achieved by holding the control gate at $0V$ and allowing the source and drain to float Richter (2014). In flash memory, the value of VT, which is determined by sensing the current, represents the state or stored value, and this can be retained for more than ten years Richter (2014), while the bulk is driven to a high voltage. Several variants of flash transistors with different structures and materials have been investigated over the past two decades to reduce the programming voltage, improve reliability, encode multiple bits, reduce the number of fabrication steps, and improve yield.



(a) Floating Gate Transistor

(b) Charge Trap Transistor

Figure C.1: Cross Section of Flash transistors.

Figure C.1 shows a cross-section of the two common types of flash transistors: (a) the floating gate transistor (FGT), which has an additional buried and un-contacted

floating gate, and (b) the charge trapping transistor (CTT) which has an oxide-nitride-oxide (ONO) layer between the gate and the substrate. Floating gate technology (Figure C.1a) is fully compatible with and used along with CMOS, and because of its dominant role as NVM in flash drives and solid-state drives (SSD), its design and fabrication have been continuously improved over two decades. However, it still has several drawbacks, including the need for additional masks, the requirement of higher voltages for programming and erasure, and most importantly, the difficulty in scaling its dimensions below 40nm due to the poor scaling of the thin oxide.

The following presents a comprehensive survey of the various flash technologies that are available in the industry today. While other surveys on flash transistors focus exclusively on their use in memories, this survey focuses on the use of flash transistors in the implementation of Boolean Logic for high-speed and low-power applications. From the perspective of fabrication, the flash transistors are mature, and their characteristics have been thoroughly studied over the years. They have several important characteristics, such as their non-volatility, speed, long retention, robustness to process variations, compatibility with standard CMOS circuits, and, most importantly, their ability to tune threshold voltage, which can be useful to realize circuits that would otherwise require a large amount of area, power, and delay if implemented using conventional CMOS-based technology.

There are two commonly used mechanisms to program and erase the floating gate transistor, *FN Tunneling* (Fowler-Nordheim Tunneling) Fowler and Nordheim (1928)and *CHEI* (Channel Hot Electron Injection) Simon Tam *et al.* (1984). FN Tunneling is slower but requires less current than the CHEI. A flash transistor can be programmed using both FN Tunneling and CHEI, but for erasing, only FN Tunneling can be used. Details of each technique are provided below:

- *FN Tunneling*:

It is a quantum effect where electrons attaining sufficiently high energy due to an external electrostatic field, break through a small energy barrier. In the case of a flash transistor with thin oxide, the electrons can tunnel through it to the floating gate when an external voltage generates an electric field across the gate oxide. An oxide thickness of about 10nm is required to achieve FN tunneling. The tunneling current due to the FN tunneling mechanism can be described by equation C.1.

$$I_{\mathrm{FN}} = A(E_{ox})^2 exp\frac{-B}{E_{ox}} \tag{C.1}$$

$$E_{ox} = \frac{V_{\mathrm{app}} - V_{\mathrm{fb}}}{t_{ox}} \tag{C.2}$$

where A and B are constants, V_{app} is the applied voltage across the oxide, V_{fb} is the flat band voltage and t_{ox} is the thickness of the oxide.

- *Channel Hot Electron Injection (CHEI)*:

  In the CHEI mechanisms, the electrons gain enough kinetic energy when a lateral electric field is applied at the transistor channel using drain biasing. When the drain terminal is positively biased against the source a current channel is formed from the source to the drain. The electrons accelerate from the source towards the drain and near the drain terminal, they accumulate enough energy to overcome the Si-SiO_2 energy barrier and get injected into the floating gate. The CHEI mechanism has been found to be difficult to represent mathematically due to many unknown physical parameters and characteristics. One model though, named the "Lucky Electron" model Simon Tam *et al.* (1984) has been able to describe the mechanism analytically. This model makes an assumption that the electrons do not lose energy due to collisions as they move from source

to drain but keep on building the kinetic energy. The CHEI current equation of the model is given in equation C.3

$$I_{\text{CHE}} = \alpha_{ox} I_{\text{sub}} exp \frac{-\beta_{ox}}{E_{ox}} \tag{C.3}$$

where α_{ox} and β_{ox} are model fitting parameters, E_{ox} is the electric field into the tunnel dielectric and I_{sub} is the substrate current.

The algorithm used to program a flash transistor to a particular threshold voltage is known as incremental step pulse programming algorithm (ISPP) Hang-Ting Lue *et al.* (2008). It involves sending a sequence of program pulses to the gate of a flash transistor to gradually raise its threshold voltage. The rate at which the threshold voltage changes depends on the amplitude and the duration of the high-voltage pulses. The work in Bayat *et al.* (2016) demonstrates that it is possible to use ISPP for programming the flash transistor within 0.3% precision. On a flash transistor, the algorithm can achieve 1500 levels and store over 10 bits per transistor.

When multiple flash transistors are present in the same system, the presence of cross-talk and short-channel effects on the voltage programmed tends to shift and create a distribution for different flash transistors in the circuit Jung *et al.* (2008). Therefore flash memory cells are programmed for a much small threshold voltage distribution. For a two-level flash-based storage cell, there are only two voltage distributions separated by a sufficient margin to have high reliability. In a multi-level cell (See Figure C.2), there are several voltage distributions, that are much closer to each other, as compared to a two-level cell. Consequently, the reliability of the flash cell degrades when we go from a two-level cell to a multi-level cell.

Reliability in flash memory is determined using two metrics namely; *retentivity* and *endurance*. Retentivity or retention is the measure of the time in years a flash cell

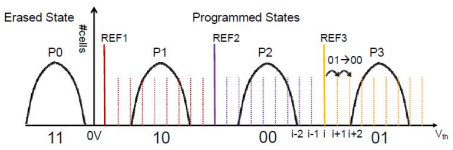


Figure C.2: Vth Distribution in a Multi-Level-Cell

holds a charge (threshold voltage) after being programmed to a particular voltage. Due to various charge leakage mechanisms in the flash transistors, the threshold voltage of the cell changes over the years. The desired retentivity of flash transistors is greater than 10 years.

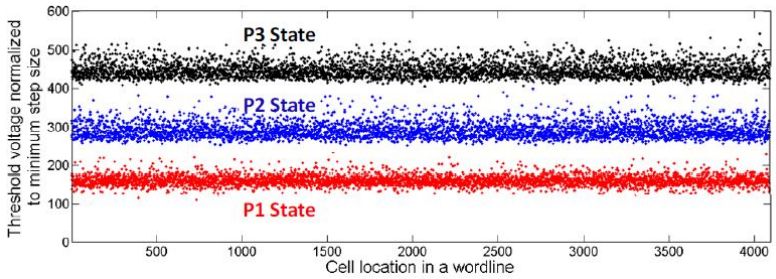


Figure C.3: Vth of Cells Programmed to Different States in a Memory Array

As the flash cell is programmed and erased, it disturbs the flash threshold voltage distribution for the different storage states. The number of program/erase cycles that a flash cell can undergo while still maintaining the desired threshold voltage for different levels is known as *endurance*. An endurance of at least 10^5 is desirable for all flash technologies. With the scaling of the floating gate transistor and use of multi-level cells, the reliability, and endurance of flash memory are expected to decrease Cai *et al.* (2013b) from 10K in 55nm to 3K in 20nm technology. The variation in the threshold voltage of cells in a memory array programmed to different states is shown in figure C.3. Figure C.3 shows the variability in the threshold voltages of the cells programmed to the same state within a memory array. Figure C.3 also shows that the cell is programmed to a particular voltage distribution rather than a fixed voltage

in a memory.

An extensive list of various flash transistors that are currently available in the industry is provided in Table C.2. The flash transistors are sorted based on their technology node, with the oldest technology node first, and the smallest and latest technology node at the bottom. There are several conclusions that can be drawn based on the data provided in the table:

1. The programming of flash transistors requires a much higher voltage than the technology's typical VDD value. As a result, dual voltage rails are needed when integrating flash transistors with conventional CMOS transistors.

2. Older generations of flash technologies used Polysilicon material to create special floating gates, whereas newer generations of flash technologies can make use of the HiK dielectric gates. This means that newer flash transistors do not require extra masks for creating floating gates. The charges are simply stored in the oxide of the gates.

3. Since most of the data was extracted from papers that deal with flash transistors as a memory unit, they are limited to storage of only up to 8 levels. However, for applications where flash transistors are not acting as memory devices, the number of storage levels can be substantially increased.

4. Programming voltages in newer generations of flash transistors are much lower and much closer to VDD as compared to previous generations. Lower programming voltages eliminate the need for transistors that support extremely high voltages.

5. Flash transistors can support a high number of program-erase cycles, much higher than what is needed for the work presented in this dissertation.

6. Almost all flash transistors can retain their programmed threshold voltage for at least a decade or more; a time much larger than the typical life cycle of a chip.

Table C.2: Comparison of Various Properties of Embedded Flash Devices

Type	**Ref**	**Node (nm)**	**Charge storage material**	**Extra Masks**	**Storage Levels**	**Vprog**	**Verase**	**Voperating**	**Endurance**	**Retention**
EERPOM	Cuppens *et al.* (1984)	2500	Polysilicon	1	4	13 V	13 V	5 V	10 K	10 years
HiV EEPROM	Gogl *et al.* (1997)	2000	Polysilicon	0	2	16 V	16 V	10 V	10 K	1000 hrs
C-Flash	Dagan *et al.* (2012)	180	Polysilicon	3	2	5 V	5 V	1.8 V	1 K	100 years
CMOS	Roizin *et al.* (2008)	180	Polysilicon	0	2	4.75 V	4.75 V	1.8 V	1 K	100 years
EEPROM	Cui *et al.* (2009)	180	Polysilicon	0	2	4 V	4.5 V	1.8 V	10 K	NA
MONOS	Ogura *et al.* (2003)	180	Nitride	7	4	4.5 V	4.5 V	1.8 V	100 K	10 years
CMOS	Torricelli *et al.* (2013)	130	Polysilicon	NA	2	12 V	12 V	5 V	10 K	10 years
EEPROM	Chung and Chang-Liao (2015)	130	Polysilicon	NA	4	8 V	14 V	3.3 V	10 K	10 years
Embedded Flash	Raszka *et al.* (2004)	130	Polysilicon	2	2	7 V	7 V	1.2 V	NA	10 years
SONOS	Cho *et al.* (2004)	130	Nitride	NA	4	5 V	5.5 V	3 V	100 K	10 years
Split-gate	noa (2014)	130	Polysilicon	NA	2	8 V	8 V	4 V	NA	10 years
1T MONOS	Mitani *et al.* (2016)	90	Nitride	NA	2	NA	NA	3.3 V	100 M	NA
2T-SONOS	Park *et al.* (2020)	90	Nitride	NA	2	7 V	8 V	3.3 V	1 K	10 years
Charge Trapping	Bartoli *et al.* (2014)	90	Nitride	NA	2	10 V	17 V	NA	100 K	NA
eNVM	Park *et al.* (2014)	90	Polysilicon	0	2	8 V	8 V	3.3 V	500	10 years
eNVM	Song *et al.* (2012)	65	Polysilicon	0	2	8.8 V	8.8 V	2.5 V	10 K	486 hrs
Split-gate	Chu *et al.* (2011)	65	Polysilicon	1	2	11 V	13 V	3.3 V	10 K	10 years
SONOS Pchannel	Shukuri *et al.* (2006)	50	Nitride	NA	2	12 V	12 V	1.8 V	10 K	10 years
Split-gate	Lee *et al.* (2014)	45	Polysilicon	NA	2	NA	NA	1.8 V	1 M	1000 hrs
SG MONOS	Kono *et al.* (2014)	40	Nitride	NA	2	NA	NA	1.25 V	10 M	20 years
SONOS	Agrawal *et al.* (2020)	40	Nitride	3	2	4 V	4 V	0.6 V	100 K	10 years
Split-gate	Luo *et al.* (2016)	40	Polysilicon	0	2	10.5 V	11.5 V	1.1 V	200 K	10 years
Charge Trapping	Tehrani *et al.* (2013)	32	Polysilicon	NA	8	NA	NA	NA	1 M	NA
SG MONOS	Taito *et al.* (2015)	28	Nitride	NA	8	NA	NA	1 V	10 K	NA
Charge Trapping	Viraraghavan *et al.* (2016)	22	HiK dielectric HfO2	0	2	2 V	2 V	1 V	10-1 K	10 years
FINFET eNVM	Ma *et al.* (2019a)	16	HiK dielectric	0	2	2 V	2 V	0.8 V	NA	10 years
FinFET CTT	Khan *et al.* (2019a)	14	HiK dielectric HfO2	NA	2	2 V	2 V	0.8 V	10 K	10 years
FinFET MONOS	Tsuda *et al.* (2016)	14	Nitride	3	2	NA	NA	0.8 V	250 K	10 years

With the matured fabrication technology and extensive development of CMOS-compatible embedded flash technology, various digital and analog logic circuits and architectures have been proposed. There are some inherent advantages to designing logic using flash transistors over CMOS stemming from the fact that the flash transistors can be programmed after the fabrication and the programming can be controlled precisely to achieve fine granularity in the threshold voltages within the circuit. These advantages are as follows:

- Speed binning and removal of setup and hold timing errors post fabrications.
- Aging effects and process variation can be compensated by reprogramming the circuit.

Various digital and analog circuits/architectures designed using flash transistors are discussed further in this chapter.

The use of flash devices for ternary logic cells was first proposed Abusultan and Khatri (2016c) in 2016. This work used proposed the use of NAND flash arrays of floating gate devices as Ternary Logic Clusters (TLCs) to implement digital logic. As compared to the conventional CMOS-based binary logic design of logic circuits this work showed improvement in power (11%), energy (29%), and area (83%) at the clock rate which was 36% as compared to the CMOS design. Ternary Content Addressable Memories (TCAMs) are popular logic structures in many applications employing look-up tables. Traditional TCAM cell implemented using CMOS is area and power inefficient consisting of 17 transistors and an SRAM cell of 6 transistors. The use of flash transistors as proposed by Fedorov *et al.* (2014) initially in 2014 and was improved in 2016 Fedorov *et al.* (2016) reduced the number of transistors in a TCAM cell to two flash transistors and SRAM cell from six transistors to a single flash transistor. Overall, this work demonstrated flash based TCAM is about 7.9x dense than

CMOS-based TCAM block with a 2.5x larger look-up delay and 1.64x lower power consumption. Field programmable gate arrays (FPGAs) are reconfigurable circuits that provide high design flexibility. One of the basic units of FPGA is the look-up table (LUT). LUT traditionally uses SRAMs to store the configuration bits and routing information. Due to the non-volatile nature of the SRAM, SRAM-based FPGAs suffer from high power consumption and longer boot time. In the year 2016, Abusultan and Khatri Abusultan and Khatri (2016b) proposed a flash-based FPGA that embedded the flash transistors directly with the logic and interconnection fabrics. The Static LUT showed improvement in performance (10%), power (static- 29%, dynamic- 12%), and energy (21%) as compared to the SRAM LUT. Whereas, the dynamic flash-based LUT achieved 32% lower delay but with higher energy consumption of 37% as compared to SRAM LUT. Additionally, the flash-based interconnect structures provide 89% delay and 71% overall power consumption improvements when compared to the traditional interconnect structures used in SRAM-based FPGAs.

Flash-based neurons with large fan-in Scott and Khatri (2022) ($>$ 1000 inputs) have also been proposed. In this work, the researchers implemented N flash-transistor-based current sources with binary-weighted currents to construct an N-bit digital-to-analog converter (DAC). These current sources were programmable to deliver currents ranging from the pA-level to the μA-level, providing a wide range of current options for the DAC output. The use of flash transistors as adjustable current sources results in fast response time, small size, and low power consumption. A follow-up work Scott and Khatri (2023) presents the first flash-based artificial neuron design that operates at an extremely low supply voltage of 100 mV. In this work, the weights of the neuron are stored in flash transistors using a novel differential conductance encoding scheme, which accomplishes highly linear voltage output, even with only a 100 mV supply voltage. Note that the flash transistors used in this work have 128 levels, but by

using a novel arrangement of flash transistors, this work shows that weight values of up to 256 (8-bits) can be reliably realized on the neuron when the supply voltage is less than the nominal threshold voltage.

Flash memory arrays are extensively used to perform vector-matrix and matrix-matrix multiplication to implement artificial neural networks (ANNs) and neuromorphic computing. In 2017 the work Guo *et al.* (2017a) designed, fabricated, and tested a 28x28-binary input, 10 output, 3-layer neuromorphic network. This architecture was tested on an MNIST dataset and achieved an accuracy of 94.65%. The inference time of one image was reported to be <1µs. It consumed about 20 nJ energy - both numbers supposedly show a 10^3x improvement over 28nm IBM TrueNorth digital chip for the same dataset and accuracy. Similarly, by Utilizing the CMOS-compatible flash, a Neuromorphic core for handwritten-digit recognition application was developed by Kim *et al.* (2018) in 2018. The core demonstrated an accuracy of 91.8% which was close to software accuracy with the same number of weight levels. Their architecture achieved maximum throughput of is 1.28G pixels/s and a single neuron circuit consumed 15.9µw power on average. A three-level flash gate was used from Song *et al.* (2013) to implement synapse in this architecture. Using the SONOS embedded NVM memory at 40nm technology node, Cypress semiconductors demonstrated analog in memory neuromorphic computing in 2020 Agrawal *et al.* (2020) with synaptic weights stored in the SONOS cell having a capacity of 8 levels/cell. The design achieved an energy efficiency of 100 TOPS/W with 8-bit levels including the energy of analog-to-digital converters (ADC) and digital-to-analog-converters (DAC). They claimed that it was possible to increase the number of levels per cell to 64 or 128 with a loss in energy efficiency. Another in-memory computing proposal using SONOS memory was made by Floadia corporation, Japan in 2020 Nii *et al.* (2020). Mythic AI, a hardware chip design company for AI, has also developed chips

using flash-based analog MAC arrays to perform neural network inference at the edge. They designed the processor at a 40nm technology node and achieved high energy efficiency which is difficult to attain even at advanced nodes of 7nm and 5nm.

The flash-based crossbar arrays have advantages over other SRAM, RRAM, and MRAM-based designs. SRAM-based design can easily be implemented in the standard CMOS process but it suffers from high variability which cannot be tackled after fabrication. MTJs, RRAM, and PCRAM are dense but have other shortcomings such as an MTJ can only store 1 bit per cell, and the difference between its high and low resistance states is only 2x which is insufficient for analog computing. RRAM suffers from high variations and its fabrication is not yet matured. PCRAM and RRAM face structural challenges to implement multi-bit weights. On the other side, flash is a mature technology and can store multi-bits per cell reliably with fine precision.

In analog circuits, circuit designers face a significant obstacle in the form of transistor mismatch, which manifests as an operational amplifier's offset voltage. Traditional methods employed to minimize offset voltage include auto-zeroing, correlated double sampling, and chopper stabilization. However, in the research conducted by Srinivasan et. al Srinivasan *et al.* (2007), floating gate devices were utilized to compensate for the offset voltages. Their proposed technique allowed for continuous operation of the amplifier with long-term offset cancellation and did not necessitate refresh circuitry. A prototype amplifier was manufactured using $0.5m$ technology and reduced the offset voltage to $25V$.

In the doctoral thesis presented by Guillermo J. Serra in 2007 Serrano (2007), he proposed a V_{th} compensated Digital to Analog Converter (DAC) based on floating gate (FG) transistors. The FG transistors were used to compensate for intrinsic V_{th} mismatch of MOS transistors. When compared to other techniques the main advantage of this approach was to achieve higher accuracy with a considerable decrease in

the die area.

APPENDIX D

ENERGY EFFICIENCY OF TULIP

This chapter explores the reason why the TULIP-PEs architecture delivers better energy efficiency as compared to a conventional MAC unit in a QNN accelerator (as detailed in Chapter 5). Before discussing the differences in the energy efficiency between the two processing elements (PEs), it is necessary to discuss the task that will be executed on the PEs. To better illustrate this, we first establish the area, power, and delay values of both these architectures for computing a given unit of workload (operation), shown in Table D.1:

	Area	Power	Delay (Clock Cycles)
MAC Unit	A_{MAC}	P_{MAC}	D_{MAC}
TULIP-PE	A_T	P_T	D_T

Table D.1: Metric Notation for a MAC Unit and a TULIP-PE

For the purpose of comparison, we establish the following correlations between the metrics of the two processing elements as follows.

$$\begin{aligned} A_{MAC} &= \alpha A_T \\ P_{MAC} &= \beta P_T \\ D_{MAC} &= \gamma D_T \end{aligned} \tag{D.1}$$

By incorporating Equation D.1 into Table D.1, we obtain Table D.2.

	Area	Power	Delay (Clock Cycles)
MAC Unit	αA_T	βP_T	γD_T
TULIP-PE	A_T	P_T	D_T

Table D.2: Metric Comparison of a MAC Unit Relative to a TULIP-PE

To compare the performance and energy efficiency of both processing elements quantitatively, we perform an equi-area comparison. Therefore, for every single MAC

unit utilized in a SIMD architecture, there is a set of α TULIP-PEs that can serve as its replacement. Thus, we can compare the performance of a single MAC unit against α TULIP-PEs for a unit workload, as illustrated in Table D.3.

	Area	Power	Delay (Clock Cycles)
MAC Unit	αA_T	βP_T	γD_T
α TULIP-PE	αA_T	αP_T	D_T/α

Table D.3: Metric Comparison of a MAC Unit Relative to α TULIP-PEs

A SIMD architecture is not just created for a single unit of workload, but for multiple units of workload that have a common underlying instruction. The number of workloads that can be initiated at the same time determines the energy cost associated with retrieving the input data. We assume that the energy cost of fetching the input data associated with an operation executing in a single cycle is E_M. As a result, when executing α units of work, a MAC unit consumes αE_M units of energy, whereas the group of TULIP-PEs only requires E_M. The resulting energy efficiency (Workloads per unit of energy) and throughput (Workloads per unit of time) are presented in Table D.4.

For TULIP-PE to outperform an equivalent MAC unit in terms of energy effi-

	Energy Efficiency	Throughput
MAC Unit	$1/(\gamma D_T \beta P_T + E_M)$	$1/\gamma D_T$
TULIP-PE	$1/(D_T P_T + E_M/\alpha)$	α/D_T
Improvement	$(\gamma D_T \beta P_T + E_M)/(D_T P_T + E_M/\alpha)$	$\alpha\gamma$

Table D.4: Energy Efficiency and Throughput Comparison of a MAC Unit Relative to α TULIP-PEs

ciency during an equi-area comparison while maintaining the same throughput, the conditions specified in Equations D.2 and D.3 must be satisfied.

$$\alpha = 1/\gamma \tag{D.2}$$

$$(\gamma D_T \beta P_T + E_M)/(D_T P_T + E_M/\alpha) > 1 \tag{D.3}$$

Substituting Equations D.2 in Equation D.3, we get Equation D.4.

$$(D_T \beta P_T + \alpha E_M)/(\alpha D_T P_T + E_M) > 1 \tag{D.4}$$

Since $E_M >> D_T P_T$ in TSMC 40nm LP Technology, the energy efficiency improvement of TULIP over MAC is approximately α, which is 16X. Furthermore, the value of γ in the same technology is 1/10, implying that a set of α TULIP-PEs collectively exceeds the throughput of a single MAC unit.

It is important to note that the derivation presented in this chapter assumes that the cost of retrieving the input data for a specific workload is solely E_M. However, this cost can vary depending on the operation. For instance, in the case of QNN workloads, the process of refetching not only includes fetching the input pixel and its corresponding weights, but also the neighboring pixels and their respective weights.

APPENDIX E

PUBLISHED PRIOR WORKS

I hereby confirm that I have obtained permission from all co-authors of the previously published paper Wagle *et al.* (2022a); Wagle and Vrudhula (2021) to include it as a chapter in my dissertation. All co-authors have agreed to the extent to which the paper will be used in the dissertation and any changes or additions that will be made. The terms of the agreement have been discussed, including any potential compensation or credit for the co-authors' contributions. Additionally, I have checked the terms of the original publication agreement and obtained any necessary permissions from the publisher to republish the paper in my dissertation. I am committed to giving proper attribution and credit to all co-authors in the dissertation and any subsequent publications that result from this work.