

2012

# Computer Architectures Using Nanotechnology

Yichun Sun  
*Lehigh University*

Follow this and additional works at: <http://preserve.lehigh.edu/etd>

---

## Recommended Citation

Sun, Yichun, "Computer Architectures Using Nanotechnology" (2012). *Theses and Dissertations*. Paper 1155.

This Dissertation is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact [preserve@lehigh.edu](mailto:preserve@lehigh.edu).

# COMPUTER ARCHITECTURES USING NANOTECHNOLOGY

by  
Yichun Sun

A Dissertation  
Presented to the Graduate Committee  
of Lehigh University  
in Candidacy for the Degree of  
Doctor of Philosophy  
in  
Electrical Engineering

**Lehigh University**  
**January 2012**

© Copyright 2011 by Yichun Sun  
All Rights Reserved

This dissertation is accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

---

(Date)

---

Meghanad D. Wagh

---

Miltiadis K. Hatalis

---

Zhiyuan Yan

---

Viswanath Annampedu

# Acknowledgments

I want to take this opportunity to place on record, my deep sense of gratitude to my Ph.D. Advisor, Dr. Meghanad D. Wagh. This work would be impossible without his motivation and untiring guidance and help with my research.

I am extremely thankful to Dr. Zhiyuan Yan, Dr. Miltiadis Hatalis and Dr. Viswanath Annampedu for serving on my dissertation committee and providing valuable feedback on my research. I specially thank Dr. viswanath Annampedu, whose research inspired my work.

Thanks to my parents, for being so cooperative and encouraging with all my work and for giving me a solid educational foundation which inspired me to accomplish such a Ph.D. level work.

This work was supported by NSF in part under grant ECCS-0925890. I am grateful for the support. Also I thank the Electrical Engineering department of Lehigh University for the teaching assistantship.

# Contents

<b>Acknowledgments</b>	<b>iv</b>
<b>Abstract</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Motivation . . . . .	3
1.2 Strategy and Contribution . . . . .	5
1.3 Organization of the Dissertation . . . . .	7
<b>2 Background</b>	<b>8</b>
2.1 Introduction . . . . .	8
2.2 Resonant Tunneling Diodes . . . . .	9
2.3 Quantum-Dot Cellular Automata . . . . .	10
2.4 Threshold Function Fundamentals . . . . .	12
2.4.1 Definition and Examples . . . . .	12
2.4.2 Implementation in Nanotechnology . . . . .	12
2.4.3 Properties of Threshold Functions . . . . .	14
2.5 Differences in Design between CMOS and Nano . . . . .	18
2.5.1 Design Methodology . . . . .	18
2.5.2 Clocking Schemes . . . . .	20
2.6 Design restriction of Nanotechnology . . . . .	21
<b>3 Adder Architectures Using Nanotechnology</b>	<b>23</b>
3.1 Introduction . . . . .	23

3.2	Conventional Adders using Threshold Logic . . . . .	24
3.3	General Scheme for Building New Adders . . . . .	30
3.4	A Low Depth Threshold Adder (LDTA) . . . . .	33
3.5	A Low Complexity Threshold Adder (LCTA) . . . . .	37
3.6	An Enhanced Low Delay Threshold Adder(ELDTA) . . . . .	42
3.7	Discussion and Conclusion of Threshold Adders . . . . .	48
<b>4</b>	<b>Tree Implementation of Combinational Functions</b>	<b>51</b>
4.1	Introduction . . . . .	51
4.1.1	Background . . . . .	52
4.2	k-ary tree decomposition structure . . . . .	58
4.3	Comparison Function Decomposition . . . . .	63
4.4	Conclusion . . . . .	69
<b>5</b>	<b>Systolic Implementation of Threshold Function Decomposition</b>	<b>72</b>
5.1	Introduction . . . . .	72
5.1.1	Systolic Architecture . . . . .	72
5.1.2	Decomposition of Threshold Functions . . . . .	73
5.2	Systolic System for Nanotechnology . . . . .	74
5.2.1	Clocking Scheme . . . . .	74
5.2.2	General Scheme of Decomposing a Threshold Function with Systolic System . . . . .	74
5.2.3	The Whole System . . . . .	78
5.3	Applications and Examples . . . . .	80
5.3.1	Majority Gate . . . . .	80
5.3.2	Pattern Matching Machine . . . . .	81
5.4	Conclusion and Discussion . . . . .	82
<b>6</b>	<b>Digital Circuits for Quantum-Dot Cellular Automata</b>	<b>85</b>
6.1	Introduction . . . . .	85
6.2	Quantum-Dot Cellular Automata . . . . .	86

6.3	Comparison Function . . . . .	87
6.4	A Majority Gate Adder . . . . .	92
6.5	Conclusion . . . . .	93
<b>7</b>	<b>Conclusions</b>	<b>95</b>
7.1	Future Work . . . . .	97
	<b>Bibliography</b>	<b>98</b>
	<b>Vita</b>	<b>104</b>



# List of Tables

2.1	Determining weights and threshold of carry $c_2$ of a 3-bit addition. . .	17
2.2	Determining weights and threshold of function $G_{2:0}$ . . . . .	17
2.3	Determining weights and threshold of function $T_{2:0}$ . . . . .	18
2.4	Complexity comparison of CMOS and Nanotechnology implementa- tions. . . . .	19
3.1	Comparison of Carry Lookahead Adder, CPA, GCLA and our LDTA.	36
3.2	Comparison of the delay of threshold implementations of CPA, GCLA, LDTA and ELDTA using gates with fan-in bound of 5. . . . .	48
3.3	Comparison of the complexity of threshold implementations of CPA, GCLA, LDTA and ELDTA using gates with fan-in bound of 5. . . . .	49
4.1	Comparison of binary tree comparator and quaternary tree compara- tor with fan-in bound of 4. . . . .	70
4.2	Comparison of binary tree comparator and quaternary tree compara- tor with fan-in bound of 8. . . . .	70
5.1	data of gates at cycle $t$ and $t + 1$ . . . . .	78
5.2	Size of majority gate implemented with bounded fan-in. Results of [1, 2] are shown in parenthesis for comparison. . . . .	83
5.3	Depth of majority gate implemented with bounded fan-in. Results of [1, 2] are shown in parenthesis for comparison. . . . .	83

# List of Figures

2.1	$I - V$ characteristics of a RTD. . . . .	9
2.2	Schematic representation of a MOBILE. . . . .	10
2.3	Basic four-dot QCA cell showing the two possible polarizations. . . . .	11
2.4	A QCA wire. . . . .	11
2.5	RTD implementation of a threshold function. . . . .	13
2.6	QCA implementation of a 3-input majority function. . . . .	13
2.7	Four-phase clocking scheme for nanotechnology. . . . .	20
3.1	Carry computation followed by the sum calculation in a Carry Propagation Adder (CPA) . . . . .	25
3.2	Computing all the carries of an 8-bit LDTA with a fan-in bound of 5. . . . .	35
3.3	Carry computation in a 16 bit LCTA with fan-in bound of 5. . . . .	38
3.4	The architecture of a 30-bit ELDTA with fan-in $2M + 1 = 5$ . . . . .	48
4.1	A threshold function of $n$ variables and threshold $T$ partitioned into four fragments and recreated by a tree of recombiners. $R$ equals sum of all positive weights minus $T$ . . . . .	53
4.2	A general structure of a $k$ -ary tree decomposition. . . . .	59
4.3	A ternary tree decomposition of a comparator of size 18 with fan-in bounded by 4. . . . .	65
5.1	A six-phase clocking scheme with the evaluate (E), hold (H), reset (R) and wait (W) states. . . . .	74

5.2	General scheme of decomposing a threshold function by serially combining the fragment outputs. . . . .	75
5.3	Parallel to serial convertor. . . . .	76
5.4	Systolic implementation of the recombiners in Fig.5.2. . . . .	77
5.5	The first recombining stage. . . . .	77
5.6	The first stage of the complete system. . . . .	79
5.7	A 16-bit majority gate with fan-in bound 4. . . . .	81
6.1	(a) The basic QCA cell (b) a polarized QCA cell representing logic 1 and (c) a polarized QCA cell representing logic 0. . . . .	86
6.2	An inverter implementation in QCA. . . . .	86
6.3	A 3-input majority gate implementation in QCA and its symbol. . . . .	87
6.4	QCA realization of function $f$ which is 1 only when its argument is between $X_1$ and $Y_1$ or between $X_2$ and $Y_2$ . Note that $C(\cdot)$ is a comparison function. . . . .	88
6.5	A serial realization of comparison $C(B)$ which outputs a 1 when $s = x_{n-1}x_{n-2} \cdots x_0 \geq b_{n-1}b_{n-2} \cdots b_0$ . . . . .	89
6.6	A 4-bit comparator architecture which produces a 1 when $x_3x_2x_1x_0 \geq b_3b_2b_1b_0$ . (The majority gates in gray need not be implemented.) . . . . .	90
6.7	Layout of a 2 bit comparator in QCA technology. . . . .	91
6.8	The same comparator as in Fig. 6.6 after elimination of wire crossings. . . . .	91
6.9	Calculating carries in an 8-bit LDFA using only 3 input majority gates. . . . .	94

# Abstract

According to the International Technology Roadmap for Semiconductors, the emerging research devices such as Resonant Tunneling Diodes/Transistors (RTD/RTT), Single Electron Transistors (SET) and Quantum Cellular Automata (QCA) are expected to start replacing the CMOS devices in many applications by the end of the next decade. Unfortunately, these new technologies cannot implement the traditional Boolean logic efficiently. On the other hand, they are well suited for threshold logic. Clearly, along with the development of the new devices, one should also explore the new design techniques that are compatible with these devices.

This work focuses on the development of strategies for design and implementation of computer architectures with nanotechnology. Recent studies have demonstrated that the reliability of a nanoelectronic logic gate is dependent upon its fan-in. All the designs presented in this work therefore employ bounded fan-in threshold logic. We develop general schemes to decompose any threshold function into a network of threshold gates with bounded fan-in using a  $k$ -ary tree structure. For some applications, e.g., comparison function, the decomposition scheme leads to a circuit that has a lower complexity and higher speed.

A new strategy to design adders which form the fundamental logic block of any computational unit will be discussed. This strategy allows one to design adders with required speed and complexity using threshold gates with a given fan-in bound. We show that by exploiting the properties of threshold functions, one can get new adder architectures that are much faster and far less complex than the Group Carry Look-ahead Adders (GCLA) presently employed in most modern processors. Our strategies also allow us a three-way trade-off between the hardware complexity, the

reliability and the speed.

Different implementation styles with nanoelectronic threshold gates will also be presented. We show that a systolic implementation of threshold logic can allow one to implement circuits with extremely low hardware complexity but with some loss of speed. We also develop designs using only majority gates which are well suited for Quantum-Dot Cellular Automata (QCA) technology.

# Chapter 1

## Introduction

### 1.1 Motivation

For the last several decades, complementary metal-oxide semiconductor (CMOS) technology has dominated the implementation of very large scale integrated (VLSI) systems. CMOS delivers high speed and consumes low power, the two advantages that are important in the demanding high technology mobile applications of today. The speed of a circuit can be increased and its power dissipation decreased by reducing the physical size of the CMOS device. However, this brings in additional problems such as leakage currents. It is therefore becoming increasingly difficult to shrink the CMOS devices to further improve high speed and low power. According to the International Technology Roadmap for Semiconductors [3], further shrinking of CMOS will become uneconomical by 2020 and new devices based upon *nanotechnology* will start replacing the CMOS devices.

The development of the physics and material technology required for nanotechnology is already well understood and some simple nanoscale devices have been created. Digital systems based on nanotechnology are expected to have smaller area (and therefore higher density), higher speed and lower power dissipation. For example, the devices based upon Resonant Tunneling Diodes/Transistors (RTD/RTT) are expected to reach switching speeds up to 16 THz, those based upon (molecular)

## 1.1. MOTIVATION

Quantum Cellular Automata (QCA) are expected to have packing densities of  $10^{12}$  devices/cm<sup>2</sup> and those based upon Single Electron Technology (SET) are expected to have switching energies as low as  $1 \times 10^{-18}$  J [3]. All three of these technologies and simple devices based on these have been demonstrated in various laboratories around the world.

Though these technologies are quite different from each other, the basic logic block implemented by all the three is a threshold gate [4–7]. In the CMOS technology also, a threshold gate can be efficiently realized within the Field Programmable Gate Array (FPGA). This is because the FPGA implements logic functions using Look Up Tables (LUT). A threshold function can be directly realized by an LUT from its truth table. As a result, both in the current CMOS technology and in the future technologies such as RTD/RTT, SET and QCA, threshold logic plays an important role.

Basic Boolean gates such as the AND/OR/NOT/NAND/NOR are variations of the threshold gate. Thus one can always express the conventional gates in a digital design as threshold gates, thereby providing its nanoelectronic (or FPGA) implementation. However, this design translation fails to harness the true power of the threshold gate which allows one to realize very complex Boolean expressions, often through single gates. (See Examples 1-6 in Subsec. 2.4.3.) Clearly there is a need to develop strategies to design digital systems directly for threshold implementation.

A Boolean function that can be realized with a single threshold gate is known as a threshold function. Finding a threshold implementation of a given Boolean function implies decomposing that function into one or more threshold functions. Unfortunately, the number of threshold functions is highly restricted. For example, while there are  $2^{2^4} = 65,536$  Boolean functions of four variables, only 1,882 ( $\approx 2.87\%$ ) of these are threshold functions. Similarly, out of more than four billion Boolean functions of five variables, less than 0.0023% (exactly 94,572) are threshold [8]. There is also another factor which complicates these designs. It has been

## 1.2. STRATEGY AND CONTRIBUTION

shown that the reliability of nanoelectronic gates is greatly dependent on their fan-in<sup>1</sup> [9–11]. This implies that the intended designs can only use threshold gates with a specified fan-in bound. However, unlike AND/OR gates which can be easily decomposed into similar gates with smaller fan-in, threshold gates with large fan-in cannot be readily decomposed into threshold gates with a smaller fan-in. Thus the design strategies should consider implementation with bounded fan-in threshold gates from the onset.

Naturally the nanoelectronic device technology research has attracted a great deal of attention recently. Unfortunately, the research in developing new architectures using these devices is lagging far behind. This dissertation focuses on the application aspect in the hope that as the technology matures, so do the techniques to use these new devices meaningfully.

## 1.2 Strategy and Contribution

This dissertation is focused on the development of threshold logic architectures for key arithmetic logic units such as the comparators, adders and majority gates. All of the circuits that we design use bounded fan-in gates. We develop different design schemes to optimize properties such as high speed and low complexity. Some of our design strategies even allow tradeoffs between the two. Since our procedures are designed from the onset to be implemented in nanotechnology, they use fewer devices and have smaller size and lower delay as compared to the direct translation of traditional CMOS designs to nanotechnology.

We develop a general scheme of decomposing arbitrary threshold logic into threshold gates with bounded fan-in for higher reliability. An example application of a comparator is given to illustrate the general decomposition scheme. The resultant circuit uses fewer gates and interconnects while having a lower delay.

A large part of this research is devoted to developing faster and low complexity designs of adders which play a central role in arithmetic computations. For this, we

---

<sup>1</sup>Number of inputs to a gate is called its fan-in.



## 1.2. STRATEGY AND CONTRIBUTION

develop an entirely new framework that can be used to develop threshold adders with any desirable features. The most common CMOS high performance adder, the *Group Carry Look-ahead Adder* is based on the carry-generation and carry-propagation logic primitives. However, the carry-propagation primitive uses ExOR gates which are complex to implement in threshold logic. Our new framework replaces this primitive with a new logic primitive that is well suited for threshold implementation. Using this new framework, we obtain designs for a *Low Delay Threshold Adder* (LDTA) and a *Low Complexity Threshold Adder* (LCTA) [12]. We then provide a design procedure for an *Enhanced Low Delay Threshold Adder* (ELDTA) which has an improved architecture to give an even lower delay and lower hardware complexity. A comparison with threshold implementations of traditional adders such as the *Carry Propagation Adder* (CPA) and the *Group Carry Look-ahead Adder* (GCLA) clearly shows the benefits of this new scheme. Note that for the purpose of this comparison, these traditional adders were also modified to optimize their performance when implemented with bounded fan-in threshold gates. It needs to be stressed that the strategy presented in this work is not limited to LDTA, LCTA or ELDTA, and can indeed be used to obtain a variety of adders with different trade-offs between the speed and the hardware complexity.

We also investigated threshold logic with feedback loops. As a result, we have been able to give a novel design strategy to implement systolic architectures with bounded fan-in threshold gates. This design features extremely low hardware complexity with a serial output stream. Since systolic circuits have feedback loops instead of one directional data flow, a new clocking scheme with six phases is introduced. The two example applications, the majority logic implementation and the pattern matching machine are provided and substantiate our claims of very low complexity.

Last we provide digital circuits design for Quantum-Dot Cellular Automata (QCA). Given the fact that QCA can implement very limited types of digital circuits efficiently, including a 3-input majority gate. We develop comparator [13] and adder with majority gates only so that they are applicable to QCA implementation.

## 1.3 Organization of the Dissertation

The Dissertation is organized as follows. In Chapter 2 we introduce the concept of threshold functions and their nanotechnology implementation using Resonant Tunneling Diodes (RTD) and Quantum-Dot Cellular Automata (QCA). We also discuss the differences in digital design between the traditional CMOS technology and new nanotechnology. Then Chapter 4 shows a general scheme of implementing combinational functions using tree architecture. After that, in Chapter 3, we discuss the nanotechnology implementation of adder architecture, which is a key combinational logic block in most arithmetic circuits. We give the implementation for traditional group carry look-ahead adder (GCLA) and present new designs of adder architecture, which shows attractive features of low hardware complexity and high speed. A systolic implementation of some combinational and sequential logics is discussed in Chapter 5. In Chapter 6 we discuss the digital circuits designed specifically for Quantum-Dot Cellular Automata (QCA) implementation. Conclusion and Future work are discussed in Chapter 7.

# Chapter 2

## Background

### 2.1 Introduction

According to the international technology roadmap for semiconductors, the emerging research devices such as Resonant Tunneling Diodes/ Transistors (RTD/RTT), Single Electron Transistors (SET) and Quantum Cellular automata (QCA) are expected to start replacing the CMOS devices in many applications by the end of the next decade. This chapter introduces the digital design using nano-scale devices. Focusing on RTD and QCA, this chapter discusses the fundamentals of these devices. These devices are well suited for *threshold functions*, which are more powerful than the Boolean logic primitives. The fundamentals of threshold logic are presented in this chapter. It also discusses threshold implementations of some Boolean functions which were traditionally built with CMOS gates.

This chapter is organized as follows. Sec. 2.2 and Sec. 2.3 introduce the fundamentals of RTD and QCA respectively. We then provide the definition and properties of the basic logic block implemented by these devices, namely the threshold function in Sec. 2.4. The implementation of threshold functions using RTD and QCA is discussed in this section as well. Examples of building complex boolean functions using nano-scaled devices are presented to illustrate the properties as well as the powerfulness of digital design based on nanotechnology. Finally, Sec. 2.5

## 2.2. RESONANT TUNNELING DIODES

discusses the differences between designs using CMOS and nanotechnology.

## 2.2 Resonant Tunneling Diodes

Integrated circuit designers are constantly thriving to reduce the size of the transistors so as to reduce the complexity and power, and increase the speed and yield. However, there exists a fundamental size limit beyond which one cannot shrink the transistor. If one goes from the  $\mu\text{m}$  scale to nanometer, current sub  $\mu\text{m}$  Si-CMOS transistors cease to operate. In particular, at base region widths comparable to the electron wavelengths, the potential barrier at base leaks severely, allowing electrons to tunnel from the emitter to the collector. At these sizes, conventional transistors lose their switching ability. On the other hand, this same tunneling effect can be exploited in nanoscaled devices such as resonant tunneling diodes (RTDs) and resonant tunneling transistors (RTTs).

RTDs and RTTs use tunneling of electrons at discrete energy levels in a double barrier quantum well structure. The tunneling phenomena create the effect of negative resistance giving the current-voltage ( $I - V$ ) characteristics shown in Fig. 2.1. The voltage and the current at the peak of the characteristic curve in Fig. 2.1 are

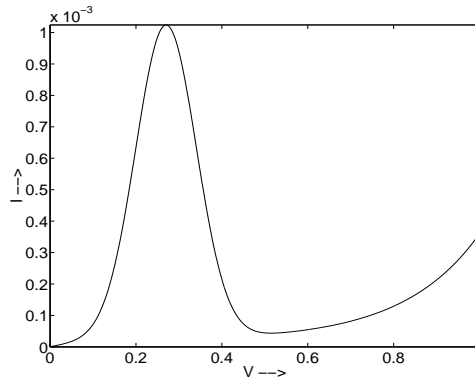


Figure 2.1:  $I - V$  characteristics of a RTD.

referred to as the peak voltage ( $V_P$ ) and peak current ( $I_P$ ) respectively.

RTDs and RTTs are attractive, because they are the most mature of all the

### 2.3. QUANTUM-DOT CELLULAR AUTOMATA

nanoelectronic devices [14, 15] and have already been demonstrated and studied by many researchers. Many large signal models for RTDs and RTTs have also been proposed [16–19].

The basic RTD structure for implementing digital logic is the monostable-bistable transition logic element (MOBILE) [20–22]. MOBILE consists of two serially connected RTDs with an trapezoidal oscillating bias voltage  $V_{SWP}$  as shown in Fig. 2.2.

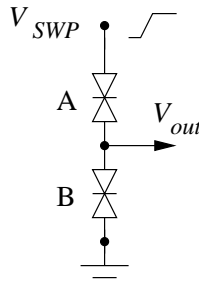


Figure 2.2: Schematic representation of a MOBILE.

The MOBILE *computes* when  $V_{SWP}$  is raised from zero to a voltage slightly above  $2V_P$ . At the end of the compute cycle, the output voltage  $V_{out}$  is high (a logic 1) if the peak current of RTD A is larger than that of RTD B. Voltage  $V_{out}$  is low (logic 0) if the peak current of RTD B is larger. One may add additional RTDs in parallel to RTDs A and B to change the effective peak current in the top or bottom section of the MOBILE. Further, these additional RTDs can be brought in and out of the circuit by HFETs in series which are driven by logical inputs. Thus the output state of a MOBILE is decided by the input variables. Using this and other similar configurations, researchers have already implemented many logic functions [23, 4, 24, 25].

## 2.3 Quantum-Dot Cellular Automata

*Quantum-dot Cellular Automata* (QCA), first described in [26], provides a very different computation platform than traditional CMOS. In QCA, the polarization,

### 2.3. QUANTUM-DOT CELLULAR AUTOMATA

rather than the current, contains the digital information. Further, the digital gates and the interconnections are all made up of the same cells in QCA.

A basic QCA cell consists of four quantum dots in a square array coupled by tunnel barriers. The cell is loaded with two extra electrons which tend to occupy the diagonally opposite positions in the cell because of coulomb repulsion. Therefore in this cell two stable polarization states exist which can be used to represent logic values 1 and 0 as shown in Fig. 2.3.

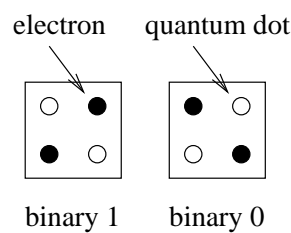


Figure 2.3: Basic four-dot QCA cell showing the two possible polarizations.

Because the electrons are quantum mechanical particles they are able to tunnel between the dots in a cell. The electrons in cells placed adjacent to each other will interact. As a result, coulomb interactions between the electrons force neighboring cells to synchronize their polarization. Therefore an array of QCA cells act as a wire, shown in Fig. 2.4, and is able to transmit information from one end to another [26]; i.e. all the cells in the wire will switch their polarizations to follow that of the input or driver cell.

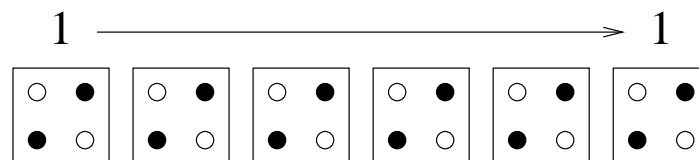


Figure 2.4: A QCA wire.

QCA is ideal for implementing *inverters* and three input *majority gates*. By

## 2.4. THRESHOLD FUNCTION FUNDAMENTALS

using 0 or 1 at one input, a three input majority gate can be converted to a two input AND or OR gate respectively. Thus any Boolean logic can be implemented using QCAs, but with poor efficiency. In the past, QCA architectures of the XOR gate, crossing wires, minority gate, full adder, and a comparison function have also been investigated [27–30, 13].

## 2.4 Threshold Function Fundamentals

### 2.4.1 Definition and Examples

As explained earlier, nanotechnologies such as RTD, RTT and QCA allow direct implementation only of threshold functions. Threshold functions are attractive because they are very powerful in expressing complex Boolean expressions in a much more hardware efficient manner. A Boolean function  $f(x_1, x_2, \dots, x_n)$  is called a *threshold function* if there exist real numbers  $w_1, w_2, \dots, w_n$  and  $T$  such that

$$f(x_1, x_2, \dots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^n w_i x_i \geq T \\ 0 & \text{otherwise.} \end{cases} \quad (2.1)$$

The constants  $w_1, w_2, \dots, w_n$  are called the *weights* of the inputs and  $T$ , the *threshold*. We denote a threshold function as  $\mathbf{TH}(x_1, x_2, \dots, x_n; w_1, w_2, \dots, w_n; T)$ . Since scaling of weights and threshold by the same amount has no effect on the inequality in (2.1), we use integer values for weights and the threshold. A *Generalized Majority function* is a threshold function with all unit weights. A *Majority function* is a generalized majority function with a threshold equal to  $\lfloor n/2 \rfloor + 1$ .

### 2.4.2 Implementation in Nanotechnology

Realizations of threshold functions are called threshold gates. A variety of physical effects can be employed to create a weighted sum of the inputs and then compare it with a preset threshold. These include voltage or current scaling and summation using operational amplifiers [31], charge deposition and summation using a parallel capacitance network [32]. Recently, nanotechnology devices such as QCA, RTD

## 2.4. THRESHOLD FUNCTION FUNDAMENTALS

and RTT, described in the previous section, are dominantly used for implementing threshold functions.

MOBILE, along with HFET as switches, is an ideal solution to computing a weighted sum and comparing it with a threshold due to the I-V characteristics of RTDs. As an example, Fig. 2.5 shows a threshold function  $x + \bar{y}z = \mathbf{TH}(x, y, z; 2, -1, 1; 1)$  implemented as a MOBILE.

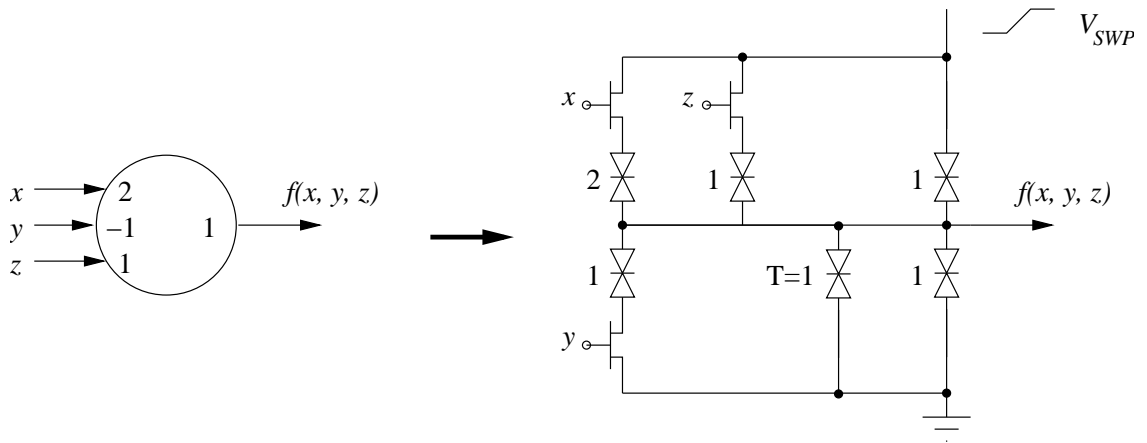


Figure 2.5: RTD implementation of a threshold function.

As mentioned in Sec. 2.3, QCA is able to realize a three input majority function. This is an important threshold function and widely used in this work. Fig. 2.6 shows QCA implementation of a 3-input majority function.

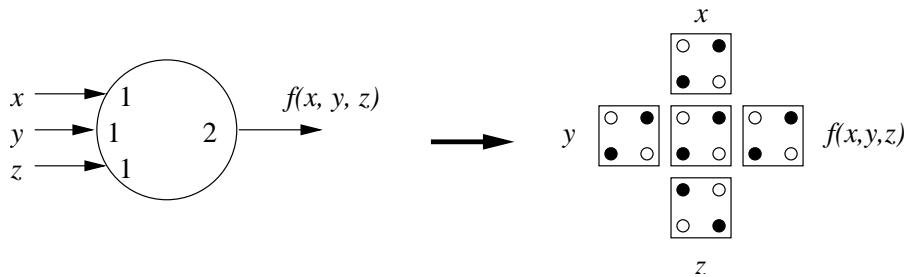


Figure 2.6: QCA implementation of a 3-input majority function.



## 2.4. THRESHOLD FUNCTION FUNDAMENTALS

### 2.4.3 Properties of Threshold Functions

The following characteristics of threshold functions are important in the rest of the dissertation. Properties (P4) - (P6) are not available in literature but are crucial to this work.

**Theorem 1** (Properties of threshold functions.)

$$(P1): x_1 + x_2 = \mathbf{TH}(x_1, x_2; 1, 1; 1).$$

$$(P2): x_1x_2 = \mathbf{TH}(x_1, x_2; 1, 1; 2).$$

$$(P3): x_1x_2 + x_3(x_1 + x_2) = \mathbf{TH}(x_1, x_2, x_3; 1, 1, 1; 2) = \mathbf{MAJ}(x_1, x_2, x_3).$$

(P4): Let  $f = \mathbf{TH}(x_1, x_2, \dots, x_n; w_1, w_2, \dots, w_n; T)$  where each  $w_i \geq 0$ . Then a function  $g$  of  $n + 2$  variables including two new variables  $x_{n+1}$  and  $x_{n+2}$  defined as  $g = x_{n+1}x_{n+2} + (x_{n+1} + x_{n+2})f$  is also a threshold function. In particular,  $g = \mathbf{TH}(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}; w_1, w_2, \dots, w_n, w_{n+1}, w_{n+1}; T')$ , where the threshold  $T'$  of  $g$  is chosen to satisfy  $T' \geq 2T$  and  $T' > \sum_{i=1}^n w_i$  and  $w_{n+1} = T' - T$ .

(P5): Define a series of functions recursively as:  $f_1 = x_2 + x_1x_0$  and  $f_i = x_{2i} + x_{2i-1}f_{i-1}$ ,  $i = 2, 3, 4, \dots$ . Then each function  $f_i$  is a threshold function. In particular,

$$f_i = \mathbf{TH}(x_0, x_1, x_2, \dots, x_{2i}; F_0, F_1, F_2, \dots, F_{2i}; F_{2i}) \text{ where } F_j \text{ is the } j\text{th number in the Fibonacci sequence}^1.$$

(P6): Define a series of functions recursively as:  $f_1 = x_1 + x_0$  and  $f_i = x_{2i-1} + x_{2i-2}f_{i-1}$ ,  $i = 2, 3, 4, \dots$ . Then each function  $f_i$  is a threshold function. In particular,

$$f_i = \mathbf{TH}(x_0, x_1, \dots, x_{2i-1}; F_0, F_1, \dots, F_{2i-1}; F_{2i-1}) \text{ where } F_j \text{ is the } j\text{th number in the Fibonacci series.}$$

---

<sup>1</sup>Fibonacci sequence is defined by the difference equation  $F_j = F_{j-1} + F_{j-2}$  with the initial conditions  $F_0 = F_1 = 1$ .

## 2.4. THRESHOLD FUNCTION FUNDAMENTALS

(P7): The function  $f_i$  of  $2i + 1$  variables in (P5) above can be realized using a chain of  $\lceil i/M \rceil$  threshold gates with fan-in bound of  $2M + 1$ . The total number of inputs to these functions equals  $2i + \lceil i/M \rceil$ .

(P8): A function  $h_i = x_0x_1x_2 \cdots x_i$  of  $i + 1$  variables can be realized using a chain of  $\lceil i/(2M) \rceil$  threshold gates with fan-in bound of  $2M + 1$ . The total number of inputs to these gates equals  $i + \lceil i/2M \rceil$ .

*Proof (Sketch).* Properties (P1)-(P3) can be verified using the function truth tables. Property (P4) can be proved by separately considering cases when the triple  $(x_{n+1}, x_{n+2}, f)$  takes distinct values and verifying that in each case, the given weights and threshold provide the correct value of the function  $g$ .

Proofs of properties (P5) and (P6) are similar. We therefore only focus on (P5) which is proved by mathematical induction. In property (P5),  $f_1$  can be easily verified to be threshold with weights of  $x_0$ ,  $x_1$  and  $x_2$  to be 1, 1 and 2 and the threshold being 2. Assume that  $f_{i-1}$  is a threshold function with the weights and threshold as stated in the theorem. Function  $f_i$  should be 1 if either  $x_{2i} = 1$  or if  $x_{2i-1} = 1$  and  $f_{i-1} = 1$ . In the first of these cases, weight of  $x_{2i}$  is the same as the threshold  $F_{2i}$ . Thus (2.1) implies that  $f_i = 1$ . In the second case, since  $f_{i-1} = 1$ , weighted sum of the variables  $x_0$  through  $x_{2i-2}$  is at least equal to the threshold of  $f_{i-1}$ , namely  $F_{2i-2}$ . Since  $x_{2i-1} = 1$  as well, the weighted sum of all the variables  $x_0$  through  $x_{2i}$  of  $f_i$  is at least equal to  $F_{2i-2} + F_{2i-1} = F_{2i}$ , the threshold of  $f_i$ . Therefore again from (2.1) one gets that  $f_i = 1$ . Similarly, Using the identity that  $F_0 + F_1 + \cdots + F_{2i-2} = F_{2i} - 1$ , one can show that when  $x_{2i} = 0$  and at least one of  $x_{2i-1}$  or  $f_{i-1}$  equal 0, the weighed sum of the variables in  $f_i$  is less than  $F_{2i}$ , the threshold of  $f_i$ , Therefore  $f_i = 0$  as expected from the form of the function.

Property (P7) is proved by construction. One can partition the original function in a series of functions: the first function with variables  $x_0$  through  $x_{2M}$ , the second with the output of the previous function along with variables  $x_{2M+1}$  through  $x_{4M}$  and so on till all variables are exhausted. There are  $\lceil i/M \rceil$  functions in this series. Each of these functions can be implemented as a single threshold gate because of (P5) if the fan-in is odd and because of (P5) and (P6) if the fan-in is even.

## 2.4. THRESHOLD FUNCTION FUNDAMENTALS

Property (P8) is proved similarly. The original function can be partitioned into a series of functions: the first with variables  $x_0$  through  $x_{2M}$ , the second with the output of the first function with  $2M$  more variables and so on until all  $i + 1$  variables are exhausted. There are  $\lceil i/2M \rceil$  such functions, each of which is a  $2M + 1$  input AND gate, which is obviously a threshold function. ■

Application of Theorem 1 is illustrated by the following examples.

**Example 1** *By using (P2) followed by (P4), one gets*

$$a_1b_1 + (a_1 + b_1)a_0b_0 = \mathbf{TH}(a_0, b_0, a_1, b_1; 1, 1, 2, 2; 4).$$

**Example 2** *By using (P2) followed by (P4) twice, one gets*

$$a_2b_2 + (a_2 + b_2)(a_1b_1 + (a_1 + b_1)a_0b_0) = \mathbf{TH}(a_0, b_0, a_1, b_1, a_2, b_2; 1, 1, 2, 2, 4, 4; 8).$$

**Example 3** *By using (P1) followed by (P4) twice, one gets*

$$\begin{aligned} & a_2b_2 + (a_2 + b_2)(a_1b_1 + (a_1 + b_1)(a_0 + b_0)) \\ &= \mathbf{TH}(a_0, b_0, a_1, b_1, a_2, b_2; 1, 1, 2, 2, 4, 4; 7). \end{aligned}$$

**Example 4** *By using (P3) followed by (P4) twice, one gets*

$$\begin{aligned} & a_2b_2 + (a_2 + b_2)(a_1b_1 + (a_1 + b_1)(a_0b_0 + c_{-1}(a_0 + b_0))) \\ &= \mathbf{TH}(c_{-1}, a_0, b_0, a_1, b_1, a_2, b_2; 1, 1, 1, 2, 2, 4, 4; 8). \end{aligned}$$

**Example 5** *From (P5) one gets*

$$\begin{aligned} & g_3 + p_3(g_2 + p_2(g_1 + p_1(g_0 + p_0c_{-1}))) \\ &= \mathbf{TH}(c_{-1}, p_0, g_0, p_1, g_1, p_2, g_2, p_3, g_3; 1, 1, 2, 3, 5, 8, 13, 21, 34; 34). \end{aligned}$$

**Example 6** *Using (P7), the function in Example 5 can be implemented by threshold gates with fan-in bound of 5 as:*

$$\mathbf{TH}(\mathbf{TH}(c_{-1}, p_0, g_0, p_1, g_1; 1, 1, 2, 3, 5; 5), p_2, g_2, p_3, g_3; 1, 1, 2, 3, 5; 5).$$

## 2.4. THRESHOLD FUNCTION FUNDAMENTALS

Table 2.1: Determining weights and threshold of carry  $c_2$  of a 3-bit addition.

function	weights							threshold	property used
	$a_2$	$b_2$	$a_1$	$b_1$	$a_0$	$b_0$	$c_{-1}$		
$c$							1	1	P1
$c_0 = a_0b_0 + (a_0 + b_0)c_{-1}$					1	1	1	2	P6
$c_1 = a_1b_1 + (a_1 + b_1)c_0$			2	2	1	1	1	4	P6
$c_2 = a_2b_2 + (a_2 + b_2)c_1$	4	4	2	2	1	1	1	8	P6

Table 2.2: Determining weights and threshold of function  $G_{2:0}$ .

function	weights						threshold	property used
	$a_2$	$b_2$	$a_1$	$b_1$	$a_0$	$b_0$		
$G_{0:0} = a_0b_0$					1	1	2	P2
$G_{1:0} = a_1b_1 + (a_1 + b_1)G_{0:0}$			2	2	1	1	4	P6
$G_{2:0} = a_2b_2 + (a_2 + b_2)G_{1:0}$	4	4	2	2	1	1	8	P6

**Example 7** Let  $c_2$  denote the output carry of a 3 bit addition  $(a_2a_1a_0) + (b_2b_1b_0) + c_{-1}$ . Then from Table 2.1,

$$\begin{aligned} c_2 &= a_2b_2 + (a_2 + b_2)(a_1b_1 + (a_1 + b_1)(a_0b_0 + (a_0 + b_0)c)) \\ &= \mathbf{TH}(a_2, b_2, a_1, b_1, a_0, b_0, c_{-1}; 4, 4, 2, 2, 1, 1, 1; 8). \end{aligned}$$

**Example 8** Let  $G_{2:0}$  denote the function specifying if a carry is generated within bit positions 0 through 2 in an addition. We show later that  $G_{2:0} = a_2b_2 + (a_2 + b_2)(a_1b_1 + (a_1 + b_1)(a_0b_0))$ . From Table 2.2 one gets

$$G_{2:0} = \mathbf{TH}(a_2, b_2, a_1, b_1, a_0, b_0; 4, 4, 2, 2, 1, 1; 8).$$

**Example 9** Let  $T_{2:0}$  denote the function specifying if a carry is generated within or propagated through bit positions 0 through 2 in an addition. We show later that  $T_{2:0} = a_2b_2 + (a_2 + b_2)(a_1b_1 + (a_1 + b_1)(a_0 + b_0))$ . From Table 2.3 one can see that

$$T_{2:0} = \mathbf{TH}(a_2, b_2, a_1, b_1, a_0, b_0; 4, 4, 2, 2, 1, 1; 7).$$

## 2.5. DIFFERENCES IN DESIGN BETWEEN CMOS AND NANO

Table 2.3: Determining weights and threshold of function  $T_{2:0}$ .

function	weights						threshold	property used
	$a_2$	$b_2$	$a_1$	$b_1$	$a_0$	$b_0$		
$T_{0:0} = a_0 + b_0$					1	1	1	P2
$T_{1:0} = a_1b_1 + (a_1 + b_1)T_{0:0}$			2	2	1	1	3	P6
$T_{2:0} = a_2b_2 + (a_2 + b_2)T_{1:0}$	4	4	2	2	1	1	7	P6

## 2.5 Differences in Design between CMOS and Nano

Although much research has been done on the development and characterization of nanoscale devices, little attention is paid to the impact of these devices on circuit design. The differences between the traditional CMOS design and the novel nanotechnology design lie in two main aspects, the design methodology (Boolean versus threshold) and the clocking (static versus dynamic).

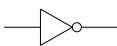
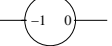

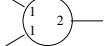

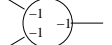



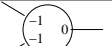
### 2.5.1 Design Methodology

The difference in the two design methodologies is essentially rooted in the fact that CMOS technology is ideal to implement Boolean logic while nanotechnology is better suited for threshold logic. Note that threshold functions allow one to determine the Boolean value of the function by computing an arithmetic sum and compare it with the threshold  $T$ . The use of real numbers and the arithmetic sum rules out CMOS logic to implement such a complex arithmetic operation. However, the nanotechnology MOBILE is ideally suited for such computation. Further, the complexity of an  $n$ -variable threshold function is only  $O(n)$ , and is independent of the algebraic complexity or the number of minterms of the function.

It is easy to interpret the implementation of some basic gates from CMOS technology to nanotechnology. Table 2.4 shows the threshold representation of some basic gates and compares their implementation complexity in the CMOS and Nano technologies. Note that the complexity of CMOS design refers to the number of transistors while RTD design the area of the circuit, governed by the sum of weights

## 2.5. DIFFERENCES IN DESIGN BETWEEN CMOS AND NANO

Table 2.4: Complexity comparison of CMOS and Nanotechnology implementations.

Logic Function	TH representation	Complexity in CMOS technology	Complexity with RTD	Complexity with QCA
		2	1	13
		8	4	5
		4	3	
		8	3	5
		4	2	

and threshold. QCA complexity is measured by the number of QCA cells used in the implementation. However, we focus on the relative costs of different gates which forces different design strategies in these cases.

One can easily draw some conclusions from the table. Firstly, in CMOS technology, NAND is one of the least expensive gate and most design tools convert circuits to NAND gates. However, in RTD technology, OR is just as inexpensive as NAND, and NOR is the least complex gate. One should therefore implement circuits using NORs. In QCA technology, ANDs and ORs are most economical. Secondly, in CMOS technology, any function and its dual have the same complexity while in nanotechnology, they cost different. Thirdly, Boolean functions may have to be expressed differently to be cost efficient in the two technologies.

This shows that due to the technology differences, substantially different design strategies may have to be adopted to minimize costs. The design methods must take into account the differences in the economics and scaling of gates. Further, while CMOS can only use basic gates, nanotechnology can often directly implement complex functions through single threshold gates.

### 2.5.2 Clocking Schemes

Another critical difference between the CMOS and the nanotechnologies is the nature of the *computation*. While CMOS combinational logic is static (does not require a clock), the computation in nanotechnology is possible only with a clock. The delay of a nanotechnology combinational circuit is dependent upon its clocking scheme and the depth. Even in a sequential circuit implementation, different clocking schemes causes quite different realizations in the two technologies.

One of the most unique and novel properties of MOBILE or QCA cell is that its output is valid when the clock is high. Thus it is self-latching. This property can often be exploited to achieve a nanopipeline by constructing a cascaded network of MOBILEs or a network of QCA cells. In such architectures, clocking should be designed such that the computation in any stage start only after the previous stage finishes. One possible solution to this is the four-phase overlapping clocking scheme shown in Fig. 2.7 [33].

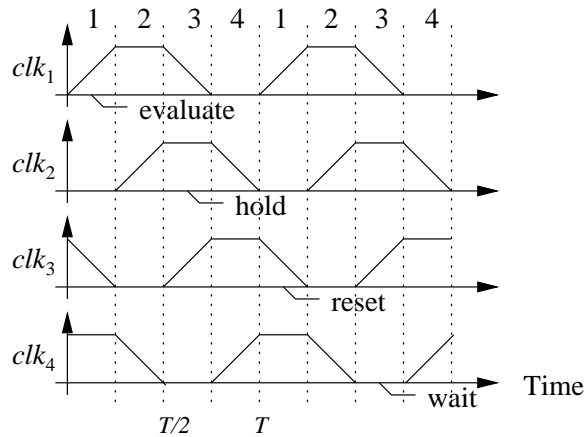


Figure 2.7: Four-phase clocking scheme for nanotechnology.

Each clock has four equal phases of  $(T/4)$  period. Clocks to successive stages are delayed by one phase. During the evaluate phase, the output of a gate is computed. The result is held valid through the hold phase while the subsequent stage is computing. In the reset phase, the gate returns to the monostable mode of operation

## 2.6. DESIGN RESTRICTION OF NANOTECHNOLOGY

and the data is erased. During the wait phase, the inputs of the gate are loaded from the outputs of the gates in the previous pipeline stage [34]. As a result of the four-phase clocking scheme of nanotechnology, the depth of a nanopipeline state is four MOBILEs or four QCA cells as a single clock cycle activates four gates in parallel due to four clock phases.

This four-phase clocking scheme causes the design focus in nanotechnology to be different from CMOS. In CMOS circuits, the depth of a circuit determines its critical path and thereby the delay of the whole system. Therefore, in CMOS technology, depth is one of the most critical aspects that evaluate the performance of the circuits. On the other hand, in nanotechnology, as the whole circuit is automatically pipelined due to the self-latching nanotechnology devices, the depth of the system no longer controls its speed and thus is not an important design parameter. This is not to say that the depth of a nanotechnology circuit is irrelevant to its performance. But if one can trade-off hardware and time complexities, then designing for a lower complexity at the cost of a small increase in depth is often preferred.

## 2.6 Design restriction of Nanotechnology

It has been shown that the reliability of nanoelectronic gates is greatly dependent on their fan-in [9–11]. Similarly, since the LUTs in an FPGA have a finite number of inputs, they can only implement threshold gates with a limited fan-in. These constraints imply that the intended designs can only use threshold gates with a specified fan-in bound. However, unlike AND/OR gates which can be easily decomposed into similar gates with smaller fan-in, threshold gates with large fan-in cannot be readily decomposed into threshold gates with a smaller fan-in. Thus the design strategies for FPGA and nanoelectronic systems should consider implementation with bounded fan-in threshold gates from the onset.

Threshold functions are often classified based on their implementations as follows. The class of Boolean functions that can be implemented by single threshold



## 2.6. DESIGN RESTRICTION OF NANOTECHNOLOGY

gates with unbounded fan-in is known as  $LT_1$ . In general, Boolean functions computable by depth- $d$  networks of unbounded fan-in threshold gates belong to class  $LT_d$ . The size (number of threshold gates) of an  $LT_d$  implementation is restricted to a polynomial function of the number of inputs. Members of the class  $LT_d$  which have weights of polynomial order form a subclass denoted by  $\widehat{LT}_d$ . Clearly functions in  $\widehat{LT}_d$  are more realistic from the implementation standpoint. The class of Boolean functions which can be realized by constant depth, polynomial size networks of threshold functions with  $\pm 1$  weights is denoted by  $TC^0$ . Since any function in  $\widehat{LT}_1$  can be converted to a threshold function with  $\pm 1$  weights by duplicating inputs, it follows that  $\widehat{LT}_d \subseteq TC^0$  for any constant  $d$  [35]. The only class that uses threshold gates with bounded fan-in is the class  $NC^k$ . This class consists of Boolean functions that can be implemented as a polynomial size, depth  $O((\log n)^k)$  network of bounded fan-in AND, OR and NOT gates. It is known that  $TC^0 \subseteq NC^1$  [36, 37]. Thus any function in  $\widehat{LT}_1$  can be implemented using a  $(\log n)$ -depth network of three specific types of bounded fan-in threshold gates; AND, OR and NOT. However, threshold functions are very powerful and a single threshold gate can often replace a complex network of AND, OR and NOT gates. To exploit the power of threshold functions, we provide explicit decomposition of any member of  $LT_d$  into a network of arbitrary threshold gates with bounded fan-in in Chapter 4. Beyond that all our designs compose of threshold gates that are fan-in bounded.

# Chapter 3

## Adder Architectures Using Nanotechnology

### 3.1 Introduction

Arithmetic Logic such as addition and addition related operations like multiplication play an important role in any computational architecture, including the omnipresent electronic computer. With the development of nanotechnology implementation of threshold, people have dedicated efforts for building these arithmetic units using threshold networks. Siu et al. have proved that the addition of two numbers belongs to  $\widehat{LT}_2$  [38]. Maciel have further improved this result by stating that the addition function can be implemented within a majority depth of 0 and a total depth of 2 [37]. A depth-2 and polynomial-sized adder is presented using only non-monotone majority gates in [39]. Unfortunately, all the adders presented above are built with unbounded fan-in threshold gates. A signed digit adder with a restricted fan-in is given by Cotofana [40]. This adder with the assumption of radix-2 signed digit representation uses  $O(n)$  threshold gates with  $O(1)$  weights and fan-in complexity [40]. The adders we propose in this chapter use threshold gates with considerably smaller weights and fan-in.

## 3.2 Conventional Adders using Threshold Logic

This section provides implementations of the *Carry Propagation Adder* (CPA) and the *Group Carry Lookahead adder* (GCLA) using threshold functions. All the adders presented in this work compute the carry bits first and then obtain the sum bits from them. Consider the addition of two  $N$ -bit numbers  $a_{n-1}, a_{n-1}, \dots, a_0$  and  $b_{n-1}, b_{n-2}, \dots, b_0$ . Let  $c_i$  and  $s_i$  denote the carry and sum at the  $i$ th bit position. Clearly,  $s_i = c_{i-1} \oplus a_i \oplus b_i$ . Though implementing an ExOR ( $\oplus$ ) in threshold gates is complex, we can show that the sum bits can be obtained from the carries through only one threshold gate stage as follows.

$$\begin{aligned} s_i &= c_{i-1} \oplus a_i \oplus b_i \\ &= \bar{c}_i(a_i + b_i + c_{i-1}) + a_i b_i c_{i-1}, \end{aligned} \quad (3.1)$$

Expression (3.1) may be proved by considering cases when  $c_{i-1} = 0$  and  $c_{i-1} = 1$ . In the first case,  $c_i = a_i b_i$  and  $s_i = a_i \oplus b_i = \overline{a_i b_i}(a_i + b_i)$  which matches with (3.1). When  $c_{i-1} = 1$ ,  $c_i = a_i + b_i$  and  $s_i = \overline{a_i \oplus b_i} = \overline{a_i} \bar{b}_i + a_i b_i$ , thus validating (3.1) in this case also. Further, because of Theorem 1 (P7),  $s_i$  in (3.1) is a single threshold function. In particular,

$$s_i = \mathbf{TH}(a_i, b_i, c_{i-1}, c_i; 1, 1, 1, -2; 1). \quad (3.2)$$

Note that the sum bit  $s_i$  can also be expressed as a 3-input majority function as follows:

$$s_i = \mathbf{TH}(a_i, b_i, c_{i-1}, \bar{c}_i, \bar{c}_i). \quad (3.3)$$

Since sum bits can be computed from the carry bits by one stage of threshold gates, rest of this work will focus only on computing the carry bits. The delay and complexity estimates given in the rest of the work represent those only to compute the carries.

The simplest of adder, the *Carry Propagation Adder* (CPA) generates a carry  $c_i$  from  $c_{i-1}$  using the expression

$$c_i = a_i b_i + (a_i + b_i) c_{i-1} = \mathbf{MAJ}(a_i, b_i, c_{i-1}).$$

### 3.2. CONVENTIONAL ADDERS USING THRESHOLD LOGIC

Thus the  $N$ -bit CPA uses  $N$  3-input majority gates to compute all the carries. Its delay is  $N$  and its hardware complexity is  $n$  gates and  $3N$  inputs. Fig. 3.1 shows the threshold implementation of a CPA.

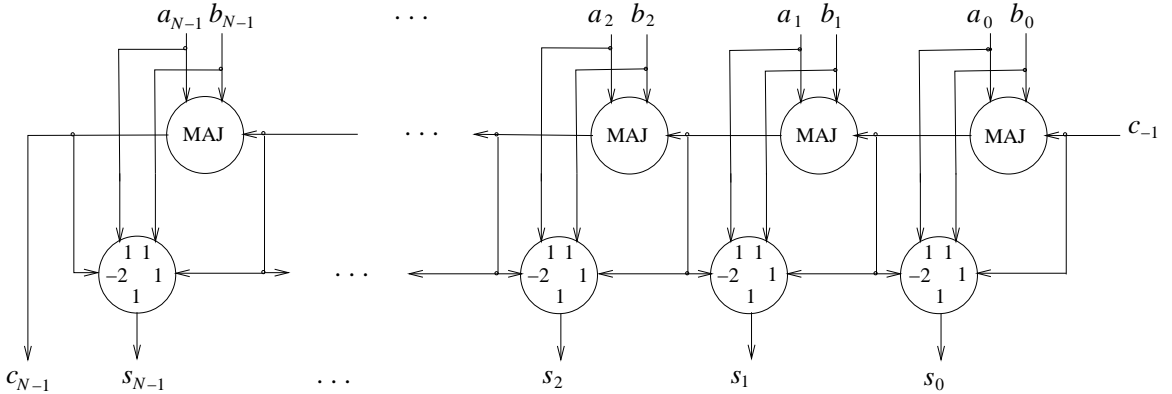


Figure 3.1: Carry computation followed by the sum calculation in a Carry Propagation Adder (CPA)

The  $O(N)$  delay of CPA is not acceptable in many situations. In these applications, one uses a *Group Carry Look-Ahead Adder* (GCLA) [41]. GCLA has an  $O(\log N)$  delay at the cost of modest increase in hardware complexity. GCLA is based upon two logic primitives defined by groups of operand bits: a carry generator and a carry propagator, these primitives are first computed over single bit positions. They can then be combined repeatedly to provide the primitives for increasing group sizes. As we show below, they can be used to compute all the carries in  $O(\log N)$  time complexity.

The carry generator and carry propagator primitives  $g_i^1$  and  $p_i^1$  are obtained from the operand bits at the  $i$ -th position,  $a_i$  and  $b_i$  as

$$g_i^1 = a_i b_i \quad \text{and} \quad p_i^1 = a_i \oplus b_i, \quad 0 \leq i < N.$$

If  $N$  is small, each carry  $c_i$  can be directly obtained as:

$$c_i = g_i^1 + p_i^1 g_{i-1}^1 + p_i^1 p_{i-1}^1 g_{i-2}^1 + \cdots + p_i^1 p_{i-1}^1 \cdots p_0^1 c_{-1}. \quad (3.4)$$

### 3.2. CONVENTIONAL ADDERS USING THRESHOLD LOGIC

Each of these carry expressions can be computed concurrently and in constant time. However, if the operand size  $N$  is larger, then one needs to compute the carry generation/propagation primitives over groups of consecutive bit positions. Generally group size is chosen to be 4. One thus computes the second level of primitives  $g^2$  and  $p^2$  from  $g^1$  and  $p^1$  as:

$$g_i^2 = g_{4i+3}^1 + p_{4i+3}^1 g_{4i+2}^1 + p_{4i+3}^1 p_{4i+2}^1 g_{4i+1}^1 + p_{4i+3}^1 p_{4i+2}^1 p_{4i+1}^1 g_{4i}^1, \quad 0 \leq i < N/4, \quad (3.5)$$

and

$$p_i^2 = p_{4i+3}^1 p_{4i+2}^1 p_{4i+1}^1 p_{4i}^1, \quad 0 \leq i < N/4. \quad (3.6)$$

All the carries  $c_{4i}$ ,  $0 \leq i < N/4$  can then be computed concurrently from these  $g^2$  and  $p^2$  values as:

$$c_{4i} = g_i^2 + p_i^2 g_{i-1}^2 + p_i^2 p_{i-1}^2 g_{i-2}^2 + \cdots + p_i^2 p_{i-1}^2 \cdots p_0^2 c_{-1}. \quad (3.7)$$

The intermediate carries for each  $i$ th group of bits are obtained concurrently from the  $g^1$  and  $p^1$  for bit positions in that group and  $c_{4i-1}$  using an equation similar to (3.4). If  $N > 16$ , the process is repeated by computing  $g^3$  and  $p^3$  for groups of 16 bit positions, computing carries that are 16 bit apart, computing carries that are 4 bit apart and computing the rest of the carries. Thus every time the adder size  $N$  increases 4 times, the time to compute carries increases by a constant amount giving the time complexity of GCLA as  $O(\log N)$ .

In this work, we also consider the group size,  $k$ , of GCLA to be a variable of the adder's performance. Now we give a nanotechnology implementation of GCLA.

**Theorem 2** *GCLA can be implemented using threshold gates with a depth of  $O(\frac{\log N}{M})$  and a number of inputs of  $O(NM)$  where  $N = 2^{2^n}$  is the length of addition and  $M + 1 = 2^m + 1$  is the fan-in of the threshold gates.*

*Proof.*

The depth of GCLA is determined by the largest expression of propagators or generators or carries in each grouping stage.

### 3.2. CONVENTIONAL ADDERS USING THRESHOLD LOGIC

For the first stage, as  $p_i^1$  and  $g_i^1$  are needed for every bit, and as XOR gate is not threshold logic, it takes 2 levels to implement  $p_i^1$ , that is

$$p_i^1 = \mathbf{TH}(\mathbf{TH}(a_i, b_i; 1, -1; 1), a_i, b_i; 2, -1, 1; 1). \quad (3.8)$$

And

$$g_i^1 = \mathbf{TH}(a_i, b_i; 1, 1; 2) \quad (3.9)$$

needs only 1 level. Thus it takes 2 levels for the implementation of  $p_i^1$  and  $g_i^1$ .

For  $p_i^2$  and  $g_i^2$ , as now the group size is  $k$  instead of 4, Eq.3.5 and Eq.3.6 can be modified to

$$p_i^2 = p_{ki}^1 p_{ki+1}^1 \cdots p_{ki+k-1}^1, \quad (3.10)$$

$$g_i^2 = g_{ki+k-1}^1 + p_{ki+k-1}^1 (g_{ki+k-2}^1 + p_{ki+k-2}^1 (g_{ki+k-3}^1 + \cdots + p_{ki+1}^1 g_{ki}^1)). \quad (3.11)$$

As both expressions are serial, they both are threshold logic. As  $g_i^2$  has  $2k - 1$  literals, which is than  $k$  of  $p_i^2$ , it determines that the levels need for this grouping stage is  $\lceil \frac{2k-2}{M} \rceil$ .

Similarly,  $p_i^3$  and  $g_i^3$  can be obtained from  $p_i^2$  and  $g_i^2$  with the same hardware. As there are totally  $\log_k N - 1$  such grouping stages till the last stage only has 1 group on  $k$  members, number of levels needed to implement all the propagators and generators is  $2 + \lceil \frac{2k-2}{M} \rceil (\log_k N - 1)$ .

For carries, we can first get  $c_{\frac{N}{k}i-1}$ , where  $i = 1, 2, \dots, k$ . From Eq.3.7, we can see that the carry with most literals is  $c_{\frac{N}{k}-1}$ , which has  $2k + 1$  literals. Thus we can get this stage takes  $\lceil \frac{2k}{M} \rceil$  levels.

With the above carries, then we can calculate the intermediate carries. Still from Eq.3.7, the most complex expression now has  $2k - 1$  literals, indicating  $\lceil \frac{2k-2}{M} \rceil$  levels, as there are only  $k - 1$  to be calculated in each group. And totally there are  $\log_k N - 1$  stages for carries calculation, thus the number of levels for these carries is  $\lceil \frac{2k-2}{M} \rceil (\log_k N - 1)$ .

With all the above sum up with 1 level for sum bits, the total depth of GCLA is  $2 \lceil \frac{2k-2}{M} \rceil (\log_k N - 1) + \lceil \frac{2k}{M} \rceil + 3$ , which is of  $O(\frac{\log_k N}{M})$ .

Then we discuss the number of inputs in GCLA.

### 3.2. CONVENTIONAL ADDERS USING THRESHOLD LOGIC

Recall Eq.3.8 and 3.9 for the implementation of propagation and generation in the first stage, it is clear that it takes 5 threshold gates to build each propagator and 2 to build each generator. As they are required for every bit position, it totally takes  $7N$  inputs.

For  $p_i^2$  and  $g_i^2$ , from Eq.3.10, we know that  $p_i^2$  each has  $k$  literals, meaning it requires  $\lceil \frac{k-1}{M} \rceil$  threshold gates to implement it. That results in  $\lceil \frac{k-1}{M} \rceil - 1$  inputs in addition to the original  $k$  inputs. So each  $p_i^2$  takes  $k + \lceil \frac{k-1}{M} \rceil - 1$  inputs.

Similarly, from Eq.3.11, each  $g_i^2$  has  $2k - 1$  literals, resulting in a number of inputs of  $2k + \lceil \frac{2k-2}{M} \rceil - 2$ .

And for  $p_i^2$  and  $g_i^2$ , there are  $\frac{N}{k}$  of them, as each  $p_i^2$  and  $g_i^2$  carries the property of  $k$  bits from the previous stage.

Similarly, each  $p_i^3$  and  $g_i^3$  can be obtained from  $p_i^2$  and  $g_i^2$  with the same complexity. And there are  $\frac{N}{k^2}$  pairs of propagator and generators in this stage. As there are totally  $\log_k N - 1$  such grouping stages till the last stage only has 1 group of  $k$  members, number of inputs in the hardware implementing all the propagators and generators is

$$\begin{aligned} & (3k - 3 + \lceil \frac{k-1}{M} \rceil + \lceil \frac{2k-2}{M} \rceil) \left( \frac{N}{k} + \frac{N}{k^2} + \dots + k^2 + k \right) \\ & = \frac{N-k}{k-1} (3k - 3 + \lceil \frac{k-1}{M} \rceil + \lceil \frac{2k-2}{M} \rceil) \end{aligned} \quad (3.12)$$

Then we calculate the number of inputs in the circuit for calculating the carries. Here we discuss this issue in two cases.

- Case1: If  $2k < M$ .

Still, we start with  $c_{\frac{N}{k}i-1}$ , where  $i = 1, 2, \dots, k$ . In this case, from Eq.3.7, the one has the most literals, namely  $c_{\frac{N}{k}-1}$  can be implemented with a single gate. As  $c_{\frac{N}{k}-1}$  has  $2k + 1$  variables and that is within the bound of the fan-in. Then the total number of inputs is the sum of literals in the expression of each  $c_{\frac{N}{k}i-1}$ . it is simple to calculate the number of inputs as  $3 + 5 + 7 + \dots + (2k + 1) = k(k + 2)$ .

### 3.2. CONVENTIONAL ADDERS USING THRESHOLD LOGIC

Then we can calculate the intermediate carries with gaps of  $\frac{N}{k}$ . Still from Eq.3.7, as the most complex expression now has  $2k - 1$  literals, which is still within the bound of the fan-in, all the carries at this stage takes only one gate each. Then similar to the previous stage, it takes  $3 + 5 + 7 + \dots + (2k - 1) = (k+1)(k-1)$  inputs for each group of  $k-1$  members and there are  $k$  such groups in the stage. So the total number of inputs in this stage is  $k(k+1)(k-1)$ .

Similarly, the next stage calculates all the carries with gaps of  $\frac{N}{k^2}$ . And it takes the same number of inputs for each group of  $k-1$  members but the number of groups is  $k^2$  so it takes  $k^2(k+1)(k-1)$  inputs for this stage. In this way, as there are  $\log_k N - 1$  stages for carries calculation, the total number of inputs for these carries can be obtained as  $(k+1)(k-1)(k+k^2+\dots+\frac{N}{k}) = (N-k)(k+1)$ .

Summing up the number of inputs discuss above from generators and propagators to carries, the total number of inputs in the case that  $2k < M$  is  $11N - 2k + Nk + \frac{N-k}{k-1}(\lceil \frac{k-1}{M} \rceil + \lceil \frac{2k-2}{M} \rceil)$ .

- Case1: If  $2k \geq M$ .

In this case, we assume that both  $k$  and  $M$  are powers of 2.

In the first stage of carry calculation, as now the carries with more than  $M + 1$  literals cannot be implemented with one single gate, we use the result of carries with small literals to calculate the more complex ones. For example, it we have a 64-bit adder with  $k = 4$  and  $M = 4$ , then in this stage  $c_{15}$ ,  $c_{31}$ ,  $c_{47}$ ,  $c_{63}$  are to be calculated. From Eq.3.7, we can write these carries as

$$c_{15} = g_0^3 + p_0^3 c_{-1}, \quad (3.13)$$

$$c_{31} = g_1^3 + p_1^3 (g_0^3 + p_0^3 c_{-1}), \quad (3.14)$$

$$c_{47} = g_2^3 + p_2^3 (g_1^3 + p_1^3 (g_0^3 + p_0^3 c_{-1})), \quad (3.15)$$

$$c_{63} = g_3^3 + p_3^3 (g_2^3 + p_2^3 (g_1^3 + p_1^3 (g_0^3 + p_0^3 c_{-1}))). \quad (3.16)$$

As now  $c_{47}$  and  $c_{63}$  has 7 and 9 literals, respectively, that exceed the bound of fan-in, 5. Then we rewrite  $c_{47}$  and  $c_{63}$  as

$$c_{47} = g_2^3 + p_2^3 c_{31}, \quad (3.17)$$



### 3.3. GENERAL SCHEME FOR BUILDING NEW ADDERS

$$c_{63} = g_3^3 + p_3^3(g_2^3 + p_2^3 c_{31}). \quad (3.18)$$

Now they take result of  $c_{31}$  as input so that it reduces the number of inputs from 7 to 3 and 9 to 5, respectively. Note that the depth of this stage does not change with this modification. With this, the number of inputs for  $c_{\frac{64}{4}i-1}$  can be got as  $2(3 + 5)$ .

Now we generalize it from the example. The fan-in limits that the number of carries implemented with one gate is  $\frac{M}{2}$  and these carries have a total number of  $3+5+7+\dots+(M+1) = \frac{M(M+4)}{4}$ . Then there are still that many carries take  $c_{\frac{N}{k}\frac{M}{2}i-1}$  as an input and they have the same number of inputs of  $\frac{M(M+4)}{4}$ . As there are totally  $\frac{2k}{M}$  such units, this stage of carries have altogether  $\frac{2k}{M} \frac{M(M+4)}{4}$  inputs.

Similarly for the intermediate carries except that there are  $k - 1$  member in each group of each stage so there are  $M + 1$  less inputs in each unit described above. And it is clear that the number of inputs for these intermediate carries sums up to  $\frac{N-k}{k-1} (\frac{k(M+4)}{2} - M - 1)$ .

And the number of inputs sums up to  $10N - 3k + \frac{k(M+4)}{2} + \frac{N-k}{k-1} (\lceil \frac{k-1}{M} \rceil + \lceil \frac{2k-2}{M} \rceil) + \frac{k(M+4)}{2} - M - 1$ .

The discussion above gives a complexity of  $O(NM)$  of number of inputs. ■

### 3.3 General Scheme for Building New Adders

As was seen in Section 3.2,  $p_i$  is important to compute  $c_i$ . However  $p_i$  computation requires the use of ExOr gates which are not threshold functions. We show in this section that  $c_i$  can be computed by a majority gate if one uses an alternate property of input bits. We call the adder based on this property as the *Low Depth Threshold Adder* (LDTA).

Let  $g_i$  and  $p_i$  denote the carry generation and propagation properties of the  $i$ -th bit position as in the GCLA. Recall that  $g_i = a_i b_i$  and  $p_i = a_i \oplus b_i$ , where  $a_i$  and  $b_i$  are the  $i$ -th bits of the two operands.

### 3.3. GENERAL SCHEME FOR BUILDING NEW ADDERS

Define  $t_i = g_i + p_i = a_i + b_i$ . One can show that the carry  $c_i$  of the adder can be expressed as

$$c_i = g_i + p_i c_{i-1} = g_i + t_i c_{i-1}. \quad (3.19)$$

However, since  $g_i = g_i t_i$  and  $t_i = g_i + t_i$ , one can also express (3.19) using Theorem 1(P2) as

$$c_i = g_i t_i + g_i c_{i-1} + t_i c_{i-1} = \mathbf{TH}(g_i, t_i, c_{i-1}; 1, 1, 1; 2) = \mathbf{MAJ}(g_i, t_i, c_{i-1}). \quad (3.20)$$

Thus carry at position  $i$  can be obtained from  $c_{i-1}$  using a three input majority function. In order to relate carry  $c_i$  to some other previous  $c_j$ , we generalize  $g_i$  and  $t_i$  to multiple bits.

Let  $G_{i:j}$ ,  $i \geq j$  denote the proposition that carry  $c_i$  is generated in the bit positions  $j$  through  $i$ . Similarly, let  $P_{i:j}$  denote the proposition that this group of bits propagate a carry. As before, define  $T_{i:j} = G_{i:j} + P_{i:j}$ . Clearly,  $G_{i:j} = G_{i:j} T_{i:j}$  and  $T_{i:j} = G_{i:j} + T_{i:j}$ . Thus, similar to (3.19) and (3.20) we have

$$\begin{aligned} c_i &= G_{i:j} + P_{i:j} c_j = G_{i:j} + T_{i:j} c_{j-1}. \\ &= \mathbf{MAJ}(G_{i:j}, T_{i:j}, c_{j-1}) \end{aligned} \quad (3.21)$$

We now show that  $G_{i:j}$  and  $T_{i:j}$  which characterize bit positions  $j$  through  $i$  can be expressed in terms of characterizations of smaller groups of bits. In particular, for any  $j < k \leq i$ , from the definition of  $G_{i:j}$ ,

$$\begin{aligned} G_{i:j} &= G_{i:k} + G_{k-1:j} P_{i:k} \\ &= G_{i:k} + G_{k-1:j} (P_{i:k} + G_{i:k}) \\ &= G_{i:k} + T_{i:k} G_{k-1:j}. \end{aligned} \quad (3.22)$$

Further, because of the relationship between  $G_{i:k}$  and  $T_{i:k}$ , this can be expressed as:

$$G_{i:j} = \mathbf{MAJ}(G_{i:k}, T_{i:k}, G_{k-1:j}). \quad (3.23)$$

Similarly,

$$T_{i:j} = G_{i:j} + P_{i:j}$$

### 3.3. GENERAL SCHEME FOR BUILDING NEW ADDERS

$$\begin{aligned}
&= G_{i:k} + G_{k-1:j}P_{i:k} + P_{i:k}P_{k-1:j} \\
&= G_{i:k} + (G_{i:k} + P_{i:k})(G_{k-1:j} + P_{k-1:j}) \\
&= G_{i:k} + T_{i:k}T_{k-1:j}.
\end{aligned} \tag{3.24}$$

Again, using the relationship between  $G_{i:k}$  and  $T_{i:k}$  one can express this as:

$$T_{i:j} = \mathbf{MAJ}(G_{i:k}, T_{i:k}, T_{k-1:j}). \tag{3.25}$$

Equations (3.23) and (3.25) show that the  $G$  and  $T$  properties of a group of bits can be obtained from the properties of its smaller partitions. Further, the computation employs only 3-input majority gates.

Quantities  $G_{i:j}$  and  $T_{i:j}$  for smaller sets of bits can be determined directly from the input bits as follows. Carry is generated in bit positions  $j$  through  $i$  only if it is generated in one of these bits and is propagated through the higher bits. Thus,

$$G_{i:j} = g_i + p_i(g_{i-1} + p_{i-1}(g_{i-2} + \cdots p_{j+1}(g_j)) \cdots). \tag{3.26}$$

Using the fact that  $g_k + p_kX = g_k + t_kX$  for any  $k$  and  $X$ , one can also write (3.26) as:

$$\begin{aligned}
G_{i:j} &= g_i + t_i(g_{i-1} + t_{i-1}(\cdots t_{j+1}(g_j) \cdots)) \\
&= a_i b_i + (a_i + b_i)(a_{i-1} b_{i-1} + (a_{i-1} + b_{i-1})(\cdots (a_{j+1} + b_{j+1})(a_j b_j) \cdots)) \\
&= \mathbf{TH}(a_j, b_j, a_{j+1}, b_{j+1}, \dots, a_i, b_i; 1, 1, 2, 2, \dots, 2^{i-j}, 2^{i-j}, 2^{i-j+1}).
\end{aligned} \tag{3.27}$$

The last step in (3.28) is obtained as in the examples following Theorem 1.

The expression for  $T_{i:j}$  can be obtained similarly using (3.26) as follows.

$$\begin{aligned}
T_{i:j} &= G_{i:j} + P_{i:j} = G_{i:j} + p_i p_{i-1} \cdots p_j \\
&= g_i + p_i(g_{i-1} + p_{i-1}(g_{i-2} + \cdots p_{j+1}(g_j + p_j) \cdots)).
\end{aligned}$$

This expression can be simplified as before to:

### 3.4. A LOW DEPTH THRESHOLD ADDER (LDTA)

$$\begin{aligned} T_{i:j} &= g_i + t_i(g_{i-1} + t_{i-1}(\cdots t_{j+1}(g_j + t_j)) \cdots) \\ &= a_i b_i + (a_i + b_i)(a_{i-1} b_{i-1} + (a_{i-1} + b_{i-1})) \end{aligned} \quad (3.29)$$

$$\begin{aligned} (\cdots & (a_{j+1} b_{j+1} + (a_{j+1} + b_{j+1})(a_j + b_j)) \cdots) \\ &= \mathbf{TH}(a_j, b_j, a_{j+1}, b_{j+1}, \dots, a_i, b_i; 1, 1, 2, 2, \dots, 2^{i-j}, 2^{i-j}; 2^{i-j+1} - 1) \end{aligned} \quad (3.30)$$

Note that as before, the last step of (3.31) is similar to the examples following Theorem 1.

Similar to the computations (3.28) and (3.31) of  $G_{i:j}$  and  $T_{i:j}$  for small input partitions, one can also compute the initial few carries directly from the input bits using threshold functions. Recall from (3.21) that the carry  $c_i$  can be obtained from  $c_{-1}$  as

$$\begin{aligned} c_i &= G_{i:0} + c_{-1} T_{i:0} \\ &= a_i b_i + (a_i + b_i)(a_{i-1} b_{i-1} + (a_{i-1} + b_{i-1})(\cdots a_1 b_1 + (a_1 + b_1) \end{aligned} \quad (3.32)$$

$$(a_0 b_0 + c_{-1}(a_0 + b_0)) \cdots)) \quad (3.33)$$

$$= \mathbf{TH}(c_{-1}, a_0, b_0, a_1, b_1, \dots, a_i, b_i; 1, 1, 1, 2, 2, \dots, 2^i, 2^i; 2^{i+1}). \quad (3.34)$$

The derivation of (3.34) is based on (3.28) and (3.31) and uses Theorem 1(P6) to convert the computation into a threshold function.

One can also obtain  $c_i$  from  $c_j$ ,  $j < i$ , in a similar fashion.

$$c_i = G_{i:j} + c_{j-1} T_{i:j} \quad (3.35)$$

$$= \mathbf{MAJ}(G_{i:j}, T_{i:j}, c_{j-1}) \quad (3.36)$$

$$= \mathbf{TH}(c_{j-1}, a_j, b_j, a_{j+2}, b_{j+2}, \dots, a_i, b_i; 1, 1, 1, 2, 2, \dots, 2^i, 2^i; 2^{i+1}). \quad (3.37)$$

For convenience, we will often refer to  $G_{i:j}$  and  $T_{i:j}$  as the  $G$  and  $T$  functions over the (bit index) range  $[i : j]$ .

## 3.4 A Low Depth Threshold Adder (LDTA)

Basically, one can use the general scheme described above to generate numerous types of adders by playing around the properties of  $G_{i:j}$  and  $T_{i:j}$ . We present here

### 3.4. A LOW DEPTH THRESHOLD ADDER (LDTA)

one strategy to design the  $N$ -bit *Small Depth Adder* using threshold functions with fan-in bound of  $M+1$  can be described as follows. For convenience, assume  $M = 2^m$  and  $N = 2^n$ .

#### Design of the Low Delay Threshold Adder (LDTA).

1. Obtain  $c_i$ ,  $0 \leq i < M/2$  using (3.34).
2. Obtain  $G$  and  $T$  over  $[jM/2 + k : jM/2]$ ,  $0 \leq k < M/2$ ,  $1 \leq j < 2N/M$  using (3.28) and (3.31).
3. For each  $i$ ,  $0 \leq i < n - m$ , for  $1 \leq j < N/(2^i M)$  and  $2^i M/2 \leq k < 2^{i+1} M/2$ , obtain  $G$  and  $T$  over  $[j2^{i+m} + k : j2^{i+m}]$  from  $G$  and  $T$  over ranges  $[j2^{i+m} + k : j2^{i+m} + 2^{i+m-1}]$  and  $[j2^{i+m} + 2^{i+m-1} - 1 : j2^{i+m}]$ ,  $2^{i+m-1} \leq k < 2^{i+m}$ ,  $1 \leq j < 2^{n-m-i}$  as in (3.23) and (3.25).
4. Obtain carries  $c_{(M/2)2^i+k}$ ,  $0 \leq k < (M/2)2^i$  using carry  $c_{(M/2)2^{i-1}}$  and the appropriate  $G$  and  $T$  using  $c_i = \mathbf{MAJ}(G_{i;j}, T_{i;j}, c_{j-1})$ ,  $0 \leq i \leq n - m$ .
5. Obtain the sum bits from the carries using  $s_i = \mathbf{TH}(a_i, b_i, c_{i-1}, c_i; 1, 1, 1, -2; 1)$ .

Fig. 3.2 shows an 8-bit LDTA using threshold gates with a fan-in bounded by 5. The threshold gates in this architecture are defined as:

$$A_0 : \mathbf{TH}(a_{2i}, b_{2i}; 1, 1; 2),$$

$$B_0 : \mathbf{TH}(a_{2i}, b_{2i}; 1, 1; 1),$$

$$A_1 : \mathbf{TH}(a_{2i}, b_{2i}, a_{2i+1}, b_{2i+1}; 1, 1, 2, 2; 4),$$

$$B_1 : \mathbf{TH}(a_{2i}, b_{2i}, a_{2i+1}, b_{2i+1}; 1, 1, 2, 2; 3),$$

$$C_0 : \mathbf{TH}(a_0, b_0, c_{-1}; 1, 1, 1; 2),$$

$$C_1 : \mathbf{TH}(a_0, b_0, a_1, b_1, c_{-1}; 1, 1, 2, 2, 1; 4), \text{ where } i = 1, 2, 3.$$

Note that functions  $C_0$  and  $C_1$  in the Fig. 3.2 correspond to the step 1 of the procedure explained above. Functions  $A_0$  and  $A_1$  compute the  $G$  over the ranges  $[2i : 2i]$  and  $[2i + 1 : 2i]$  respectively as in step 2 of the procedure. Functions  $B_0$  and  $B_1$  compute the  $T$  over the same ranges. The four majority gates on the left in the second row of Fig. 3.2 compute the  $G$  and  $T$  functions by combining  $G$  and  $T$  functions in the first row as described in step 3 of the procedure (In this example,

### 3.4. A LOW DEPTH THRESHOLD ADDER (LDTA)

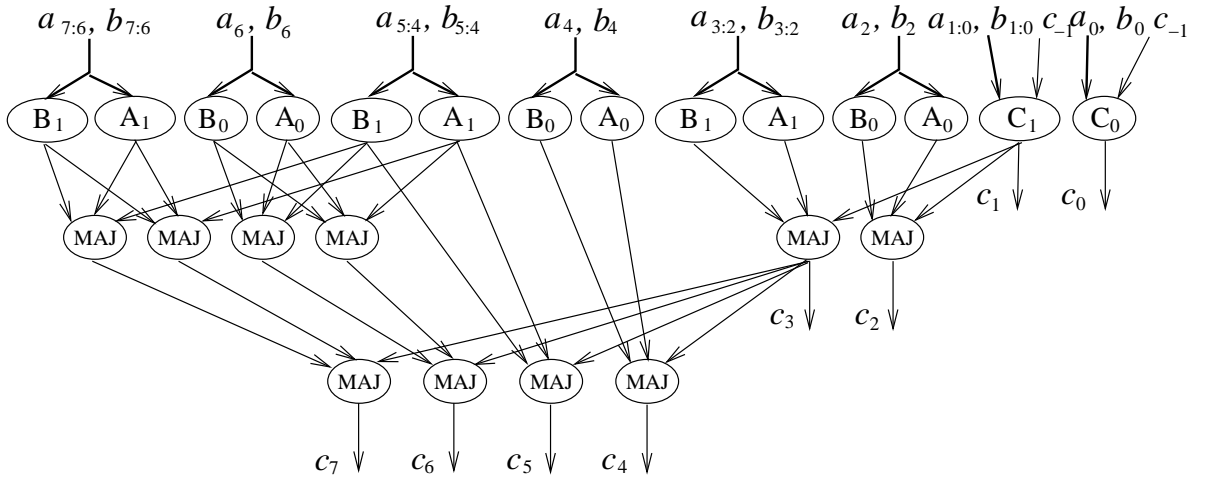


Figure 3.2: Computing all the carries of an 8-bit LDTA with a fan-in bound of 5.

the only value level  $i$  assumes is 0.) Finally, the remaining majority gates in the figure are used to compute the carries as in step 4 of the procedure.

The LDTA procedure described above gives the following Theorem.

**Theorem 3** *All the carries of an  $N$ -bit Low Depth Threshold Adder (LDTA) can be obtained with a depth  $\log_2(4N/M)$  circuit using threshold gates with fan-in bounded by  $M + 1$ . In all levels, except the first, this circuit uses 3-input majority gates. The hardware complexity of this design is  $N(M + 3\log_2(N/M) + 2) + 3M/2 - M^2/4$  inputs.*

*Proof.* Let  $N = 2^n$  and  $M = 2^m$ . Steps 1 and 2 of the LDTA design procedure follow directly from (3.28), (3.31) and (3.34). The threshold functions used in these steps clearly satisfy the fan-in bound.

We now show that the  $G$  and  $T$  required in step 3 for any particular calculation for a given  $i$ ,  $j$  and  $k$  are available either from step 2 or from a smaller  $i$  in step 3. For convenience, we denote the range  $[j2^{i+m} + k : j2^{i+m}]$  in step 3, by  $R(i, j, k)$ . Similarly the range  $[jM/2 + k : jM/2]$  used in step 2 is denoted by  $R'(j, k)$ . To compute  $G$  and  $T$  over  $R(i, j, k)$ , one needs  $T$  and  $G$  over two smaller ranges, namely,

### 3.4. A LOW DEPTH THRESHOLD ADDER (LDTA)

Adder	Delay	Max fan-in	Number of gates
CPA	$N$	4	$2N$
GCLA	$\log N + 2$	9	$(4/3)(N - 1) + 3N$
CLA	$\log N$	$(N/4) + 1$	$(5/2)N$
LDTA	$2 + \log(N/M)$	$2M + 1$	$O(N \log(N/M))$

Table 3.1: Comparison of Carry Lookahead Adder, CPA, GCLA and our LDTA.

$[j2^{i+m} + k : j2^{i+m} + 2^{i+m-1}]$  and  $[j2^{i+m} + 2^{i+m-1} - 1 : j2^{i+m}]$ . One can verify that the second of these ranges is  $R(i-1, 2j, 2^{i+m+1})$ . (For  $i = 0$ , this range is  $R'(2j, 2^{i+m+1})$  in step 2.) Since  $G$ s and  $T$ s for  $i-1$  are computed before those for  $i$ ,  $G$ s and  $T$ s over  $R(i-1, 2j, 2^{i+m+1})$  are available for the computation of  $G$ s and  $T$ s over  $R(i, j, k)$ .

To show that the  $G$ s and  $T$ s over the other smaller range,  $[j2^{i+m} + k : j2^{i+m} + 2^{i+m-1}]$ , are also available for computing  $G$ s and  $T$ s over  $R(i, j, k)$ , we consider cases based upon the value of  $k$ . When  $2^i M/2 \leq k < (2^i + 1)M/2$ ,  $G$ s and  $T$ s over the required range are available from step 2 when they are computed over the range  $R'((2j+1)2^i, k - 2^{i+m-1})$ . When  $i = 0$ , this covers the entire  $k$  range. For other  $i$  values, when  $(2^i + 2^t)M/2 \leq k < (2^i + 2^{t+1})M/2$ ,  $0 \leq t < i$ , the required range is  $R(t, (2j+1)2^{i-t-1}, k - 2^{i+m-1})$ . Thus the required  $G$  and  $T$  over that range are available for computing  $G$ s and  $T$ s over  $R(i, j, k)$ .

Finally, carry computation in step 4 requires  $G$ s and  $T$ s over the range  $[(M/2)2^i + k : (M/2)2^i]$ . It is easy to show that when  $i = 0$ , these are computed in step 2 over the range  $R'(1, k)$  and when  $i > 0$ , in step 3 over the range  $R(i-1, 1, k + 2^i M/2)$ .

The complexity (number of inputs to all gates) of the carry calculation circuit of LDTA can be obtained easily from the complexity of gates in each step in the procedure. Gates used in the steps 1 through 4 require  $M(M+4)/4$ ,  $(2N-M)(M+2)/2$ ,  $3(N(n-m-1)+M)$  and  $3(N-M/2)$  inputs respectively. Adding these, one gets the complexity of  $N$  bit LCTA carry computation as stated in the theorem. ■

The new adder is compared with others available in literature in Table.3.1.

One can see that the new LDTA is the only adder that allows a tradeoff among

### 3.5. A LOW COMPLEXITY THRESHOLD ADDER (LCTA)

the fan-in bound, delay and the complexity. The new adder can achieve a delay better than  $\log N$ . Moreover, it used only majority gates except for the first level.

Another design of adder can be easily derived with the design procedure of LDTA which uses majority gates only including the first level. Also as mentioned before, the sum bits can be implemented with 5-input majority gates. As a result, the whole adder is composed of majority gates only. This makes the design perfectly suitable for Quantum-dot Cellular Automata (QCA) which implements only majority gates and invertors. The design procedure and analysis of the adder is provided in Chapter 6 along with other applications that are developed for QCA.

## 3.5 A Low Complexity Threshold Adder (LCTA)

One of the drawback of the LDTA presented in Section 3.4 is its  $O(N \log(N/M))$  hardware complexity. The principle reason for this can be traced to the fact that in steps 1 and 2 of its design procedure, one applies many of the same inputs to multiple threshold gates. This clearly increases the input complexity of the circuit. One can combine the inputs in various threshold gates without overlap to reduce this complexity. We refer to the resultant adder as the *Low Complexity Threshold Adder* (LCTA). Design of an  $N$ -bit LCTA using threshold functions with fan-in bound of  $M + 1$  can be described as follows. For convenience, assume  $M = 2^m$  and  $N = 2^n$ .

### Design of the Low Complexity Threshold Adder (LCTA).

1. Obtain  $G$  and  $T$  over the range  $[(j+1)M/2 - 1 : jM/2]$ ,  $1 \leq j < 2N/M$  using (3.28) and (3.31).
2. For each level  $i$ ,  $0 \leq i < n - m$ , obtain  $G$  and  $T$  over the range  $[(j+1)2^{i+m} - 1 : j2^{i+m}]$ ,  $1 \leq j < 2^{n-m-i}$  from  $G$  and  $T$  over ranges  $[(j+1)2^{i+m} - 1 : (2j+1)2^{i+m-1}]$  and  $[(2j+1)2^{i+m-1} - 1 : j2^{i+m}]$  using majority gates as in (3.23) and (3.25).
3. Compute carries  $c_{2^i-1}$ ,  $m \leq i \leq n$ , from  $c_{2^{i-1}-1}$  using appropriate  $T$  and  $G$  as in (3.36).



### 3.5. A LOW COMPLEXITY THRESHOLD ADDER (LCTA)

4. For each level  $i$ ,  $0 \leq i < n - m - 1$ , obtain carries  $c_{(4j+2)2^{i+m-1}-1}$ ,  $1 \leq j < 2^{n-m-i-1}$  from  $c_{(4j+1)2^{i+m-1}}$  using appropriate  $T$  and  $G$  as in (3.36).
5. Compute carries  $c_{Mi+M/2-1}$ ,  $0 \leq i < N/M$  from  $c_{Mi-1}$  using appropriate input bits as in (3.37).
6. For each  $i$ ,  $0 \leq i < 2N/M$ , compute carries  $c_{(M/2)i-1+j}$ ,  $1 \leq j < (M/2)$ , from  $c_{(M/2)i-1}$  using appropriate input bits as in (3.37). When  $M = 4$ , these carries can be computed through three input threshold gates.
7. Obtain all the sum bits from the carries using (3.2) or (3.3).

Fig. 3.3 shows a carry computation circuit of a 16-bit LCTA with a fan-in bound of 5, i.e.,  $M = 4$ .

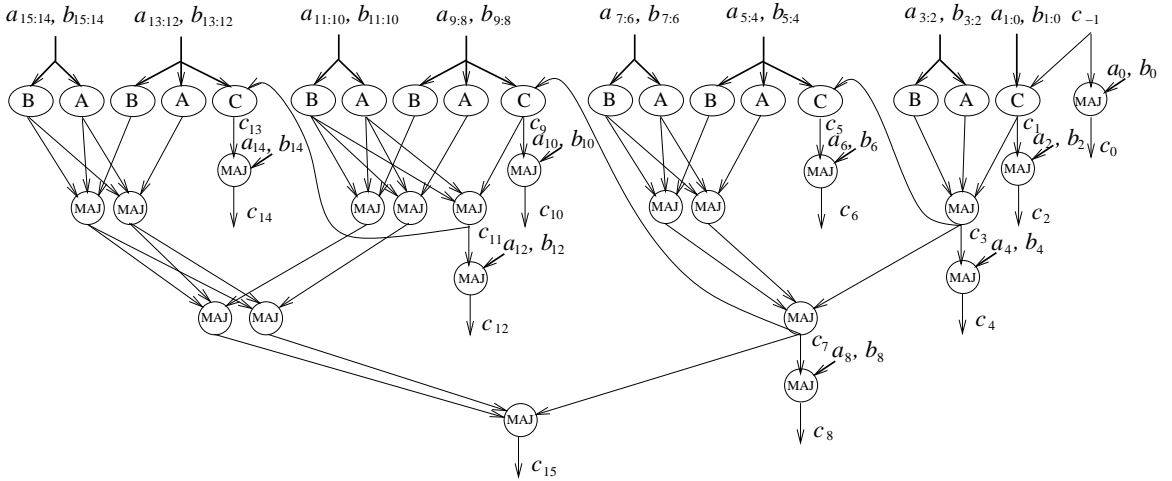


Figure 3.3: Carry computation in a 16 bit LCTA with fan-in bound of 5.

Threshold functions  $A$  and  $B$  in the figure compute  $G$  and  $T$  directly from the inputs as stated in step 1 of the procedure. The three pairs of majority functions in the following level and one pair in the next level correspond to the  $G$  and  $T$  computation described in step 2 of the procedure. Computation of carries  $c_3$ ,  $c_7$  and  $c_{15}$  corresponds to step 3 of the design. Carry  $c_{11}$  is obtained as per step 4 of the

### 3.5. A LOW COMPLEXITY THRESHOLD ADDER (LCTA)

design. Threshold functions  $C$  in the figure compute carries  $c_1$ ,  $c_5$ ,  $c_9$ , and  $c_{13}$  as in step 5. Finally, step 6 provides the remaining carries, namely,  $c_2$ ,  $c_4$ ,  $c_6$ ,  $c_8$ ,  $c_{10}$ ,  $c_{12}$  and  $c_{14}$ .

The procedure described above gives the following Theorem.

**Theorem 4** *All the carries of an  $N$ -bit Low Complexity Threshold Adder (LCTA) can be obtained with a depth  $1 + \log_2(2N/M) \log_2(4N/M)/2$  circuit using threshold gates with fan-in bounded by  $M + 1$ . In all levels, except the first, this circuit uses 3-input majority gates. The hardware complexity of this design is  $N(5 + (M/2) + 14/M) - 2M - 6 \log_2(4N/M)$  inputs.*

*Proof.* Let  $N = 2^n$  and  $M = 2^m$ . It is easy to see that steps 1 and 2 of the procedure provide all the  $T$ 's and  $G$ 's over the range  $[(s + 1)2^t - 1 : s2^t]$ ,  $m - 1 \leq t < n - 1$ ,  $1 \leq s < n - t$ . Therefore the  $T$ s and  $G$ s required for carry computation in steps 3 and 4 are available. Steps 5 and 6 compute carries from smaller carries and input operand bits. The threshold gates used satisfy the fan-in bound of  $M + 1$  because at most  $M$  operand bits are used in any of these computations. Further, because each carry depends on a lower carry, they are all computable.

To prove that the carries computed in steps 3 through 6 cover all the carries  $c_0$  through  $c_{N-1}$ , we first show that steps 3, 4 and 5 provide all the carries with a separation of  $M/2$ . In particular, we show steps 3, 4 and 5 compute carries  $c_i$  such that  $(i + 1)/(M/2)$  take all integer values from 1 to  $N/(M/2)$ . Firstly note that index of  $i$  of every carry  $c_i$  computed in these steps is such that  $(i + 1)$  is divisible by  $(M/2)$ . Thus  $(i + 1)/(M/2)$  is an integer and lies in the specified range. Clearly any positive integer can be uniquely expressed as  $(i + 1)/(M/2) = q2^p$ , for some odd integer  $q$  and a non-negative integer  $p$ . It is easy to check that when  $q = 1$ , the corresponding  $i$ s are the indices of carries computed in step 3. Similarly, when  $p = 0$ , the corresponding  $i$ s are the indices of carries computed in step 5. Finally, when  $q \geq 3$  and  $p > 0$ , the corresponding  $i$ s are the indices of carries computed in step 4. Thus steps 3, 4 and 5 obtain distinct carries and cover all carries separated by  $(M/2)$ . Since step 6 obtains all the intermediate carries, the procedure described here provides all the carries  $c_0$  through  $c_{N-1}$ .

### 3.5. A LOW COMPLEXITY THRESHOLD ADDER (LCTA)

To compute the depth of the LCTA, we first calculate the delay of each carry  $c_j$  for which  $(j+1)/(M/2)$  has integer values. We obtain the depth of such a  $c_j$  by tracing its dependence on the previous carries. Let a carry  $c_{j_1}$  is obtained directly from a carry  $c_{j_2}$ . The discussion above shows that we can obtain odd integers  $q_1$  and  $q_2$  and non-negative integers  $i_1$  and  $i_2$  such that  $(j_1+1)/(M/2) = q_1 2^{i_1}$  and  $(j_2+1)/(M/2) = q_2 2^{i_2}$ . Then steps 3, 4 and 5 of the design procedure then show that these quantities are related as follows.

$$q_2 2^{i_2} = \begin{cases} q_1 2^{i_1-1} & \text{if } q_1 = 1, i_1 \neq 0 & \text{(step 3),} \\ (q_1 - 1) 2^{i_1} & \text{if } i_1 = 0 & \text{(step 5),} \\ (2q_1 - 1) 2^{i_1-1} & \text{if } q_1 > 1, i_1 \neq 0 & \text{(step 4).} \end{cases} \quad (3.38)$$

Carry  $c_{j_1}$  can be obtained from  $c_{j_2}$  if integer  $q_1 2^{i_1}$  can be obtained from  $q_2 2^{i_2}$  using rules specified in (3.38). We will denote this relation between the two integers as  $q_1 2^{i_1} \leftarrow q_2 2^{i_2}$ . We now consider two cases based on of  $q_1$  to obtain the chains of integers that indicate the carries one has to go through to obtain  $c_{j_1}$ .

Case 1.  $q_1 = 1$ .

In this case one can see from (3.38) that the carry dependence is given by the chain

$$2^{i_1} \leftarrow 2^{i_1-1} \leftarrow 2^{i_1-2} \leftarrow \dots \leftarrow 2^0 \leftarrow 0.$$

Note that the last integer of this chain, 0, corresponds to carry  $c-1$ . Since the length of the chain is  $i_1 + 1$ , we conclude that the delay of computing such a  $c_{i_1}$  is  $i_1 + 1$ .

Case 2.  $q_1 > 1$ .

The carry dependence chain in this case is given by

$$\begin{aligned} q_1 2^{i_1} &\leftarrow (2q_1 - 1) 2^{i_1} \leftarrow (4q_1 - 3) 2^{i_1-2} \leftarrow (8q_1 - 7) 2^{i_1-3} \leftarrow \\ &\dots \leftarrow (2^{i_1} q_1 - 2^{i_1} + 1) 2^0 \leftarrow (2^{i_1} q_1 - 2^{i_1}). \end{aligned} \quad (3.39)$$

The last integer in this length  $i_1+1$  chain,  $(2^{i_1} q_1 - 2^{i_1})$  is closely related to the starting integer  $q_1 2^{i_1}$ . In particular, the binary representation of this integer is obtained from the binary representation of  $q_1 2^{i_1}$  by removing the 1 with the minimum weigh (at

### 3.5. A LOW COMPLEXITY THRESHOLD ADDER (LCTA)

bit position  $i_1$ ). Thus each 1 at the  $k$ th bit position in the binary representation of  $q_1 2^{i_1}$  can be removed with a carry chain of length  $k + 1$ . Notice that every time one such 1 is removed, the weight of the representation decreases by 1. Eventually, when only one 1 at position  $t$  is left in the representation, it can be removed with a carry chain of length  $t + 1$ .

From these two cases, one sees that to evaluate the delay of carry  $c_j$ , in  $2^n$  bit LCTA, one should first express  $(j + 1)/(M/2)$  in its binary representation, i.e.,

$$(j + 1)/(M/2) = \sum_{k=0}^{k < n-m} x_k 2^k.$$

The delay  $d(j)$  of carry  $c_j$  is then given by

$$d(j) = \sum_{k=0}^{n-m} x_k (k + 1) \quad (3.40)$$

The worst case delay is therefore given by the carry  $c_j$  when every  $x_k$ ,  $0 \leq k \leq n - m$  equals 1. The carry with the worst delay (within the set of carries with integer values  $(j + 1)/(M/2)$ ) has index  $j = (M/2)(2^{n-m+1} - 1) - 1$  and from (3.40), has a delay of  $(n - m + 1)(n - m + 2)/2$ .

Note that the calculation of the delays here assumes that when a carry is computed, the required  $T$  and  $G$  are already available. We verify this assumption now. Note that steps 1 and 2 compute  $T$  and  $G$  over the range  $[(s + 1)2^t - 1 : s2^t]$  with a delay of  $t - m + 2$ . In step 3, computation of  $c_{2^i-1}$  requires  $c_{2^{i-1}-1}$  and  $T$  and  $G$  over the range  $[2^i - 1 : 2^{i-1}]$ . Clearly the delay of these  $T$  and  $G$  and of  $c_{2^{i-1}-1}$  is  $i - m + 1$ . Similarly in step 4, the carry computation requires  $c_{(4j+1)2^i(M/2)}$  and  $T$  and  $G$  over the range  $[(4j + 2)2^i(M/2) - 1 : (4j + 1)2^i(M/2)]$ . These  $T$  and  $G$  require a delay of  $i + 1$  while the delay of  $c_{(4j+1)2^i(M/2)}$  is at least  $i + m$  since there is a 1 in the bit position  $i + m - 1$  of the carry index. Thus the delays can be calculated only from the delays of carry propagation.

Finally, step 6 of the design procedure shows that the delay of carries dependent on this  $c_j$  is exactly one more than that of  $c_j$ . Therefore the delay of the carry computation in LCTA is  $1 + (n - m + 1)(n - m + 2)/2$ .

The complexity (number of inputs to all gates) of the carry calculation circuit of LCTA can be obtained easily from the complexity of gates in each step in the procedure. Gates used in the steps 1 through 6 require  $4N - 2M$ ,  $12(N/M) - 6(n - m + 2)$ ,  $3(n - m + 1)$ ,  $3(N/M) - 3(n - m + 1)$ ,  $N + (N/M)$  and  $(NM/2) - (2N/M)$  inputs respectively. Adding these, one gets the complexity of  $N$  bit LCTA carry computation as  $N(5 + (M/2) + 14/M) - 2M - 6(n - m + 2)$ . ■

### 3.6 An Enhanced Low Delay Threshold Adder(ELDTA)

The Low Delay Threshold Adder (LDTA) presented in the last section does not fully utilize the power of the threshold functions to implement complex Boolean functions. In particular, steps 3 and 4 of of LDTA design use 3 input majority gates irrespective of the fan-in bound of the technology. Instead, if one uses more complex threshold gates for these computations, one can significantly reduce the hardware complexity (both gates and interconnects) of the adder without significantly impacting the delay.

To achieve this, we first explore computing  $G$  and  $T$  over a larger range from  $G$  and  $T$  over multiple smaller ranges using single threshold functions. This would generalize (3.23) and (3.25). Let  $i \geq l > k > j$  and assume that the  $G$  and  $T$  over sub-ranges  $[i : l]$ ,  $[l - 1, k]$  and  $[k - 1 : j]$  are available, then one can obtain  $G$  and  $T$  over the range  $[i : j]$  as:

$$\begin{aligned}
 G_{i:j} &= G_{i:k} + T_{i:k}G_{k-1:j} \\
 &= G_{i:l} + T_{i:l}G_{l-1:k} + (G_{i:l} + T_{i:l}T_{l-1:k})G_{k-1:j} \\
 &= G_{i:l} + T_{i:l}(G_{l-1:k} + T_{l-1:k}G_{k-1:j}). \\
 &= \mathbf{TH}(G_{k-1:j}, T_{l-1:k}, G_{l-1:k}, T_{i:l}, G_{i:l}; 1, 1, 1, 2, 2; 4), \quad (3.41)
 \end{aligned}$$

where the last step follows from Theorem 1(P5) and  $F_0$  through  $F_5$  are the Fibonacci sequence elements equal to 1, 1, 2, 3, 5 and 8 respectively. (See also examples 5 and 6.)

### 3.6. AN ENHANCED LOW DELAY THRESHOLD ADDER(ELDTA)

Similarly, the  $T$  function may be computed from the three contiguous sub-ranges as

$$\begin{aligned} T_{i:j} &= G_{i:l} + T_{i:l}(G_{l-1:k} + T_{l-1:k}T_{k-1:j}) \\ &= \mathbf{TH}(T_{k-1:j}, T_{l-1:k}, G_{l-1:k}, T_{i:l}, G_{i:l}; 1, 1, 1, 2, 2, 2; 4), \end{aligned} \quad (3.42)$$

Note that both (3.41) and (3.42) use identical threshold functions (same weights and threshold) to compute the pair of  $G$  and  $T$  functions over the range  $[i : j]$ . All, but the first inputs of these functions are identical as well.

Similar to (3.41) and (3.42),  $G$  and  $T$  can also be obtained from  $G$  and  $T$  defined over 4 or more intervals. For example, when  $i \geq l > k > m > j$ ,  $G$  and  $T$  over the range  $[i : j]$  can be obtained from those over  $[i : l]$ ,  $[l - 1, k]$ ,  $[k - 1 : m]$  and  $[m - 1 : j]$  using

$$G_{i:j} = \mathbf{TH}(G_{m-1:j}, T_{k-1:m}, G_{k-1:m}, T_{l-1:k}, G_{l-1:k}, T_{i:l}, G_{i:l}; 1, 1, 1, 2, 2, 4, 4; 8),$$

$$T_{i:j} = \mathbf{TH}(T_{m-1:j}, T_{k-1:m}, G_{k-1:m}, T_{l-1:k}, G_{l-1:k}, T_{i:l}, G_{i:l}; 1, 1, 1, 2, 2, 4, 4; 8).$$

Note that previously,  $G$  and  $T$  were computed either from other  $G$  and  $T$  functions (see (3.23) and (3.25)) or from direct inputs (see (3.28) and (3.31)). To reduce the complexity of the adder, we also allow computation of  $G$  and  $T$  from other  $G$  and  $T$  as well as inputs to the adder. Consider the computation of  $G_{i:j}$  using  $G_{k:j}$  and inputs at bit positions  $k + 1$  through  $i$ . From (3.27) one gets

$$\begin{aligned} G_{i:j} &= a_i b_i + (a_i + b_i)(a_{i-1} b_{i-1} + (a_{i-1} + b_{i-1})(\cdots (a_{k+1} + b_{k+1})G_{k:j} \cdots)) \\ &= \mathbf{TH}(G_{k:j}, a_{k+1}, b_{k+1}, a_{k+2}, b_{k+2}, \dots, a_i, b_i; \\ &\quad 1, 1, 1, 2, 2, \dots, 2^{i-k-1}, 2^{i-k-1}, 2^{i-k}). \end{aligned} \quad (3.43)$$

Note that this last expression is obtained by applying Theorem 1(P4) repeatedly  $i - k$  times.

Similarly, from (3.30) one gets

$$\begin{aligned} T_{i:j} &= a_i b_i + (a_i + b_i)(a_{i-1} b_{i-1} + (a_{i-1} + b_{i-1})(\cdots (a_{k+1} + b_{k+1})T_{k:j} \cdots)) \\ &= \mathbf{TH}(T_{k:j}, a_{k+1}, b_{k+1}, a_{k+2}, b_{k+2}, \dots, a_i, b_i; \\ &\quad 1, 1, 1, 2, 2, \dots, 2^{i-k-1}, 2^{i-k-1}, 2^{i-k}). \end{aligned} \quad (3.44)$$

### 3.6. AN ENHANCED LOW DELAY THRESHOLD ADDER(ELDTA)

Note that the threshold gates used to obtain  $G_{i:j}$  and  $T_{i:j}$  in (3.44) and (3.44) are identical i.e., they use the same weights and the threshold.

Finally, in Section 3.3, carries were calculated either from  $G$  and  $T$  (see (3.21)) or from inputs (see (3.37)). To reduce the complexity, we now compute the carry  $c_i$  from  $G_{k:j}$ ,  $T_{k:j}$ , a previous carry  $c_{j-1}$  and some inputs. From (3.35) one has express  $c_k$  as

$$c_k = G_{k:j} + c_{j-1}T_{k:j} = \mathbf{TH}(c_{j-1}, \mathbf{G}_{k:j}, \mathbf{T}_{k:j}; \mathbf{1}, \mathbf{1}, \mathbf{1}; \mathbf{2}). \quad (3.45)$$

However, using a carry expression as in (3.33), one gets

$$\begin{aligned} c_i = & a_i b_i + (a_i + b_i)(a_{i-1} b_{i-1} + (a_{i-1} + b_{i-1}) \\ & \cdots a_{k+2} b_{k+2} + (a_{k+2} + b_{k+2})(a_{k+1} b_{k+1} + c_k(a_{k+1} + b_{k+1})) \cdots) \end{aligned} \quad (3.46)$$

Since  $c_k$  is a threshold function as in (3.45), one can use Theorem 1(P4) to show that  $c_i$  expression (3.46) is also a threshold function. Applying this theorem repeatedly  $i - k$  times, one gets

$$\begin{aligned} c_i = & \mathbf{TH}(c_{j-1}, G_{k:j}, T_{k:j}, a_{k+1}, b_{k+1}, a_{k+2}, b_{k+2}, \dots, a_i, b_i; \\ & 1, 1, 1, 2, 2, 4, 4, \dots, 2^{i-k}, 2^{i-k}, 2^{i-k+1}). \end{aligned} \quad (3.47)$$

The strategy to design the new  $N$ -bit adder using threshold functions with fan-in bound of  $2M + 1$  can now be described as follows. Here for computational convenience, we assume  $N = (M+1)^{d-M} + M^2 - 1$ . As will be seen in the algorithm,  $d$  represents the depth (number of levels) of the algorithm. We also need the following definitions. Let  $R_i = M(M+1)^{i-M}$ . Define  $s_k^i$ ,  $M+1 \leq i < d$ ,  $0 < s_k^i < N$  recursively as:  $s_0^{M+1} = R_{M+1}$ ,  $s_0^{i+1} = s_0^i + R_i$  and  $s_k^i = s_0^i + kR_i$ .

#### Design of the Enhanced Low Delay Threshold Adder (ELDTA).

1. For each level  $i$ ,  $1 \leq i \leq M+1$ , obtain  $G$  and  $T$  over the range  $[s_k^{M+1} + iM - 1 : s_k^{M+1}]$ , for all  $ks$  such that  $0 < s_k^{M+1} < N$ , from the adder inputs using (3.28) and (3.31), when  $i = 1$ ; and using  $G$  and  $T$  over the range  $[s_k^{M+1} + (i-1)M - 1 : s_k^{M+1}]$  and the proper adder inputs as in (3.43) and (3.44) when  $2 \leq i \leq M+1$ .

### 3.6. AN ENHANCED LOW DELAY THRESHOLD ADDER(ELDTA)

2. For each level  $i$ ,  $M+2 \leq i < d$ , obtain  $G$  and  $T$  over the range  $[s_k^i + jM - 1 : s_k^i]$ , for  $js$  satisfying  $R_{i-1} < jM \leq R_i$ , from  $G$  and  $T$  in level  $(i-1)$  with at most one pair of  $G$  and  $T$  from levels lower than  $(i-1)$  using (3.41) and (3.42).
3. For each level  $i$ ,  $1 \leq i \leq M+1$ , obtain carries  $c_{(i-1)M+k}$ ,  $0 \leq k < M$  from  $c_{(i-1)M-1}$  using (3.37).
4. For each level  $i$ ,  $M+2 \leq i \leq d$ , let  $t_i = s_0^{i-1}$ . Obtain carries  $c_j$ ,  $t_i \leq j < t_{i+1}$ , from  $c_{t_{i-1}}$  and
  - adder inputs, when  $t_i \leq j \leq t_i + M - 2$  using (3.37);
  - $G$  and  $T$  over  $[t_i + Y(j)M - 1 : t_i]$ , where  $Y(j) = \lceil (j - t_i - M + 2)/M \rceil$  and adder inputs, when  $t_i + M - 1 \leq j < t_{i+1} - 1$  using (3.47);
  - $G$  and  $T$  over  $[j : t_i]$ , when  $j = t_{i+1} - 1$  using (3.36).
5. Obtain the sum bits from the carries using (3.2).

Step 1. For level  $i = 1$  each  $G$  and  $T$  are computed from  $2M$  inputs with one threshold gate with fan-in bound of  $2M + 1$ . For  $2 \leq i \leq M + 1$ , one combines  $G$  or  $T$  from level  $i - 1$  and  $2M$  inputs in a threshold gate with fan-in bound  $2M + 1$ .

Step 2. From the definition of  $s_k^i$ , one has

$$s_k^i = s_{k(M+1)+1}^{i-1} \quad \text{and} \quad s_{k+1}^i = s_k^i + R_i. \quad (3.48)$$

With (3.48), one can partition the range  $[s_k^i + jM - 1 : s_k^i]$ ,  $R_{i-1} < jM \leq R_i$  into as many sub-ranges, each with  $R_{i-1}$  elements, as possible:

$$\begin{aligned} [s_k^i + jM - 1 : s_k^i] &= [s_k^i + jM - 1 : s_{k(M+1)+t}^{i-1} + R_{i-1}] \\ &\quad \bigcup_{1 \leq r \leq t} [s_{k(M+1)+r}^{i-1} + R_{i-1} - 1 : s_{k(M+1)+r}^{i-1}]. \end{aligned} \quad (3.49)$$

Note that since the number of elements in the original range,  $jM$ , may not be a multiple of  $R_{i-1}$ , the first subrange may have less than  $R_{i-1}$  elements. The value of  $t$  is given by

$$t = \lceil jM/R_{i-1} \rceil - 1. \quad (3.50)$$



The expression for the first of these subranges can be rewritten as

$$[s_k^i + jM - 1 : s_{k(M+1)+t}^{i-1} + R_{i-1}] = [s_{k'}^{i-1} + jM - tR_{i-1} - 1 : s_{k'}^{i-1}], \quad (3.51)$$

where,  $k' = (k(M+1) + t + 1)$ . When  $i > (M+2)$ , this range is identical to one at level  $i-1$  since  $(jM - tR_{i-1}) < R_{i-1}$  from 3.50. Therefore the required  $G$  and  $T$  over this range are already available. When  $i = M+2$ , using the fact the  $R_{i-1}$  is a multiple of  $M$ , this subrange can be further simplified to

$$[s_k^i + jM - 1 : s_{k(M+1)+t}^{i-1} + R_{i-1}] = [s_{k'}^{M+1} + i'M - 1 : s_{k'}^{M+1}], \quad (3.52)$$

where,  $i' \leq M+1$  because the number of elements in the range  $\leq R_{M+1} = M(M+1)$ . This range is identical to the one used in level  $(M+1)$  of step 1, therefore the required  $G$  and  $T$  over it are available at level  $(M+2)$ .

Similarly, for  $i = M+2$ , the expression for each of the last  $t$  sub-ranges in (3.49),  $[s_{k(M+1)+r}^{i-1} + R_{i-1} - 1 : s_{k(M+1)+r}^{i-1}]$  can be rewritten as  $[s_{k'}^{M+1} + M(M+1) - 1 : s_{k'}^{M+1}]$ , where  $k' = k(M+1) + r$ . This range is thus identical to the range for level  $M+1$  in step 1 of the algorithm. Therefore the  $G$ s and  $T$ s over these ranges are available from level  $M+1$ . Similarly, when  $i > M+2$ , this range,  $[s_{k(M+1)+r}^{i-1} + R_{i-1} - 1 : s_{k(M+1)+r}^{i-1}]$ , is identical to the level  $(i-1)$  range since  $R_{i-2} < R_{i-1} < R_i$ . The required  $G$  and  $T$  over this range are therefore available from level  $i-1$ .

Finally, note that each  $G$  or  $T$  in step 2 is computed from one  $G$  or  $T$  and  $t$  pairs of  $G$  and  $T$ . Thus it is a threshold function of  $2t + 1 = 2\lceil jM/R_{i-1} \rceil - 1$  variables from (3.50). But since  $jM \leq R_i = (M+1)R_{i-1}$ , this threshold function uses at most  $2M+1$  variables and can therefore be implemented by a threshold gate with fan-in bound of  $2M+1$ .

Step 3. Follows directly from (3.37). In each carry calculation, exactly  $2k$  inputs and a previous carry are combined in a threshold gate with  $2k+1$  inputs. Since  $k < M$ , the fan-in bound of  $2M+1$  is satisfied.

Step 4. We first show that the required carry  $c_{t_{i-1}}$  is available for use in level  $i$ . When  $i = M+2$ , this carry,  $c_{M(M+1)-1}$  is the carry of level  $M+1$  with  $k = M-1$  obtained in step 3. If  $i > M+2$ ,  $c_{t_{i-1}}$  is the carry obtained in level  $i-1$  of step 4 with the maximum value of  $j$  for that level.

When  $t_i \leq j \leq t_i + M - 2$ , carry  $c_{t_i-1}$  is combined with adder inputs at bit positions  $t_i$  to  $j$ . Therefore the number of inputs to the threshold gate is  $2(j - t_i + 1) + 1 \leq 2M + 1$ , which clearly satisfies the fan-in bound.

Consider now values of  $j$  satisfying  $t_i + M - 1 \leq j \leq t_{i+1} - 2$ . In this case, one needs  $G$  and  $T$  over the range  $[t_i + Y(j)M - 1 : t_i]$ , where  $Y(j) = \lceil (j - t_i - M + 2)/M \rceil$ . However, for the  $j$  values used in this case,  $1 \leq Y(j) \leq R_{i-1}/M$ . When  $1 \leq Y(j) \leq M + 1$ , the range over which  $G$  and  $T$  are needed can be expressed as  $[s_0^{M+1} + i'M - 1 : s_0^{M+1}]$ , where  $1 \leq i' \leq M + 1$ . These  $G$ s and  $T$ s are available from step 1. On the other hand, when  $R_{M+1}/M = M + 1 < Y(j) \leq R_{i-1}/M$ , we can show that the  $G$ s and  $T$ s needed are available from step 2. In particular, when  $R_{i-1}/M < Y(j) \leq R_{i'}/M$ ,  $M + 2 \leq i' \leq i - 1$ , one can see that the  $G$ s and  $T$ s needed are over the range  $[s_0^{i'} + j'M : s_0^{i'}]$ , where  $j' = Y(j)$  satisfies the conditions required in step 2. Thus these  $G$ s and  $T$ s are available from level  $i' \leq i - 1$  of step 2. To compute carry  $c_j$  in this case, one needs a threshold gate whose inputs are the carry  $c_{t_i-1}$ , a pair of  $G$  and  $T$  over  $[t_i + Y(j)M - 1 : t_i]$  and adder inputs at bit positions  $t_i + Y(j)M$  to  $j$ . Thus its fan-in is  $3 + 2(j - t_i - Y(j)M + 1)$ . However, from the definition of  $Y(j)$ , one has  $j - t_i - Y(j)M \leq M - 2$ . Therefore the threshold gate fan-in is less than or equal to the fan-in bound of  $2M + 1$ .

Finally, when  $j = t_{i+1} - 1$ , the  $G$  and  $T$  over the range  $[t_{i+1} - 1 : t_i]$  are needed. However, this range can be rewritten as  $[s_0^{i-1} + R_{i-1} - 1 : s_0^{i-1}]$ . When level  $i = M + 2$ ,  $R_{i-1} = M + 1$  and therefore one has these  $G$  and  $T$  from step one, level  $M + 1$ . If  $i > M + 2$ , one has the required  $G$  and  $T$  from step 2, level  $i - 1$ . ■

The  $N$ -bit ELDTA,  $N = (M + 1)^{d-M} + M^2 - 1$ ,  $d \geq M$ . has a delay of  $d + 1$ . It uses  $2d(N - M^2 + 1)/(M + 1) + 2M^2 - 2$  threshold gates with a fan-in bound of  $2M + 1 \geq 5$  and has an interconnect complexity of  $MN + 6N - 6M - 4M^2 + (4MN + 6N)/(M + 1) - (2M - 2)(d - M - 1) + (M + 2)((2NM - 2M^3 + 2M)(d - M - 2) - 2N + 4M^2 + 4M)/(M(M + 1))$ .

Figure 3.4 shows a 30 bit ELDTA realized with threshold gates with fan-in bound of 5. The description of the gates in the figure is provided in the legend.

### 3.7. DISCUSSION AND CONCLUSION OF THRESHOLD ADDERS

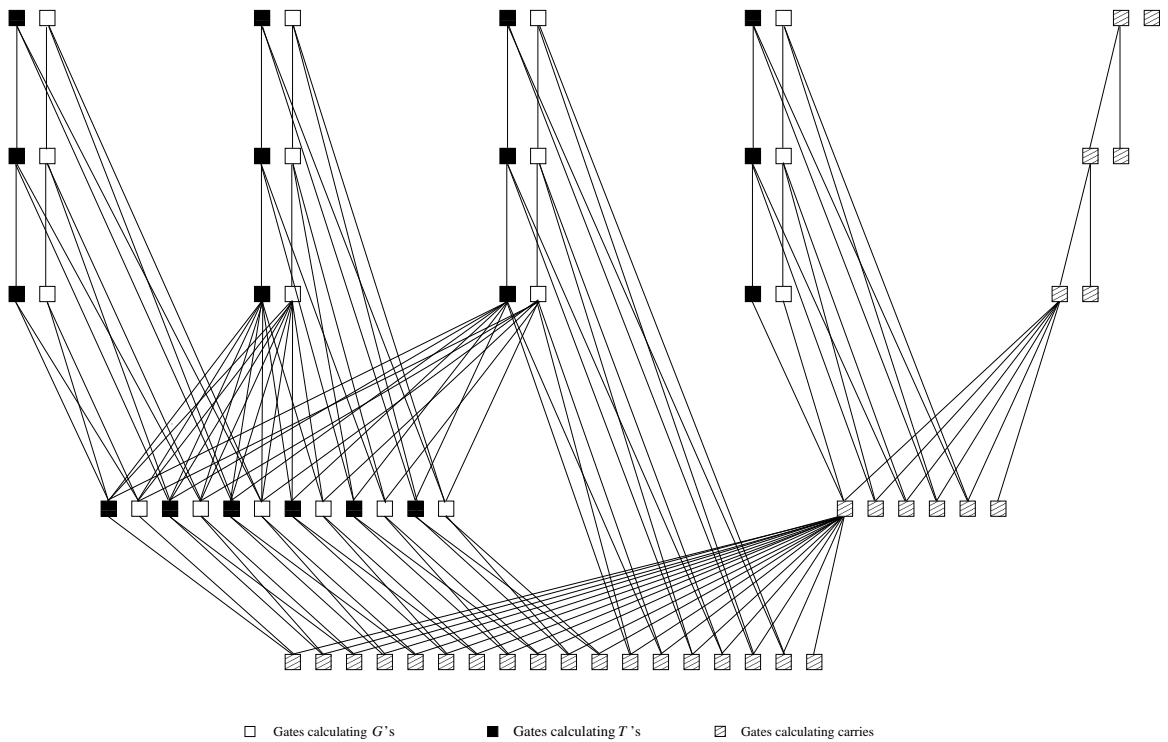


Figure 3.4: The architecture of a 30-bit ELDTA with fan-in  $2M + 1 = 5$ .

Table 3.2: Comparison of the delay of threshold implementations of CPA, GCLA, LDTA and ELDTA using gates with fan-in bound of 5.

$N$	CPA	GCLA	LDTA	ELDTA
8	9	8	4	5
16	17	9	5	6
32	33	12	6	7
64	65	13	7	7
128	129	16	8	8
256	257	17	9	9

## 3.7 Discussion and Conclusion of Threshold Adders

This work has developed a general strategy to design adder architectures directly in terms of threshold functions that are compatible with nanoelectronics. To increase

### 3.7. DISCUSSION AND CONCLUSION OF THRESHOLD ADDERS

Table 3.3: Comparison of the complexity of threshold implementations of CPA, GCLA, LDTA and ELDTA using gates with fan-in bound of 5.

$N$	CPA		GCLA		LDTA		ELDTA	
	gates	inputs	gates	inputs	gates	inputs	gates	inputs
8	16	56	46	142	32	106	18	70
16	32	112	92	284	80	258	42	170
32	64	224	190	590	192	610	102	418
64	128	448	380	1180	448	1410	236	958
128	256	896	766	2382	1024	3202	514	2116
256	512	1792	1532	4764	2304	7170	1174	4848

the implementation reliability, we have assumed that the threshold gates used have a bounded fan-in. Our strategy is based on defining new logic primitives  $G$ s and  $T$ s of operand bits (Section 3.3) such that their evaluation as well as combination requires only threshold gates. These primitives may be computed over appropriate ranges of operand bits, either directly (see (3.28 and (3.31)) or by combining those over smaller ranges (see (3.22) - (3.25)). Once the appropriate  $G$ s and  $T$ s are available, the carries are computed from these with some previous carry (see (3.21)) or directly from the input operand bits and some previous carry (see (3.34) and (3.37)). Sum bits may be computed from the carries in one level of threshold gates. The chosen interdependence between carries (the carry chain) determines the delay and the hardware complexity of the adder. By experimenting with the carry chains, one can choose a suitable compromise between the complexity and the delay.

This work has applied the strategy to obtain an adder architecture, LDTA, with a very low delay. This adder has 3-input majority gates on all but the first and the last levels. Table 3.2 and 3.3 compares the new LDTA architecture with CPA and GCLA.

One can see from Table 3.2 and 3.3 that the complexity of LDTA is comparable to that of GCLA till  $N = 64$ . However, the delay of GCLA is about twice as large as LDTA. The delays of GCLA and LDTA are  $O(\log N)$  and  $O(\log(N/M))$  respectively. Thus, devices with larger fan-in can further decrease the delay of

### 3.7. DISCUSSION AND CONCLUSION OF THRESHOLD ADDERS

LDTA. The CPA complexity is logarithmically lower than that of the LDTA ( $O(N)$  versus  $O(N \log(N/M))$ ). However, as far as the delay is concerned, LDTA is far superior ( $O(\log(N/M))$  as against  $O(N)$ ).

The strategy presented in this work provides a new direction in adder architecture exploration. It is applicable to multiple nanotechnologies because it exploits the identical logic primitives in all these technologies. We believe that one can use the tools developed in this work to design other new adders with the right balance of the delay and the complexity, all the while remaining within the realistic fan-in bounds.

# Chapter 4

## Tree Implementation of Combinational Functions

### 4.1 Introduction

As mentioned in Section 2.6, fan-in is an important factor of reliable digital design with nanotechnology. In [42], a general scheme of threshold logic is provided that can be used to decompose any threshold logic into a network of threshold gates with bounded fan-in. This decomposition employs the novel concept of *error* of the threshold function. It is shown that the value of the error is always non-negative and can be obtained by adding (non-negative) errors due to independent groups of inputs. When the total error of a threshold function exceeds the *critical error*, the output of the function is 0, otherwise it is 1. Critical error is dependent on the weights and the threshold of the function, and plays a central role in the design of our network. The network can be visualized as made up of *fragments* which compute the errors of different groups of inputs and a binary tree of *recombiners* which add these errors to eventually compare it to the critical error.

Our work uses the same concepts of error and critical error for decomposing a threshold logic, though with a different approach to collecting and calculating the amount of error. To be more specific, in [42], a binary tree structure is used

## 4.1. INTRODUCTION

for this task. Our work extends this structure to a more generalized  $k$ -ary tree to provide a greater flexibility in the decomposition architecture. We show that this new procedure results in better performance in some applications. We give the decomposition of comparison logic as an example. The resultant circuit is superior in both speed and hardware complexity.

### 4.1.1 Background

We restate the concepts and some preliminaries of the decomposition scheme from [42] here. Our work is based on the same concepts with extension and generalization.

The classical decomposition of threshold functions exploits the fact that threshold functions are unate<sup>1</sup>. A unate function  $f(x_1, x_2, \dots, x_n)$  can be decomposed as [43]

$$f(x_1, x_2, \dots, x_n) = \begin{cases} f_1(x_2, x_3, \dots, x_n) + x_1 f_2(x_2, x_3, \dots, x_n) & \text{if } x_1 \text{ is positive} \\ f_1(x_2, x_3, \dots, x_n) + \bar{x}_1 f_2(x_2, x_3, \dots, x_n) & \text{if } x_1 \text{ is negative} \end{cases} \quad (4.1)$$

If  $f$  in (4.1) is threshold, then so are  $f_1$  and  $f_2$ . Equation (4.1) can be employed recursively to reduce the fan-in of the threshold functions to any desired small value  $M$  ( $< n$ ). The final decomposition results into a binary tree of  $n - M$  levels with  $2^{n-M} - 1$  internal nodes, each representing the 3-input threshold function and  $2^{n-M}$  leaves representing  $M$ -input threshold functions (not necessarily distinct). Clearly, this decomposition has a depth of  $(n - M + 1)$  which is  $O(n - M)$  and a size of  $O(2^{n-M})$ .

In [44] an alternate decomposition of threshold functions into a network of bounded fan-in threshold functions is proposed. The new decomposition has a polynomial size (with respect to  $n$ ). The blocks to which inputs are applied are called the *Fragments* and the blocks which combine the outputs of the fragments into the final output of the function are called the *Recombiners*. This scheme is shown in Fig. 4.1. We show that each output of a fragment and a recombiner is a threshold

---

<sup>1</sup>A *unate* function is a Boolean function in which every variable is either positive or negative. A variable (say)  $x_1$  is positive in  $f(x_1, x_2, \dots, x_n)$  if  $f(1, x_2, \dots, x_n) \geq f(0, x_2, \dots, x_n)$  for all  $x_2, x_3, \dots, x_n$ . It is negative, if  $f(1, x_2, \dots, x_n) \leq f(0, x_2, \dots, x_n)$ .

#### 4.1. INTRODUCTION

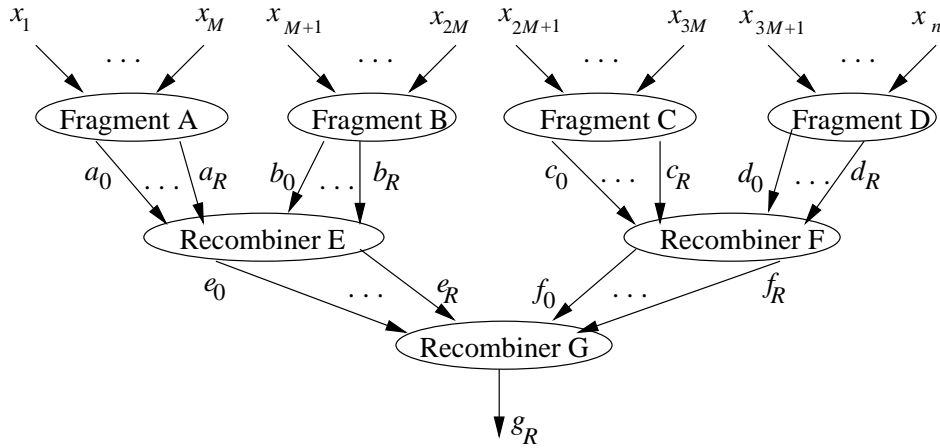


Figure 4.1: A threshold function of  $n$  variables and threshold  $T$  partitioned into four fragments and recreated by a tree of recombiners.  $R$  equals sum of all positive weights minus  $T$ .

function.

A global quantity, *error* of the threshold function is defined to avoid large weighted sum of fragments or recombiners. Error  $E$  of the threshold function  $f$  is defined as

$$E = K - \sum_{i=1}^n w_i x_i, \quad \text{where} \quad K = \sum_{i=1, w_i > 0}^n w_i. \quad (4.2)$$

Clearly,  $E \geq 0$  because  $K$  is the largest value of  $\sum w_i x_i$ . This non-negative character of  $E$  is ideal for information transfer between the fragments and recombiners.

Now define  $R$  as a constant  $R = K - T$ , where  $T$  is the threshold of the function being decomposed. It is easy to see that  $R \geq 0$ ; otherwise the function would always be 0. We can determine the output of the function from  $E$  and  $R$ . To do this, recall that a threshold function  $f(x_1, x_2, \dots, x_n) = 1$  if and only if  $\sum_{i=1}^n w_i x_i \geq T$ . But from (4.2), this condition can be rewritten as

$$f(x_1, x_2, \dots, x_n) = \begin{cases} 1 & \text{if } E \leq R \\ 0 & \text{otherwise.} \end{cases} \quad (4.3)$$

Because of (4.3),  $R$  is referred as the *critical error* of the threshold function.



#### 4.1. INTRODUCTION

In the proposed decomposition, we compute the error of each fragment  $j$  based on the set of indices  $S_j$  that determines its inputs as

$$E_j = K_j - \sum_{i \in S_j} w_i x_i, \quad \text{where} \quad K_j = \sum_{i \in S_j, w_i > 0} w_i. \quad (4.4)$$

Note that similar to  $E$ ,  $E_j$  is also non-negative. By adding  $E_j$ s of all the fragments one gets

$$\begin{aligned} \sum_j E_j &= \sum_j K_j - \sum_j \sum_{i \in S_j} w_i x_i \\ &= K - \sum_{i=1}^n w_i x_i = E. \end{aligned} \quad (4.5)$$

Thus the total error  $E$  needed to establish the value of the function by (4.3) can be obtained by summing  $E_j$ s of individual fragments. We use a binary tree of recombiners to add these  $E_j$ s. Since one is only interested in comparing  $E$  to  $R$ , a fragment or a recombiner only needs to transfer the value of error if it is less than or equal to  $R$ . Thus all the values (of error) transmitted within our network will be bounded by 0 on the lower side and  $R$  on the higher side.

We use  $(R + 1)$  outputs from each fragment to indicate the value of  $E_j$ . The  $t$ th output from a fragment, a Boolean variable  $p_t$ ,  $0 \leq t \leq R$  is defined as

$$p_t = \begin{cases} 1 & \text{if error } E_j \text{ in that fragment} \leq t \\ 0 & \text{if error } E_j \text{ in that fragment} > t \end{cases} \quad (4.6)$$

Note that  $p_t$ s defined by (4.6) are related. In particular, one has

$$p_i \leq p_j \quad \text{if } i < j. \quad (4.7)$$

Equation (4.7) has the following implications which are used later.

$$\begin{aligned} p_i \cdot p_j &= p_i & \text{if } i < j, \\ p_i + p_j &= p_j & \text{if } i < j. \end{aligned} \quad (4.8)$$

The implementation of the fragments is stated as in the following theorem.

#### 4.1. INTRODUCTION

**Theorem 5 (Fragment outputs)** *Each output  $p_t$  of fragment  $j$  is a threshold function of inputs  $x_i$  with weights  $w_i$ ,  $i \in S_j$  and a threshold of  $K_j - t$ .*

Note that using  $(R + 1)$  output lines to carry  $\log_2(R + 1)$  bits of information from a fragment seems wasteful. If one attempts to pass this same value as a binary number of  $\log_2(R + 1)$  bits, then the output lines are indeed reduced, but unlike  $p_t$ s, they are not outputs of threshold functions. The outputs of all the fragments as defined here may be combined by a single threshold gate to provide a two level decomposition of any threshold function as given in the following theorem.

It is also shown that all the outputs of any recombiner are themselves majority functions. Further, each of these functions can be easily decomposed into threshold functions with bounded fan-in.

Note that each recombiner outputs the combined error in all the fragments feeding into it. As before, we are only interested in the error values between 0 and  $R$  and can output these through  $(R + 1)$  lines defined in the same manner as in (4.6). Let  $p_i, q_i, 0 \leq i \leq R$  represent the inputs to a recombiner from its two parents. The  $t$ th output of the recombiner, Boolean variable  $s_t$ , is based on the  $(2t + 2)$  inputs  $p_i, q_i, 0 \leq i \leq t$  and is given by the Boolean expression

$$s_t = \sum_{i=0}^t p_i q_{t-i}, \quad 0 \leq t \leq R. \quad (4.9)$$

Relation (4.9) may be justified by noting that the product  $p_i q_{t-i}$  is 1 only if one parent of the recombiner has at most  $i$  errors and the other, at most  $t - i$  errors. Thus, each term of the summation (4.9) accounts for a case when the total errors are  $t$  or less. Each term is 0 if the combined error from the two parents is greater than  $t$ .

Note that the output  $s_t$  of a recombiner is 1 if the total number of errors in *all* the fragments, of whom it is a descendant, is less than or equal to  $t$ . Because of this similarity of  $s_t$  with the output  $p_t$  of a fragment,  $s_t$  also satisfies the condition (4.7). Thus the output of a recombiner whose parents are recombiners behaves similar to the output of a recombiner whose parents are fragments.

#### 4.1. INTRODUCTION

Theorem 6 describes the threshold nature of each output of a recombiner and Theorem 7 states that it can be decomposed into a network of majority gates.

**Theorem 6 (Recombiner outputs)** *Function  $s_t$  defined by (4.9) is a majority function.*

**Theorem 7 (Recombiner output implementation)** *Each output  $s_t$  defined by (4.9) can be implemented using a multilevel network of generalized majority functions with any given bound on the fan-in.*

The hardware complexity of threshold decomposition can be reduced by making intelligent groupings of the inputs based on their weights. We show that the choice of the partition often affects the complexity of the fragments as well as that of the recombiners. In particular, if the *greatest common divisor* (gcd) of the input weights in a fragment is greater than 1, then some outputs of that fragment and its descendant recombiners may be redundant. Similarly, if the weights in a fragment are small compared with  $R$ , the fragment and its descendant recombiners may have an output redundancy.

Two kinds of redundancies in the fragment and the recombiner outputs are explored. Let the output of a fragment or a recombiner be denoted by  $p_i$ ,  $0 \leq i \leq R$ . We often refer to this entire sequence of outputs simply by  $p$ . When every  $k$  consecutive  $p_i$ s are the same irrespective of the input as in (4.10) below,

$$p_{kt+a} = p_{kt}, \quad 0 \leq a < k, \quad \text{for all } t. \quad (4.10)$$

we say that  $p$  has a *block redundancy* of  $k$ .

The second kind of redundancy arises because of the relationship between  $p_i$ s given in (4.7). This equation suggests that when some  $p_i = 1$ , all subsequent  $p_i$ s are 1 as well. When  $p_i = 1$  for all  $i \geq B$  irrespective of the input, we say that  $p$  is *bounded by  $B$* .

In [44], theorems of hardware reduction for a binary tree decomposition are introduced. We describe them briefly here. The proofs are omitted for brevity. In later section we generalize these theorems to  $k$ -ary tree decomposition.

#### 4.1. INTRODUCTION

The following two theorems show the block redundancy and bound properties of fragments to reduce the number of fragment outputs that are needed to be calculated.

**Theorem 8 (Fragment Redundancy-I)** *If the weights in a fragment have the greatest common divisor of  $g$ , then its output  $p$  has a block redundancy of  $g$ .*

**Theorem 9 (Fragment Redundancy-II)** *Let  $w_i, i \in S_j$  denote the weights of inputs to the  $j$ th fragment. Then the output of the fragment is bounded by  $\sum_{i \in S_j} |w_i|$ .*

The next three theorems allow us to reduce recombiner complexity using input redundancies.

**Theorem 10 (Recombiner Redundancy-I)** *If the input  $p$  of a recombiner has a block redundancy of  $g$ , then the recombiner outputs can be expressed as*

$$s_t = \sum_{i=0}^{\lfloor t/g \rfloor} p_{gi} q_{t-gi}, \quad (4.11)$$

where  $q$  is its other input.

**Theorem 11 (Recombiner Redundancy-II)** *Let inputs  $p$  and  $q$  of a recombiner have block redundancies of  $g_1$  and  $g_2$  respectively. Then its output  $s$  has a block redundancy of  $g = \gcd(g_1, g_2)$ .*

Theorems 10 and 11 play an important role in minimizing the recombiner architecture when both its inputs have block redundancies. Theorem 11 shows that one only needs to compute every  $g$ th output of such a combiner where  $g$  is the gcd of the two block redundancies. Theorem 10 shows that the architecture for each of these outputs can be reduced by a factor equal to the larger of the two redundancies.

We next explore the bounds on the output of a recombiner.

## 4.2. K-ARY TREE DECOMPOSITION STRUCTURE

**Theorem 12 (Recombiner Redundancy-III)** *Let  $p$  and  $q$  denote the inputs of a recombiner and  $g$ , the block redundancy of its output. Also, let  $p_{gi} = 1$  if  $i \geq P$  and  $q_{gi} = 1$  if  $i \geq Q$ . Then the output of the recombiner satisfies*

$$s_{gt} = 1, \quad \text{if } t \geq P + Q. \quad (4.12)$$

Note that Theorems 9 and 12 together imply that a recombiner output is bounded by the sum of absolute values of the weights within all the fragments of whom it is a descendant.

Finally we present a theorem that exploits the input block redundancies together with their bounds.

**Theorem 13 (Recombiner Redundancy-IV)** *Let inputs  $p$  and  $q$  of a recombiner have block redundancies  $g_1$  and  $g_2$  respectively with  $g = \gcd(g_1, g_2)$ . Further, let  $p$  and  $q$  be bounded such that  $p_{ug} = 1$  for  $u \geq P$  and  $q_{vg} = 1$  for  $v \geq Q$ . Then, the output of the recombiner is given by*

$$s_{gt} = p_{(t-Q)g} + q_{\lfloor (t-P)g/g_2 \rfloor g_2} + \sum_{i=\lfloor (t-P)g/g_2 \rfloor + 1}^{Qg/g_2 - 1} p_{gt-ig_2} q_{ig_2} \quad (4.13)$$

## 4.2 k-ary tree decomposition structure

In the previous section, the concept of error is introduced. Also it shows how error is defined in fragments and accumulated through recombiners. In [42], error is accumulated through recombiners that have a binary tree structure. We extend the structure into a  $k$ -ary structure. We show that the new structure can also be used as a general decomposition scheme for threshold logic and that the hardware reduction theorems described in the previous section are also applicable to the  $k$ -ary tree decomposition.

## 4.2. *K*-ARY TREE DECOMPOSITION STRUCTURE

Figure 4.2 shows a general  $k$ -ary tree decomposition structure. Instead of recombining outputs from two parent fragments or recombiners in the binary structure, the number of parent fragments or recombiners that each recombiner recombines is a parameter  $k$ .

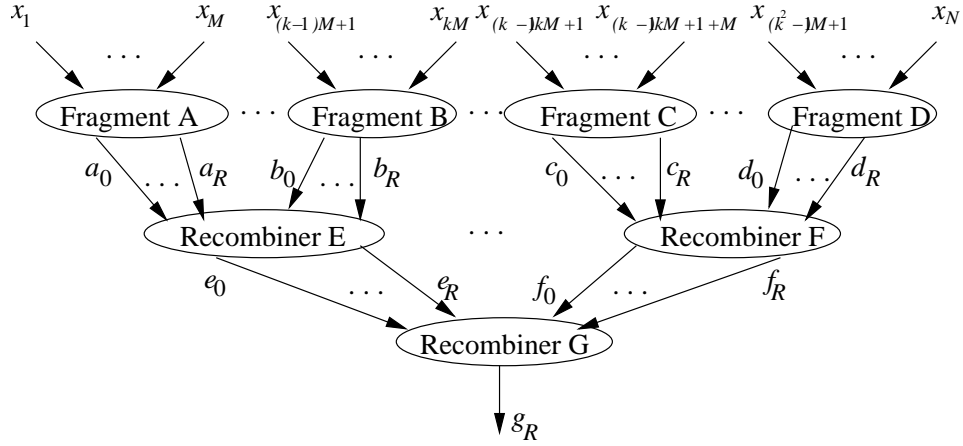


Figure 4.2: A general structure of a  $k$ -ary tree decomposition.

The implementation and properties of the fragments of the  $k$ -ary tree decomposition are the same as those of the binary tree that are shown in Theorem 5 and Equations 4.6, 4.7, 4.8.

Instead of combining error from two parent fragments or recombiners as in the binary tree decomposition, now each recombiner in the  $k$ -ary tree decomposition would combine error from  $k$  parents. We define outputs from the  $j$ -th parent  $p_{i_j}^j$ , where  $0 \leq j \leq k-1$  and  $0 \leq i_j \leq R$ . The output of the recombiner  $s_t$  is 1 if the total error of all the parenting fragments or recombiners is at most  $t$ . Thus  $s_t$  can be given by the Boolean expression

$$s_t = \sum p_{i_0}^0 p_{i_1}^1 \cdots p_{i_{k-1}}^{k-1}, \quad \text{where } i_0 + i_1 + \cdots + i_{k-1} = t \quad (4.14)$$

or

$$s_t = \sum \prod_{j=0}^{k-1} p_{i_j}^j, \quad \text{where } \sum_{j=0}^{k-1} i_j = t \quad (4.15)$$

## 4.2. K-ARY TREE DECOMPOSITION STRUCTURE

Similar to Theorem 6 stating that  $s_t$  is a majority gate, the recombiner output  $s_t$  of the  $k$ -art tree is a generalized majority gate, as stated in the following theorem.

**Theorem 14 ( $k$ -ary tree recombiner outputs)** *Function  $s_t$  defined by 4.15 is a generalized majority function.*

*Proof.* Suppose there exist integers  $d_j$ ,  $0 \leq j \leq k-1$  that

$$p_{i_j}^j = \begin{cases} 0 & \text{if } 0 \leq i_j < d_j \\ 1 & d_j \leq i_j \leq t. \end{cases} \quad (4.16)$$

Then  $s_t$  can be described as

$$\begin{aligned} s_t &= \sum \prod_{j=0}^{k-1} p_{i_j}^j, & \sum_{j=0}^{k-1} i_j &= t \\ &= \sum \prod_{j=1}^{k-1} p_{i_j}^j, & \sum_{j=1}^{k-1} i_j &= t - d_0 \\ &= \dots \\ &= p_{i_{k-1}}^{k-1}, & i_{k-1} &= t - \sum_{j=0}^{k-2} d_j. \end{aligned} \quad (4.17)$$

So for  $s_t$  to be 1, the last term  $p_{t - \sum_{j=0}^{k-2} d_j}^{k-1}$  should also be 1, indicating that  $d_{k-1} \leq t - \sum_{j=0}^{k-2} d_j \leq t$ , thus  $\sum_{j=0}^{k-1} d_j \leq t$ . It also gives us that  $\sum_{j=0}^{k-1} (t+1-d_j) \geq (k-1)t+k$ . Note that  $t+1-d_j$  is the number of  $p_{i_j}^j$ 's that are 1. So the inequality simply means that the total number of outputs from the parents that are 1 should exceed  $(k-1)t+k$ , showing that  $s_t$  is a generalized majority function.  $\blacksquare$

The decomposition of the recombiner output would be very similar to that of a binary tree.

Similar to binary tree,  $k$ -ary tree decomposition also possesses the properties of hardware reduction, including the block redundancy and the bound properties. We now show that all the hardware reduction theorems that are developed for binary tree decomposition are also applicable to the  $k$ -ary tree, giving it the same potential of lowering hardware complexity when applied to specific arithmetic functions.

## 4.2. $K$ -ARY TREE DECOMPOSITION STRUCTURE

Fragment block redundancy property (**Fragment Redundancy-I**) and fragment bound property (**Fragment Redundancy-II**) are the same as shown in the corresponding theorems in [44].

Note that when the output  $p$  has a block redundancy of  $g$ , the only outputs one needs to compute are  $p_{gt}$ ,  $0 \leq t < (R + 1)/g$ . Each output  $p_{gt}$  is a threshold function with threshold  $K_j - gt$ , where  $K_j$  is the sum of all the positive weights of the fragment. Thus all the weights in the fragment as well as its threshold are multiples of  $g$  and consequently  $p_{gt}$  can be implemented as a threshold function with weights  $(w_i/g)$  and a threshold of  $(K_j/g - t)$ . Hence, the conditions of Theorem 8 not only imply fewer threshold functions, but also threshold functions with smaller weights. To illustrate this theorem, consider a fragment with inputs  $x_1, \dots, x_4$  with weights 2,  $-2$ , 4 and  $-6$ . Since the greatest common divisor of the weights is 2, the outputs of the fragment can be shown to be

$$p_{2t+1} = p_{2t} = \mathbf{TH}(x_1, x_2, x_3, x_4; 1, -1, 2, -3; 3 - t).$$

The second kind of redundancy shows up in the fragment output when the weights of inputs to a fragment are small relative to the critical error. In this case, the fragment contributes only small errors leading to the bounds on its output.

Theorem 9 is important in reducing the complexity of a fragment that has small weights in relation to weights of the other fragments. To illustrate this theorem, once again consider the fragment with weights 2,  $-2$ , 4 and  $-6$ . The output of this fragment is bounded by 14, i.e., output  $p_i = 1$ , for all  $i \geq 14$  irrespective of the input. Thus, no matter how large  $R$  is, one need not compute these  $p_i$ s.

We generalize the hardware reduction theorems for recombiners for  $k$ -ary tree and show the proof of them.

**Theorem 15 ( $k$ -ary tree recombiner redundancy-I)** *If the input  $p^0$  has a block redundancy of  $g$ , then the recombiner outputs can be expressed as*

$$s_t = \sum_{i_0=0}^{\lfloor t/g \rfloor} p_{gi_0}^0 \prod_{j=1}^{k-1} p_{i_j}^j, \quad \sum_{j=1}^{k-1} i_j = t - gi_0. \quad (4.18)$$



## 4.2. K-ARY TREE DECOMPOSITION STRUCTURE

*Proof.* As the input  $p^0$  has a block redundancy of  $g$ , then  $p_{gi_0+i'_0}^0 = p_{gi_0}^0$ , where  $0 \leq i'_0 < g$ . Thus

$$s_t = \sum_{i_0=0}^{\lfloor t/g \rfloor} p_{gi_0}^0 \prod_{j=1}^{k-1} p_{i_j}^j, \quad \sum_{j=1}^{k-1} i_j = t - gi_0 - i'_0. \quad (4.19)$$

From Equations 4.8, the summation equals the term with the largest index, which is  $t - gi_0$ , meaning  $\sum_{j=1}^{k-1} i_j = t - gi_0$ .  $\blacksquare$

**Theorem 16 ( $k$ -ary tree recombiner redundancy-II)** *If the input  $p_{i_j}^j$  has a block redundancy of  $g_j$  and  $g = \gcd(g_j)$ ,  $0 \leq j < k-1$ , then  $s_t$  has a block redundancy of  $g$ .*

*Proof.* We will show that the recombiner output  $s_{gt+a}$ ,  $0 \leq a < g$  is independent of  $a$ . Since the block redundancy  $g_j$  is a multiple of  $g$ , from Theorem 15,

$$\begin{aligned} s_{gt+a} &= \sum_{i_0=0}^t p_{gi_0}^0 \left( \sum_{j=1}^{k-1} \prod_{i_j=0}^{i_j} p_{i_j}^j \right), \quad \sum_{j=1}^{k-1} i_j = gt + a - gi_0 \\ &= \sum_{i_0=0}^t p_{gi_0}^0 \left( \sum_{i_1=0}^{t-i_0} \left( \sum_{j=2}^{k-1} \prod_{i_j=0}^{i_j} p_{i_j}^j \right) \right), \quad \sum_{j=2}^{k-1} i_j = gt + a - gi_0 - gi_1 \\ &= \dots \\ &= \sum_{i_0=0}^t p_{gi_0}^0 \sum_{i_1=0}^{t-i_0} p_{i_1}^1 \cdots \sum_{i_{k-2}=0}^{t-\sum_{j=0}^{k-3} i_j} p_{i_{k-2}}^{k-2} p_{i_{k-1}}^{k-1}, \quad i_{k-1} = gt + a - g \sum_{j=0}^{k-2} i_j. \end{aligned} \quad (4.20)$$

Since  $p_{i_{k-1}}^{k-1}$ 's have a block redundancy that is also a multiple of  $g$ ,

$$p_{g(t-\sum_{j=0}^{k-2} i_j)+a}^{k-1} = p_{g(t-\sum_{j=0}^{k-2} i_j)}^{k-1}. \quad (4.21)$$

Combining Equation 4.20 and 4.21 shows that  $s_{gt+a}$  is independent of  $a$ .  $\blacksquare$

Theorems 15 and 16 can be used to reduce the hardware by minimizing the number of outputs of a recombiner that need to be calculated. Next we generalize the bound property of recombiners to the  $k$ -ary tree.

**Theorem 17 ( $k$ -ary tree recombiner redundancy-III)** *Let  $g$  denote the block redundancy of a recombiner with inputs  $p_{i_j}^j$  and let  $p_{gi_j}^j = 1$  if  $i_j \geq d_j$ , then the*

### 4.3. COMPARISON FUNCTION DECOMPOSITION

recombiner output satisfies

$$s_{gt} = 1, \quad \text{if } t \geq \sum_{j=0}^{k-1} d_j. \quad (4.22)$$

*Proof.* The recombiner output can be expressed as

$$\begin{aligned} s_{gt} &= \sum \prod_{j=0}^{k-1} p_{i_j}^j, \quad \text{where } \sum_{j=0}^{k-1} i_j = gt. \\ &= \sum \prod_{j=0}^{k-2} p_{i_j}^j p_{gt - \sum_{j=0}^{k-2} i_j}^{k-1}. \end{aligned} \quad (4.23)$$

If  $t \geq \sum_{j=0}^{k-1} d_j$ , then  $gt \geq g \sum_{j=0}^{k-1} d_j$ . Let  $i_j = gd_j$ ,  $0 \leq j < k-1$ , so that  $p_{i_j}^j = 1$  for  $0 \leq j < k-1$ , meaning the first  $k-1$  terms in the product term  $\prod_{j=0}^{k-2} p_{i_j}^j p_{gt - \sum_{j=0}^{k-2} i_j}^{k-1}$  are 1. Since  $gt \geq g \sum_{j=0}^{k-1} d_j$ , the index of the last term in the product  $gt - \sum_{j=0}^{k-2} i_j \geq gd_{k-1}$ , indicating the  $k$ -th term in the product  $p_{gt - \sum_{j=0}^{k-2} i_j}^{k-1} = 1$ . Then at least one product term in the summation on the right side of Equation 4.23 is 1, meaning  $s_{gt} = 1$ . ■

So far we have explored the implementation and hardware reduction properties of the fragments and recombiners of a  $k$ -ary decomposition tree. We omit the generalization of the last theorem developed for the binary tree for simplicity since it is simply a combination of the block redundancy and the bound property of the recombiners. With the extended theorems developed for the  $k$ -ary tree decomposition structure, we are ready to explore their applications for key arithmetic digital circuits.

## 4.3 Comparison Function Decomposition

Two  $N$ -bit numbers  $\mathbf{x} = \langle x_{N-1}, \dots, x_2, x_1, x_0 \rangle$  and  $\mathbf{y} = \langle y_{N-1}, \dots, y_2, y_1, y_0 \rangle$  may be compared by the threshold function

$$\mathbf{TH}(\mathbf{x}, \mathbf{y}; \mathbf{w}, -\mathbf{w}; 0), \quad (4.24)$$

### 4.3. COMPARISON FUNCTION DECOMPOSITION

where vector  $\mathbf{w} = \langle 2^{N-1}, \dots, 2^2, 2, 1 \rangle$ . The output of this threshold function is 1 if  $\mathbf{x} \geq \mathbf{y}$ .

One can see that the comparison threshold function (4.24) is not a member of  $\widehat{LT}_1$  since its weights increase exponentially with  $n$ . In fact, it is often used to show that  $\widehat{LT}_1$  is a proper subset of  $LT_1$ . This function has attracted quite a bit of attention [39, 45] because the number of inputs and the weights in this function get rather large with an increase in  $n$ . To compare two 32 bit numbers as in (4.24), one needs a threshold function with 64 inputs and weights as large as  $2^{31}$ . [42] shows that the methods allow one to decompose (4.24) in a variety of ways. For example, this 64 input threshold function can be decomposed into 16 identical fragments and 15 identical recombiners arranged in a depth 5 network. Each fragment of this network will have two 4-input threshold functions with a maximum weight of 2. Each recombiner will also be made up of two 4-input threshold functions with a maximum weight of 2.

While binary tree serves greatly for comparators with sizes that are exponents of 2. The  $k$ -ary tree gives a more general structure of decomposing the comparison function into bounded fan-in threshold network. For example, Figure 4.3 shows the decomposition of comparing two 18-bit numbers with a fan-in bounded by 4. Although it is more practical in real computers and digital circuits to have structures adapted to  $N = 2^n$ -bit numbers, the  $k$ -ary structure provides an alternate of decomposition and apply better to specific arithmetic functions. Here we show that the  $k$ -ary tree decomposes comparator into bounded threshold gate networks with lower hardware complexity and comparable depth.

The gates shown in the figure are defined as followed.

$$A : \mathbf{TH}(a_{4j+1}, a_{4j}, b_{4j+1}, b_{4j}; 2, 1, -2, -1; 0), \quad (4.25)$$

$$B : \mathbf{TH}(a_{4j+1}, a_{4j}, b_{4j+1}, b_{4j}; 2, 1, -2, -1; 1), \quad (4.26)$$

$$C : \mathbf{TH}(B_2, \mathbf{TH}(A_2, B_1, A_1, A_0; 3, 2, 1, 1; 5); 1, 1; 1). \quad (4.27)$$

The function  $C$  is a serial expansion expressed as  $B_2 + A_2(B_1 + A_1A_0)$ . Serial expansions can be easily decomposed into threshold functions with bounded fan-in.

### 4.3. COMPARISON FUNCTION DECOMPOSITION

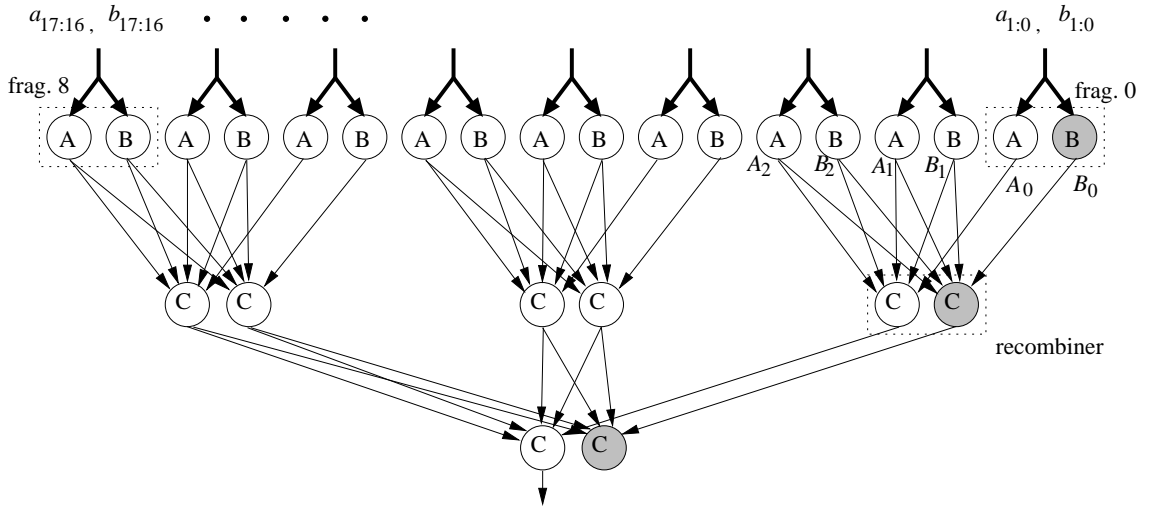


Figure 4.3: A ternary tree decomposition of a comparator of size 18 with fan-in bounded by 4.

In this case, there are 5 literals in the expression of  $C$  which exceeds the fan-in bound of 4. Thus it is implemented with 2 threshold gates in series.

The resultant network has one level of fragments, each composes of two threshold gates, followed by 2 levels of recombiners. Due to the feature of weights of input bits, we show that the number of outputs from each recombiner is reduced to at most 2. The correctness of the decomposition follows from the following theorem.

Here is the implementation of 2  $N$ -bit numbers comparator using a  $k$ -ary tree with bounded fan-in of  $2M$ . We define  $G_l = 2^{Mk^{l-1}}$  for convenience.  $G_l$  has the following property which would be used later.

$$G_l = (G_{l-1})^k. \quad (4.28)$$

And also for convenience, we define  $q_i = p_{gi}$  when a recombiner has a gcd of  $g$  and output  $p_i$ .

**Theorem 18** *To implement a comparison of 2  $N$ -bit numbers using a  $k$ -ary tree with bounded fan-in of  $2M$ , the only outputs required from any recombiner are  $p_{gt_i}$*

### 4.3. COMPARISON FUNCTION DECOMPOSITION

and  $p'_{gt_l}$ , where  $t_l = G_{l+1} - 1$ ,  $t'_l = G_{l+1} - 2$ , and  $g$  is the block redundancy of that recombiner.

*Proof.* We prove that  $q_{t'_l}$  and  $q_{t_l}$  can be expressed as

$$q_{t'_l} = \sum_{m=0}^{k-1} q_{t'_{l-1}}^m \prod_{n=m+1}^{k-1} q_{t_{l-1}}^n, \quad (4.29)$$

$$q_{t_l} = \sum_{m=1}^{k-1} q_{t'_{l-1}}^m \prod_{n=m+1}^{k-1} q_{t_{l-1}}^n + \prod_{n=0}^{k-1} q_{t_{l-1}}^n. \quad (4.30)$$

$q_{t'_{l-1}}$  and  $q_{t_{l-1}}$  correspond to the outputs from the  $(l-1)$ -th level. We first prove (4.29).

The proof of (4.29) can be separated into two parts. First we will prove that if the righthand of (4.29) is 0, then so is  $q_{t'_{l-1}}$ . Note that the case when there is minimum error in  $i$ -th recombiner in level  $l$  when the output is 0 happens when  $q_{t'_{l-1}}^m = 0$ ,  $m = 0, 1, \dots, k-1$ . Thus the total error in  $i$ -th recombiner in level  $l$  is bigger than or equal to the product of  $\sum_{m=0}^{k-1} (t'_{l-1} + 1)$  and the gcd of  $(ki + m)$ -th recombiner in level  $l-1$ . So the total error satisfies

$$\begin{aligned} &\geq \sum_{m=0}^{k-1} (G_l - 1) G_l^{ki+m} \\ &= G_l^{ki} (G_{l+1} - 1) \\ &> G_l^{ki} (G_{l+1} - 2), \end{aligned} \quad (4.31)$$

and (4.31) is the product of  $t'_l$  and the gcd of  $i$ -th recombiner in level  $l$ . Therefore, as the error of this recombiner exceeds the critical error, the output  $q_{t'_l} = 0$ .

Then we prove that if the righthand of (4.29) is 1, then so is  $q_{t'_l}$ . We can prove that by assuming any term of the summation equal to 1, that is,

$$q_{t'_{l-1}}^m \prod_{n=m+1}^{k-1} q_{t_{l-1}}^n = 1. \quad (4.32)$$

So the total error in  $i$ -th recombiner in level  $l$  is

$$\begin{aligned} &\leq (t'_{l-1}) G_l^{ki+m} + t_{l-1} \sum_{n=m+1}^{k-1} G_l^{ki+n} \\ &= (G_l - 2) G_l^{ki+m} + G_l^{ki} (G_l^k - G_l^{m+1}) \\ &= G_l^{ki} (G_l^k - 2G_l^m). \end{aligned} \quad (4.33)$$

### 4.3. COMPARISON FUNCTION DECOMPOSITION

And (4.33) is smaller than or equal to the total critical error of the  $i$ -th recombiner in level  $l$ . Therefore,  $q_{t'_l} = 1$ .

The proof of (4.30) is similar, which can be divided into three parts. First we prove that if  $\sum_{m=1}^{k-1} q_{t'_{l-1}}^m \prod_{n=m+1}^{k-1} q_{t_{l-1}}^n$  and  $\prod_{n=0}^{k-1} q_{t_{l-1}}^n$  both are 0, then so is  $q_{t_l}$ . The total error in  $i$ -th recombiner in level  $l$  is counted to be

$$\begin{aligned}
&\geq \sum_{m=1}^{k-1} (t'_{l-1} + 1)G_l^{ki+m} + (t_{l-1} + 1)G_l^{ki+n} \\
&= G_l^{ki}(G_l^{k-1} - 1)G_l + G_l^{ki+n+1} \\
&= G_l^{ki}(G_{l+1} - 1) + G_l^{ki+n+1} \\
&> G_l^{ki}t_l,
\end{aligned} \tag{4.34}$$

which is the critical error of that recombiner. Thus  $q_{t_l} = 0$ .

Then we prove if  $\sum_{m=1}^{k-1} q_{t'_{l-1}}^m \prod_{n=m+1}^{k-1} q_{t_{l-1}}^n = 1$ , then  $q_{t_l} = 1$ . As we can also prove by assuming any term in the summation to be 1, this part of proof is similar to the second part of proof of (4.29).

Last we prove that if  $\prod_{n=0}^{k-1} q_{t_{l-1}}^n = 1$ , then  $q_{t_l} = 1$ . As each term of the product should be 1, the total error in  $i$ -th recombiner in level  $l$  is

$$\begin{aligned}
&\leq t_{l-1} \sum_{n=0}^{k-1} G_l^{ki+n} \\
&= G_l^{ki}(G_l^k - 1) \\
&= G_l^{ki}t_l.
\end{aligned} \tag{4.35}$$

Thus we can get  $q_{t_l} = 1$ . ■

The depth and the hardware complexity of the comparison decomposed with  $k$ -ary tree is given in the following theorem. Without loss of generality, we analyze the depth and hardware complexity for complete tree only, namely, for 2  $N$ -bit number comparison,  $k$  is such that  $N = Mk^n$ , where the fan-in bound is  $2M$ .

**Theorem 19** *A comparison function of two  $N = Mk^n$ -bit numbers can be decomposed into a network of threshold gates with fan-in bounded by  $2M$  with a  $k$ -ary tree*

### 4.3. COMPARISON FUNCTION DECOMPOSITION

*structure.* The network has a depth of  $1 + (\log_k N/M) \lceil (2k-2)/(2M-1) \rceil$  and a gate complexity of  $2N/M - 1 + \lceil (2k-2)/(2M-1) \rceil (2(N-M)/M(k-1) - \log_k N/M)$ . The interconnect complexity is  $4N - 2M + (2k-2 + \lceil (2k-2)/(2M-1) \rceil) (2(N-M)/M(k-1) - \log_k N/M)$ .

*Proof.* From Theorem 18, the recombiners of a  $k$ -ary tree comparator can be expressed with a serial expansion in the same form described in (P5) in Chapter 2. Along with (P7) we get that since the number of literals in the expression of the recombiner output is  $2k-1$ , each recombiner output needs  $\lceil (2k-2)/(2M-1) \rceil$  gates to implement. The total number of inputs to these gates is then  $2k-2 + \lceil (2k-2)/(2M-1) \rceil$ . With this, the depth, number of gates and interconnect complexity of a  $k$ -ary tree comparator can be obtained simply by calculation.

The  $k$ -ary tree comparator has one level of fragments and  $\log_k N/M$  levels of recombiners. Each recombiner level is composed of  $\lceil (2k-2)/(2M-1) \rceil$  gates since there are  $2k-1$  literals. Thus this leads the total depth to be  $1 + (\log_k N/M) \lceil (2k-2)/(2M-1) \rceil$ .

The architecture has  $N/M$  fragments, each of which has two threshold gates according to Theorem 18 except for the least significant one, which has only one. Thus the number of gates in the fragments is  $2N/M - 1$ . The total number of inputs to these fragment gates is thus  $2M(2N/M - 1)$  since the fan-in bound is  $2M$ .

For each level  $l$  of the recombiners, where  $1 \leq l \leq n$ , there are  $N/(Mk^l)$  recombiners, each of which has two outputs except for the least significant one. Thus the total number of outputs required from the  $l$ -th level recombiners is  $2N/(Mk^l) - 1$ . Each of these outputs has  $2k-1$  literals, thus is implemented with  $\lceil (2k-2)/(2M-1) \rceil$  gates. The interconnect complexity of each recombiner output is  $2k-2 + \lceil (2k-2)/(2M-1) \rceil$ . Summing up all  $n$  levels of recombiners gives the total number of gates and the total number of inputs as provided in Theorem 19. ■

## 4.4 Conclusion

This chapter has focused on developing a  $k$ -ary tree architecture to decompose any threshold function in  $\widehat{LT}_1$  into a network of threshold functions with bounded fan-in. This work uses the same concepts of errors and critical error as in [44], but we extend the binary tree structure developed in [44] to a  $k$ -ary tree structure, where  $k$  is a parameter of the network, which makes the design more flexible for different fan-in bound values. By choosing the right value of  $k$ , one can get resultant circuits with lower depth and lower hardware complexity.

With the new architecture, the implementation of the recombiners becomes complex. Thus, we also extend the hardware complexity reduction properties of fragments and recombiners outputs from binary tree structure to  $k$ -ary tree structure. Most of these properties can be extended in the similar manner, which are useful in developing our application circuits.

We show the decomposition of a comparison function using a  $k$ -ary tree structure. Due to the relation among the weights of the inputs of the function, we eliminate the complexity of the recombiner implementation by using the extended hardware complexity reduction theorems. As a result, the comparison function has a  $O(\log N)$  depth and a  $O(N)$  gate count and interconnect complexity.

Table 4.1 and 4.2 shows a comparison of the depth, gate complexity and interconnect complexity of the comparison function with a binary ( $k = 2$ ) tree and a quaternary ( $k = 4$ ) tree with fan-in bounded by  $4(M = 2)$  and  $8(M = 4)$ , respectively.

One can see that when fan-in bound is 4, the depth of both comparators is the same, while the quaternary tree comparator has slightly improved gate and interconnect complexity. However in the case when fan-in bound is 8, while the interconnect complexity is comparable, the quaternary tree has very improved depth and gate complexity. One can see that for all values of  $N$ , quaternary tree has almost half as much depth as the binary tree implementation. As for the gate complexity, the quaternary tree structure improves the binary tree structure by around 30%.

Since the binary tree has all 3-input majority gates in the recombiners levels,



#### 4.4. CONCLUSION

Table 4.1: Comparison of binary tree comparator and quaternary tree comparator with fan-in bound of 4.

$N$	k=2			k=4		
	delay	gate complexity	interconnect complexity	delay	gate complexity	interconnect complexity
8	3	11	40	3	9	36
32	5	57	202	5	47	188
128	7	247	868	7	205	820
512	9	1013	3550	9	843	3372
2048	11	4083	14296	11	3401	13604

Table 4.2: Comparison of binary tree comparator and quaternary tree comparator with fan-in bound of 8.

$N$	k=2			k=4		
	delay	gate complexity	interconnect complexity	delay	gate complexity	interconnect complexity
16	3	11	68	2	8	63
64	5	57	326	3	39	304
256	7	247	1376	4	166	1289
1024	9	1013	5594	5	677	5250
4096	11	4083	22484	6	2724	21115

when fan-in bound is small (e.g. 4), it fits the 3 inputs into one threshold gate fine. However, it fails to fully harness the larger fan-in bound of threshold gates. As a result, for example, when fan-in is bounded by 8, it wastes more than half of the inputs to each threshold gate. Thus the gate count would be large for binary tree implementation. However, the  $k$ -ary tree structure allows one to choose how many inputs to each recombiner gate by changing the value of  $k$ . Thus one can choose the right value of  $k$  to make full use of the fan-in of each threshold gate. Recall that the number of literals of each recombiner gate is  $2k - 1$ , thus for fan-in equal to 8,  $k = 4$  would be the best choice of  $k$  to fit all inputs into one gate. In fact, it is always good to choose  $k$  to be equal to  $M$  so that each recombiner output is implemented

#### 4.4. CONCLUSION

with only one gate thus to take full advantage of large fan-in bounds.

# Chapter 5

## Systolic Implementation of Threshold Function Decomposition

### 5.1 Introduction

#### 5.1.1 Systolic Architecture

Systolic Architecture is developed for many hardware implementation given the advantages of

- Local communications
- regularity
- cost efficiency.

Systolic architecture is well developed in signal processing including FFT implementation [46], image processing [47], and convolution [48], etc. We take convolution as an example to illustrate the functioning of systolic architectures.

In this chapter, we develop a systolic architecture for implementing threshold logic using threshold gates with bounded fan-in. Systolic architecture is well suitable

## 5.1. INTRODUCTION

for nanoelectronics because of two major features,

- localized connections implying less complex interconnects, and
- bounded fan-in.

However, the 4-phase clocking scheme does not fit the systolic architecture. As mentioned before, the 4-phase clock is perfect for pipelined architecture where the data flows in one direction. For example, a combinational logic circuit can be implemented as a tree such that the computation only goes in one direction. However, for more complex architectures such as a systolic architecture, where the data flow is in both directions, a new clocking scheme is required, which will be described in the subsequent sections.

### 5.1.2 Decomposition of Threshold Functions

As mentioned before, it is practically important to restrain the fan-in of threshold networks to a bounded number. We explore some decomposing strategy for threshold networks.

In [44], a strategy of decomposing threshold functions into a network of bounded fan-in threshold functions is proposed. Here we use the same definition and implementation of the outputs from the fragments and the recombiners as in [44]. Also some of the hardware complexity reduction theorems developed in [44] are useful to our systolic architecture implementation. Thus we restate these theorems here.

**(Fragment outputs)** Each output  $p_t$  of fragment  $j$  is a threshold function of inputs  $x_i$  with weights  $w_i$ ,  $i \in S_j$  and a threshold of  $K_j - t$ .

**(Fragment Redundancy-II)** Let  $w_i$ ,  $i \in S_j$  denote the weights of inputs to the  $j$ th fragment. Then the output of the fragment is bounded by  $\sum_{i \in S_j} |w_i|$ .

**(Recombiner outputs)** Function  $s_t$  defined by (4.9) is a majority function.

In [44], a binary tree is used to recombine the signals carrying the error. In our work, we use a different recombining approach, which enables us to implement it using a novel systolic system.

## 5.2 Systolic System for Nanotechnology

### 5.2.1 Clocking Scheme

First we propose a new clocking scheme that is different from the classic four-phase clock. Fig.5.1 shows the diagram of the new six-phase clocking scheme.

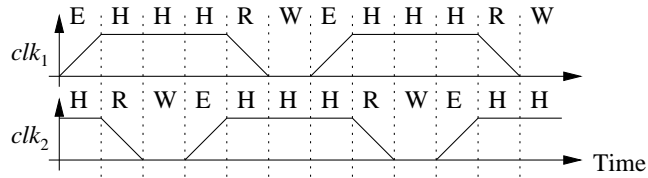


Figure 5.1: A six-phase clocking scheme with the evaluate (E), hold (H), reset (R) and wait (W) states.

If two threshold gates are cascaded, with this new clocking scheme, we can see that when one of them is in its evaluation phase. Each clock has 3 holding phase so that gate with  $clk_1$  holds the data longer enough for  $clk_2$  to evaluate. it is guaranteed that the other one is providing stable data and vice versa.

### 5.2.2 General Scheme of Decomposing a Threshold Function with Systolic System

Any large threshold function can be implemented as a tree of threshold functions as in Fig. 5.2.  $N$  and  $M$  is the number of inputs and the fan-in bound of the circuit, respectively.  $g_R$  is the output of the system and  $R$  denotes the critical error.

As shown in the figure, we recombine two groups of outputs from two fragments first and then add one group at a time.

As described in Theorem 5, for Fragment  $i$  in Fig. 5.2, there are  $(R+1)$  threshold gates. The threshold gate gives the output  $p_j^i$  of the fragment is described as

$$p_j^i = \mathbf{TH}(x_{iM}, x_{iM+1}, \dots, x_{(i+1)M-1}; w_{iM}, w_{iM+1}, \dots, w_{(i+1)M-1}; \sum_{k=iM, w_k > 0}^{(i+1)M-1} w_k - j). \quad (5.1)$$

## 5.2. SYSTOLIC SYSTEM FOR NANOTECHNOLOGY

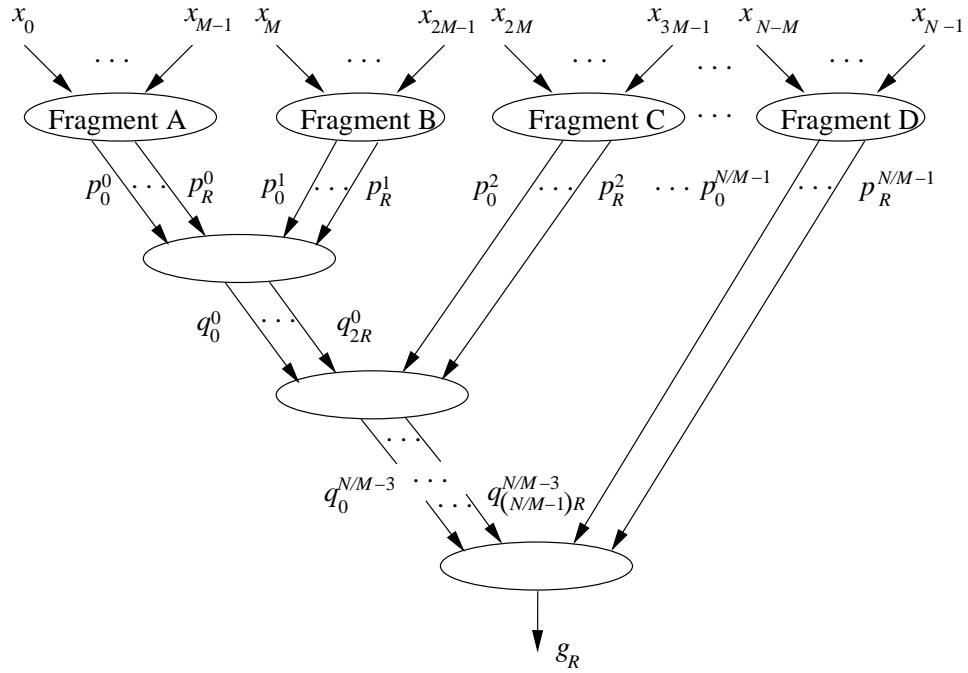


Figure 5.2: General scheme of decomposing a threshold function by serially combining the fragment outputs.

As we can see from Fig. 5.2, the outputs of the fragments are parallel. Since the systolic system implementation requires one group of the inputs to be parallel while the other serial, we use the following scheme to convert one group of the outputs from parallel to serial.

In Fig. 5.3,  $z$  works as an outside control signal determining whether the gates load or shift. When  $z = 0$  at the first clocking cycle when gates  $C_i$  are triggered, the gates  $C_i$  are loaded with  $p_i^0$  from the fragments. After that,  $C_i'$  are clocked, they are loaded with  $p_i^0$ . Then  $z$  is set to 1 afterwards so the convertor does the shifting operation. Buffers  $B_i$  are used to temporally store the data  $p_i^0$  so that when gates  $C_i'$  are triggered in the shifting operation, they get the data shifted from the left gate, thus forming a serial sequence of  $p_i^0$ .

By adding error from one fragments to the total each time, one can get the total error of the inputs and compare it with the critical error of the threshold function.

## 5.2. SYSTOLIC SYSTEM FOR NANOTECHNOLOGY

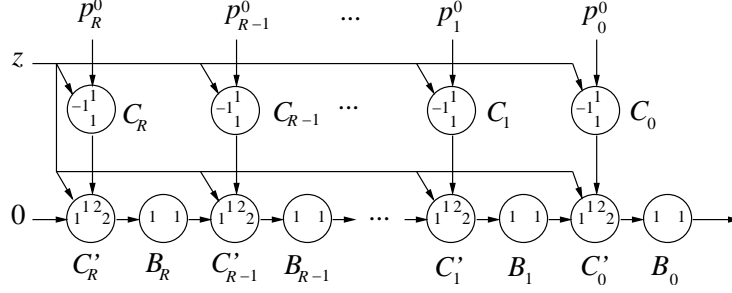


Figure 5.3: Parallel to serial convertor.

Thus  $g_R$  can be obtained as

$$g_R = \sum_{i_{N/M-1}} \left( \sum_{i_{N/M-2}} \cdots \left( \sum_{i_2} \left( \sum_{i_1} \left( \sum_{i_0} p_{i_0}^0 p_{i_1}^1 \right) p_{i_2}^2 \right) \cdots p_{i_{N/M-2}}^{N/M-2} \right) p_{i_{N/M-1}}^{N/M-1} \right). \quad (5.2)$$

We implement the calculation of  $g_R$  using the novel systolic system. Each systolic stage is used to add the error of one fragment to the total error. Thus the recombining part of the circuit is implemented as shown in Fig. 5.4.

To validate the functionality of the systolic implementation of the recombining circuit, without loss of generality, we now show that the first recombining stage that recombines  $p_{i_0}^0$  and  $p_{i_1}^1$  can be implemented using systolic array as shown in Fig. 5.5.

We apply the six-phase clock and its delayed version to alternate gates in Fig. 5.5. Since all the loops in the systolic array shown here have an even length, the data applied to each gate input stays stable while that gate is computing.

We use induction to prove that this systolic array would output the recombining data of  $p_{i_0}^0$  and  $p_{i_1}^1$ . We only show the case when  $R$  is even. The verification with odd  $R$  is similar.

It is clear from Fig. 5.5 that when the corresponding gate is evaluated, we have the following.

$$x_{2i+1} \leftarrow x_{2i},$$

$$x_{2i} \leftarrow x_{2i-1},$$

5.2. SYSTOLIC SYSTEM FOR NANOTECHNOLOGY

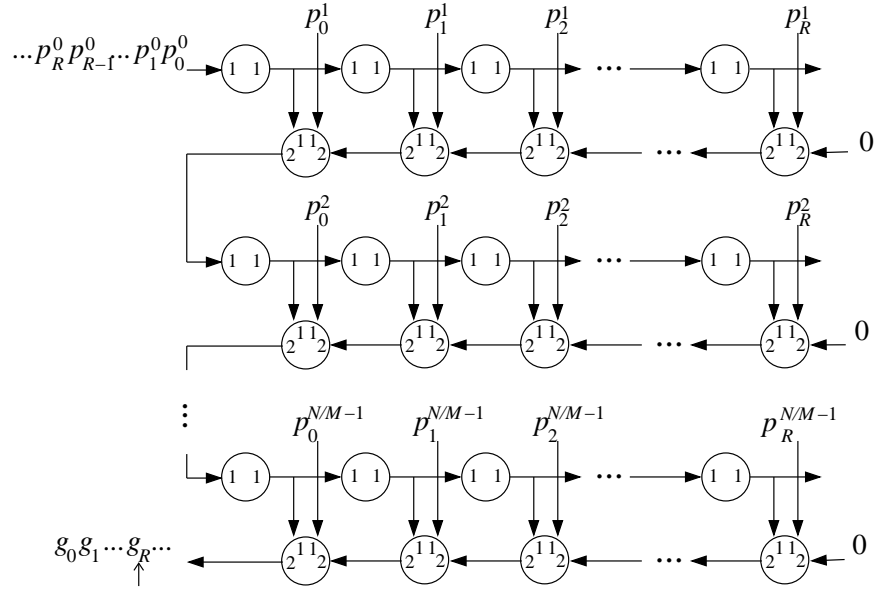


Figure 5.4: Systolic implementation of the recombiners in Fig.5.2.

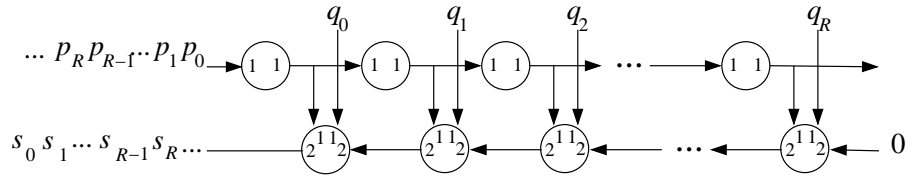


Figure 5.5: The first recombining stage.

$$y_{2i} \leftarrow x_{2i} p_{2i}^1 + y_{2i+1},$$

$$y_{2i+1} \leftarrow x_{2i+1} p_{2i+1}^1 + y_{2i+2}.$$

With these, we now show that at cycle  $t$  after  $y_{2i+1}$  and  $x_{2i}$  are evaluated, the data held in each gate is

$$x_{2i} = p_{t-i}^0,$$

$$x_{2i+1} = p_{t-i-1}^0,$$



## 5.2. SYSTOLIC SYSTEM FOR NANOTECHNOLOGY

Table 5.1: data of gates at cycle  $t$  and  $t + 1$ .

	$x_{2i}$	$x_{2i+1}$	$y_{2i}$	$y_{2i+1}$
at cycle $t$ after clocking $x_{2i}$ and $y_{2i+1}$	$p_{t-i}^0$	$p_{t-i-1}^0$	$\sum_{j=0}^{t-1-i} p_j^0 p_{t-j+i-1}^1$	$\sum_{j=0}^{t-1-i} p_j^0 p_{t-j+i}^1$
at cycle $t$ after clocking $x_{2i+1}$ and $y_{2i}$	$p_{t-i}^0$	$p_{t-i}^0$	$p_{t-i}^0 p_{2i}^1$ $+ \sum_{j=0}^{t-1-i} p_j^0 p_{t-j+i}^1$ $= \sum_{j=0}^{t-i} p_j^0 p_{t-j+i}^1$	$\sum_{j=0}^{t-1-i} p_j^0 p_{t-j+i}^1$
at cycle $t + 1$ after clocking $x_{2i}$ and $y_{2i+1}$	$p_{t-i+1}^0$	$p_{t-i}^0$	$\sum_{j=0}^{t-i} p_j^0 p_{t-j+i}^0$	$p_{t-i}^1 p_{2i+1}^1$ $+ \sum_{j=0}^{t-1-i} p_j^0 p_{t-j+i+1}^1$ $= \sum_{j=0}^{t-i} p_j^0 p_{t-j+i+1}^1$
at cycle $t + 1$ after clocking $x_{2i+1}$ and $y_{2i}$	$p_{t-i+1}^0$	$p_{t-i+1}^0$	$p_{t-i+1}^0 p_{2i}^1$ $+ \sum_{j=0}^{t-i} p_j^0 p_{t-j+i+1}^1$ $= \sum_{j=0}^{t-i+1} p_j^0 p_{t-j+i+1}^1$	$\sum_{j=0}^{t-i} p_j^0 p_{t-j+i+1}^1$

$$y_{2i+1} = \sum_{j=0}^{t-1-i} p_j^0 p_{t-j+i}^1,$$

$$y_{2i} = \sum_{j=0}^{t-1-i} p_j^0 p_{t-j+i-1}^1.$$

As the two clocks are applied to gates alternately, the data of each gate in cycle  $t$  and the following  $t + 1$  can be evaluated as in Table 5.1.

### 5.2.3 The Whole System

To show how the whole system is connected, we show the fragments, parallel to serial convertor and the first systolic stage in Fig. 5.6. The whole system is connected in the same manner.

Before input  $\mathbf{x}$  is applied to the fragments, control signal  $z$  is held high. Because of the 0 applied, the output of every gate in the parallel to serial convertor and systolic recombiner is reset. This works as initialization of the whole system before the input comes in.

## 5.2. SYSTOLIC SYSTEM FOR NANOTECHNOLOGY

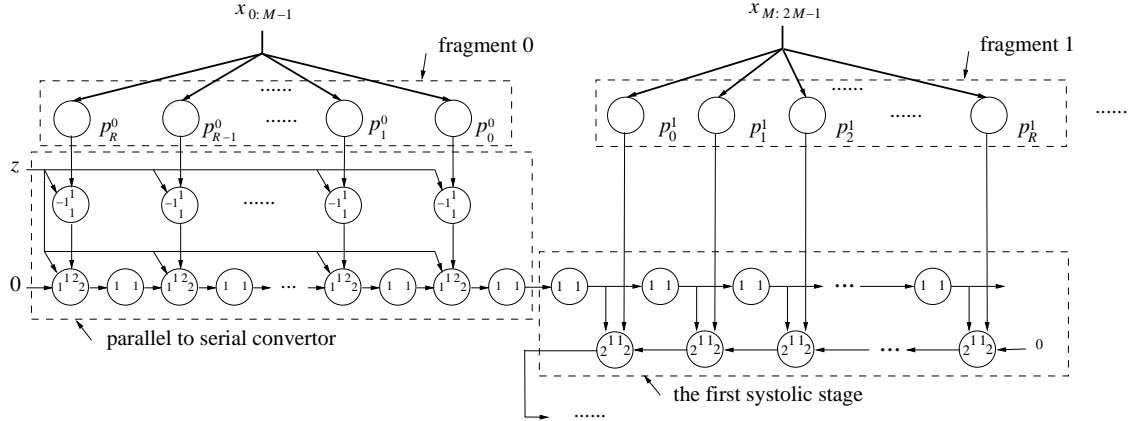


Figure 5.6: The first stage of the complete system.

When the input  $\mathbf{x}$  is applied, as soon as the gates of the fragments have steady output  $p_i^j$ ,  $0 \leq i < M$ ,  $0 \leq j < N/M$ , the clocking of those gates is held high to make sure  $p_i^j$  is available throughout the whole computational period.

The complexity and the delay analysis of the decomposed network is described in the following theorem.

**Theorem 20** *An  $N$  bit threshold function with critical error  $R$  can be decomposed into a network of threshold functions with fan-in  $\leq M$  for any  $M$ . This network has a size  $3NR/M + 5N/M + R - 1$  and a time delay  $N/M + R + 1$ .*

*Proof.* With the description of the fragments, parallel to serial converter and systolic recombiner, it is trivial to get the number of gates needed for each part is  $(R + 1)N/M, 3(R + 1)$ , and  $2(R + 2)(N/M - 1)$ , respectively. Summing up the three gives the total number of gates as stated. Note that with the new clocking scheme, two consecutive gates form a clocking cycle. And the unit of time delay stated is clocking cycle. Thus the fragments and the parallel to serial converter together has a delay of 2, adding to the delay of the recombiner, which is  $(N/M - 1) + R$ , gives the total system delay.  $\blacksquare$

## 5.3 Applications and Examples

We provide two example applications using the systolic implementation of decomposition.

It is interesting to note that it is possible to implement *any* generalized majority function with  $2^n$  inputs using the *same* circuit, despite the threshold of the generalized majority function. Similarly, the exactly *same* systolic architecture can be employed in every  $2^n$  bit approximate pattern matching application irrespective of the number of errors that is to be tolerated.

### 5.3.1 Majority Gate

**Theorem 21** *The  $N$  bit generalized majority function with any value of threshold can be decomposed into a network of threshold functions with fan-in  $\leq M$  for any  $M$ . This network has a size  $3N + 2N/M + M - 2$  if  $M \leq R + 1$ , or  $3N^2/M - 3TN/M + 5N/M + N - T - 1$  if  $M > R + 1$ , and a time delay  $N + N/M - T + 1$ , where  $T$  is the threshold of the generalized majority gate and  $R$  is the critical error.*

*Proof.* Recall Theorem 9, as the weight of each input bit is 1, the bound of each fragment gate is  $M$ . Thus the number of gates in each fragment is either  $M$  or  $R + 1$ , depending on which is less. When  $M > R + 1$ , the gates in fragments are bounded by  $R + 1$ , then the hardware complexity and the time delay is exactly the same as stated in Theorem 20. For the case when  $M \leq R + 1$ , then the fragment gates are bounded by  $M$ . Substituting  $R + 1$  with  $M$  gives the complexity and delay of the case. Note that for generalized majority gate,  $R = N - T$ . ■

An example of a 16-bit majority function with a fan-in bound 4 is given in Fig. 5.7. Note that for a 16-bit majority function, the threshold  $T = 9$  and the critical error  $R = 7$ . Thus,  $g_7$  is the output.

The fragment gates are threshold gates as follows.

A:  $\mathbf{TH}(x_{4i}, x_{4i+1}, x_{4i+2}, x_{4i+3}; 1, 1, 1, 1; 1)$ ,

B:  $\mathbf{TH}(x_{4i}, x_{4i+1}, x_{4i+2}, x_{4i+3}; 1, 1, 1, 1; 2)$ ,

### 5.3. APPLICATIONS AND EXAMPLES

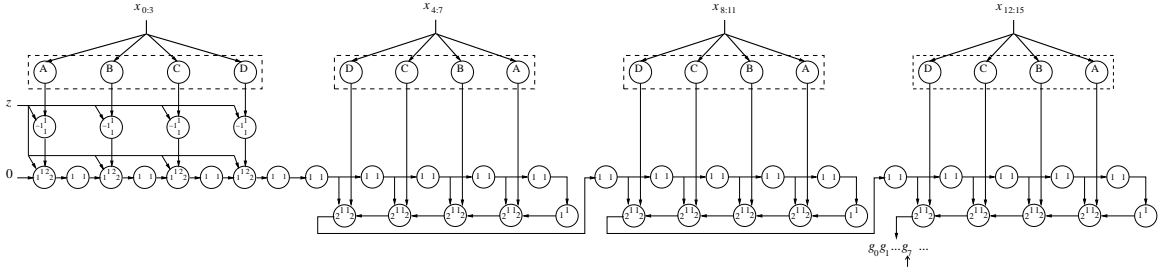


Figure 5.7: A 16-bit majority gate with fan-in bound 4.

C:  $\mathbf{TH}(x_{4i}, x_{4i+1}, x_{4i+2}, x_{4i+3}; 1, 1, 1, 1; 3)$ ,

D:  $\mathbf{TH}(x_{4i}, x_{4i+1}, x_{4i+2}, x_{4i+3}; 1, 1, 1, 1; 4)$ .

The rest of the circuit consists of the parallel to serial convertor and 3 systolic stages, which are described in Sec. 5.

#### 5.3.2 Pattern Matching Machine

In many quality control and robotics applications, one has to compare a pattern captured by sensors with a stored template. In most of these applications the comparison needs to allow for a certain number of sensor errors. It is known that this problem of *error tolerant pattern matching* for binary patterns can be solved by a single threshold logic circuit [42]. Let binary vectors  $\mathbf{x}$  and  $\mathbf{y}$  denote the input and the template respectively. The weight vector is created from  $\mathbf{y}$  by replacing all the zeros in it by  $-1$ s. The threshold is chosen to be  $wt(\mathbf{y}) - \epsilon$ , where  $\epsilon$  is the error tolerance and  $wt(\mathbf{y})$  denotes the weight of the  $\mathbf{y}$ . For example, the threshold function

$$\mathbf{TH}(\mathbf{x}; 1, 1, 1, -1, -1, 1, -1, 1; 2)$$

will output a 1 if the 8-bit input vector  $\mathbf{x}$  matches with the pattern  $\langle 1, 1, 1, 0, 0, 1, 0, 1 \rangle$  with three or less errors.

In most applications, the number of inputs to this threshold function may get very large, rendering the threshold function impractical. In such cases, the methods of this work can be used to decompose the function into smaller threshold functions as described by the following theorem.

#### 5.4. CONCLUSION AND DISCUSSION

**Theorem 22** *The  $N$  bit pattern matching function detecting any number of errors can be decomposed into a network of threshold functions with fan-in  $\leq M$  for any  $M$ . This network has a size  $3N+2N/M+M-2$  if  $M \leq R+1$ , or  $3NR\varepsilon/M+5N/M+\varepsilon-1$  if  $M > R+1$ , and a time delay  $N/M + \varepsilon + 1$ , where  $\varepsilon$  is the number of errors.*

The proof of the Theorem 22 is very similar to that of Theorem 21. Note that the critical error  $R = \varepsilon$ .

### 5.4 Conclusion and Discussion

This chapter proposes a novel decomposing scheme for general threshold functions, which are implementable with nanotechnology. The resultant circuits have bounded fan-in for the sake of reliability. Except for the first two levels, input fragments and parallel to serial converter, respectively, the circuit is implemented using systolic architecture, which leads to very low hardware complexity. With such architecture, we are able to add the error of one fragments each time to the total to get the total error of the circuit, which would be compared to the critical error to determine the output of the threshold logic.

We show examples of two key logic, generalized majority gate and pattern matching machine to illustrate the general scheme of decomposing threshold functions. Table 5.2 and Table 5.3 give the comparison of the gate count and depth of the majority gate implemented with the new scheme to that proposed in [1] and [2]. As can be seen from the table, the depth of the majority gate is sacrificed, however, the saving in hardware complexity is considerable. Take  $N = 64$  and fan-in bound  $M = 4$  as an practical example, the speed is slowed down, but the hardware is only about 0.14% of the original implementation.

The systolic system is able to implement the recombiners of any decomposed threshold gate by combining errors from two fragments at first and adding error from one more fragment each time. The resultant architecture has a gate count of  $O(NR/M)$  and a time complexity of  $O(N/M+R)$ , where  $R$  is the critical error of the threshold gate. It is obvious that the critical error  $R$  is crucial to both the hardware

#### 5.4. CONCLUSION AND DISCUSSION

Table 5.2: Size of majority gate implemented with bounded fan-in. Results of [1,2] are shown in parenthesis for comparison.

N	M			
	2	4	8	16
4	16 (7)			
8	32 (31)	30 (13)		
16	64 (511)	58 (77)	58 (25)	
32	128 (16383)	114 (2493)	110 (217)	114 (49)
64	256 (1.05E6)	226 (1.6E5)	214 (13945)	214 (689)

Table 5.3: Depth of majority gate implemented with bounded fan-in. Results of [1,2] are shown in parenthesis for comparison.

N	M			
	2	4	8	16
4	4 (3)			
8	8 (6)	6 (3)		
16	16 (10)	12 (6)	10 (3)	
32	32 (15)	24 (10)	20 (6)	18 (3)
64	64 (21)	48 (15)	40 (10)	36 (6)

complexity and the time complexity. For application for systolic implementation of threshold gate decomposition, we also analyze the comparison function, which is also a very important arithmetic unit used in computers. The resultant decomposed

#### 5.4. CONCLUSION AND DISCUSSION

circuit turns out to have large gate count and delay. The reason for this is analyzed thus. Unlike generalized majority function and pattern matching function, which has unity weights, the comparison function has exponential weights, leading to large bounds of fragments. Combined with the fact that the critical error of a comparison function is also large ( $2^N - 1$  for the comparison of two  $N$ -bit numbers), the number of gates each systolic stage is composed of is large as well. For example, an 8-bit comparison function with fan-in bound of 4 will have 4 fragments in the first level, which have bounds of 6, 24, 96, 384, respectively. The critical error of this function is 255. Thus, the number of gates for each systolic stage is 194, 50, 17, respectively. And due to the large critical error, the delay of the implementation is also large.

Thus one can see that, since the time and hardware complexity of a decomposed threshold gate implemented with systolic system is dependant with the critical error and the bounds of the fragments. Also the bounds of the fragments are related to the weights of the inputs. The systolic implementation is ideal for threshold gates with smaller weights. As a result, the generalized majority function and the pattern matching function are well suited applications of the systolic implementation scheme.

The decomposing scheme that is proposed in this work fits for any general threshold function. It provides a reliable implementation of threshold functions with bounded fan-in and extremely compact architecture.

# Chapter 6

## Digital Circuits for Quantum-Dot Cellular Automata

### 6.1 Introduction

As mentioned in Chapter 5.1, QCA is one of the most interesting realizations of threshold gates. QCA can be implemented in many technologies including ferromagnetic and molecular. The molecular QCA is particularly interesting because of its projected density of up to  $1 \times 10^{12}$  devices per  $\text{cm}^2$  [3]. A major advantage of QCA over other nanoelectronic architectural styles is that the same cells that are used for making logic gates can be used to build wires carrying logic signals. However, QCA architectures have to rely upon only two basic building blocks, namely a three input majority gate and an inverter. As a result, QCA implementations of only a few logic circuits including binary adders, multipliers, barrel shifters, serial/parallel converters are currently available [49–51].

Although a majority gate can convert to an AND and OR gate easily by tying one of its inputs to 0 and 1, respectively so that all digital circuits can be translated to ones that compose of majority gates only. However, again this way of direct translation will waste the logic of majority gate as well as one-third of the interconnects of the circuits. In this chapter, we develop efficient implementation of



## 6.2. QUANTUM-DOT CELLULAR AUTOMATA

key arithmetic circuits of comparison and addition using majority gates only. The resultant circuits will thus be well suited for QCA implementation.

## 6.2 Quantum-Dot Cellular Automata

A basic QCA building block can be described as a cell with four quantum dots and two charged particles may occupy the dots. The charged particles can migrate between quantum dots when the barriers between them are lowered by an external *clock*. When the barriers are raised by lowering the clock, the particles settle into two possible stable (polarized) positions. These stable positions represent logic 0 and 1 as shown in Fig. 6.1.

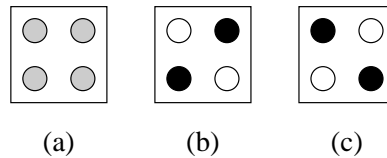


Figure 6.1: (a) The basic QCA cell (b) a polarized QCA cell representing logic 1 and (c) a polarized QCA cell representing logic 0.

When the clock to a QCA cell is lowered, the polarization states of its surrounding cells determines its own polarization state. This enables one to design an inverter and a three input majority gate shown in Figs. 6.2 and 6.3.

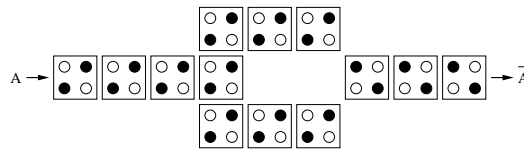


Figure 6.2: An inverter implementation in QCA.

A three input majority gate outputs a logic 1 when 2 or more of its inputs are 1. By fixing one of the inputs to the majority gate at 1, one can convert the majority

### 6.3. COMPARISON FUNCTION

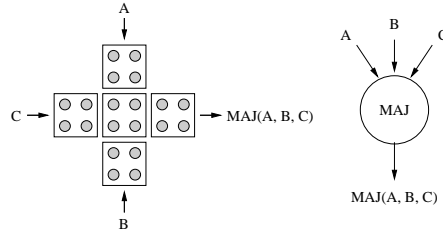


Figure 6.3: A 3-input majority gate implementation in QCA and its symbol.

gate into a 2-input OR gate. Similarly by fixing one of the inputs to 0, one can turn it into a 2-input AND gate. It therefore appears that it should be possible to implement any Boolean function in the QCA architecture. However, this produces very inefficient realizations since about 33% of the gate inputs are tied to constant values. Further, the large number of gates makes circuit layout a lot harder. It is therefore important to develop designs that allow one to fully utilize the capabilities of the 3-input majority gates.

## 6.3 Comparison Function

Comparison is one of the common function used in many important applications including realization of arbitrary Boolean functions. In CMOS technology, one can easily realize an  $N$  bit comparison with a carry propagation subtractor having  $O(N)$  delay. This speed can be improved to  $O(\log N)$  by using a more complex block carry look-ahead subtractor. However, a further dramatic improvement in speed is only possible through use of nanoelectronic technologies such as the Quantum-dot cellular automata (QCA) [26]. In this chapter we describe an efficient implementation of the comparison function using QCA. The resultant architecture has optimal delay  $O(\log n)$  and a low gate complexity  $O(n)$  for an  $n$  bit comparison. The architecture can be easily pipelined to improve its throughput.

We now illustrate the use of comparison function in implementing an arbitrary Boolean function. Consider a Boolean function  $f(x_{n-1}, x_{n-2}, \dots, x_0)$  which is 1 only

### 6.3. COMPARISON FUNCTION

when the  $n$ -bit input string  $s = x_{n-1}x_{n-2} \cdots x_0$  has a value between  $X$  and  $Y$ . Let function  $C(X)$  denote a comparison function that compares value of string  $s$  with  $X$  and outputs a 1 only if value of  $s$  is greater than or equal to  $X$ . It is then obvious that  $f = C(X)\overline{C}(Y + 1)$ . Note that since an inverter as well as a 2-input AND gate can be realized in the QCA technology, so can the function  $f$  if we can design the comparison function  $C(\cdot)$  in QCA. Similarly if  $f$  is 1 anytime the input string  $s$  is either between  $X_1$  and  $Y_1$  or between  $X_2$  and  $Y_2$ , then that function is described as  $f = C(X_1)\overline{C}(Y_1 + 1) + C(X_2)\overline{C}(Y_2 + 1)$  and realized by QCA. Fig. 6.4 shows this realization.

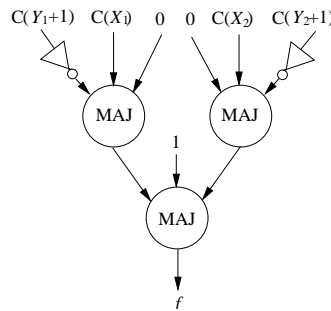


Figure 6.4: QCA realization of function  $f$  which is 1 only when its argument is between  $X_1$  and  $Y_1$  or between  $X_2$  and  $Y_2$ . Note that  $C(\cdot)$  is a comparison function.

It should thus be clear that any arbitrary Boolean function can be implemented using the strategy described here. Creating such an implementation requires one to determine contiguous groups of 1's in the truth table of the function. The end points of each group are then compared with the input variable string, and the results ANDed. Finally, outputs of all the ANDs are added together to get the function. Clearly, All the comparisons can be done concurrently and all the ANDs can be concurrent. Thus the resultant Boolean function realization may have a fairly small depth provided one can find a small depth comparison function implementation.

The comparison function  $C(B)$  that compares two  $n$  bit strings  $B = b_{n-1}b_{n-2} \cdots b_0$  and  $s = x_{n-1}x_{n-2} \cdots x_0$  can be implemented by subtracting  $B$  from  $s$  while keeping track of the carry only. Fig. 6.5 shows a QCA implementation of this strategy.

### 6.3. COMPARISON FUNCTION

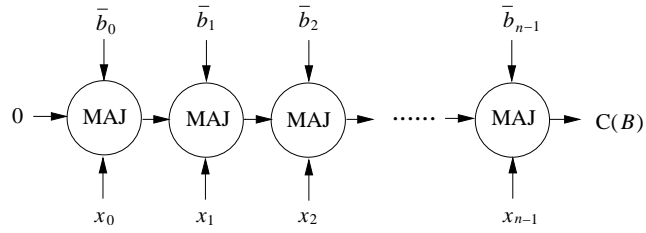


Figure 6.5: A serial realization of comparison  $C(B)$  which outputs a 1 when  $s = x_{n-1}x_{n-2}\cdots x_0 \geq b_{n-1}b_{n-2}\cdots b_0$ .

Unfortunately, the architecture in Fig. 6.5 has an  $O(n)$  delay. Minimizing this delay in QCA architectures is important because even in the combinational logic, unlike CMOS, all the gates in QCA implementations need to be clocked. Further, in applications such as the Boolean function implementation, the comparator delay can directly impact the delay of the Boolean function. In order to achieve the optimal delay, we build the comparison output recursively. Suppose operands  $X$  and  $B$  are partitioned as  $X = [X_1|X_0]$  and  $B = [B_1|B_0]$ .  $X \geq B$  is true if  $X_1 > B_1$  or if  $(X_1 = B_1)$  and simultaneously  $(X_0 \geq B_0)$ . However, computing equality of  $X_1$  and  $B_1$  requires XOR gates which are expensive in QCA technology. We therefore define intermediate logical variables  $p_i = (X_i > B_i)$  and  $q_i = (X_i \geq B_i)$ ,  $i = 0$  or  $1$ . With this, the output of comparison  $X \geq B$  is given by the Boolean expression  $p_1 + (q_1\bar{p}_1)q_0 = p_1 + q_1q_0$ . Further, using the fact that  $p_1 = 1$  implies  $q_1 = 1$ , one gets  $p_1 = p_1q_1$  and  $q_1 = p_1 + q_1$ . Thus  $p_1 + q_1q_0$  can be rewritten as  $p_1q_1 + p_1q_0 + q_1q_0$ , which is precisely the output of a three input majority gate with inputs  $p_1$ ,  $q_1$  and  $q_0$ .

Similarly, the truth value of  $X > B$  can also be computed from the intermediate logical variables  $p_i$  and  $q_i$ . In particular,  $X > B$  if either  $X_1 > B_1$  or  $A_1 \geq B_1$  and simultaneously  $X_0 > B_0$ . This can be expressed by the Boolean expression  $p_1 + q_1p_0$ . Once more using the fact that  $p_1 = p_1q_1$  and  $q_1 = p_1 + q_1$ , the expression for  $X > B$  can be rewritten as  $p_1q_1 + (p_1 + q_1)p_0$ . But this says that  $X > B$  can be computed with a 3-input majority gate with inputs  $p_1$ ,  $p_0$  and  $q_1$ .

### 6.3. COMPARISON FUNCTION

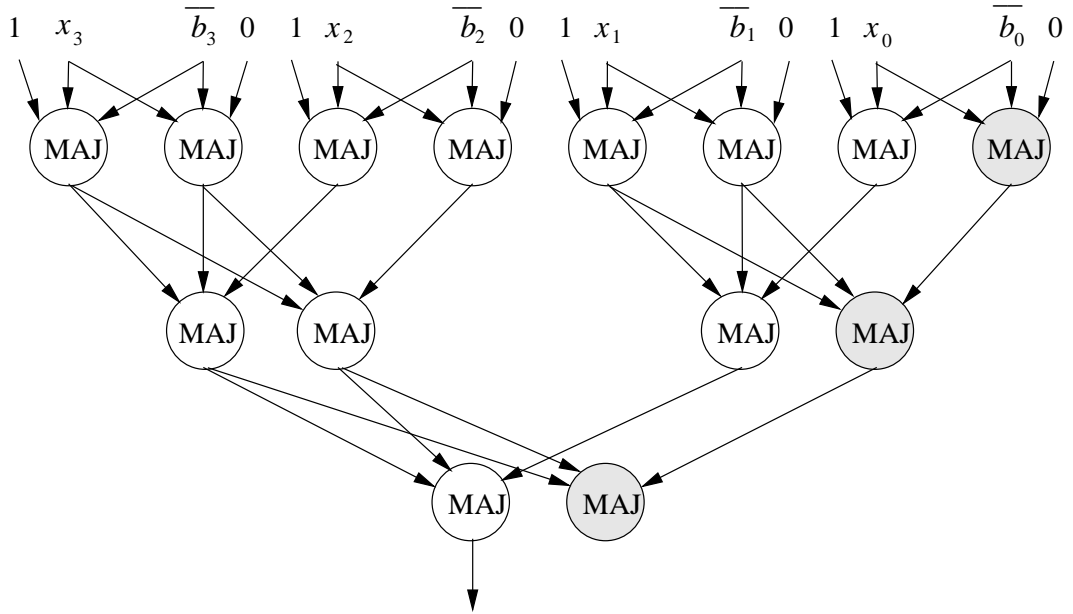


Figure 6.6: A 4-bit comparator architecture which produces a 1 when  $x_3x_2x_1x_0 \geq b_3b_2b_1b_0$ . (The majority gates in gray need not be implemented.)

To compute  $p_i$  and  $q_i$ , the operands  $X_i$  and  $B_i$  can be partitioned recursively and the same procedure used. This can be continued till each partition is a single bit. Fig. 6.6 shows an implementation of a 4 bit comparator in QCA technology.

The number of majority gates used in this realization is  $4N - 3 - \lceil \log_2 N \rceil$ . The tree-like structure allows for easy separation of clocking zones in QCA. Further, if one of the strings, say  $B$ , is constant, then all the  $\bar{b}_i$  are known in the design and one can apply 1's and 0's, as appropriate at the inputs that expect  $\bar{b}_i$ 's.

A two bit comparator architecture layout is illustrated in Fig. 6.7. It should be noted that for  $n$  as small as 2, the proposed strategy actually will have the same delay but more gates than the sequential strategy of Fig. 6.5. One may note that the advantage of the new architecture in Fig. 6.6 can be realized only for larger values of  $n$ . It reduces the comparator delay from  $n$  to  $\lceil \log_2 n \rceil + 1$ , while increasing the number of majority gates from  $n$  to  $4n - \lceil \log_2 n \rceil - 3$ .

However, as can be seen from Fig. 6.6, the tree architecture has wire crossings

### 6.3. COMPARISON FUNCTION

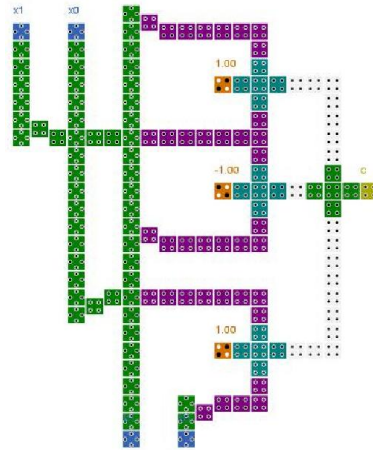


Figure 6.7: Layout of a 2 bit comparator in QCA technology.

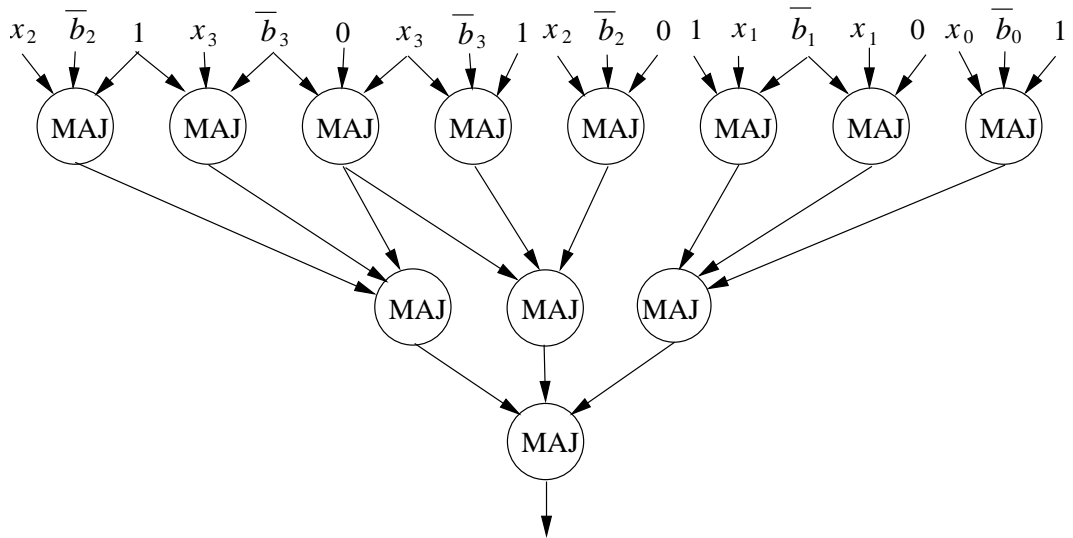


Figure 6.8: The same comparator as in Fig. 6.6 after elimination of wire crossings.

at every level of the tree. QCA being a planer architecture, minimizing these wire crossings is important. To achieve this objective, we duplicate certain majority gates as shown in Fig. 6.8.

#### 6.4. A MAJORITY GATE ADDER

Note that this conversion does not affect the architecture but it completely eliminates the crossing of wires from all tree levels except the first. The resultant comparators of 2, 4, 8 and 16 bits use only 4, 12, 33 and 89 majority gates respectively.

### 6.4 A Majority Gate Adder

Derived from the design procedure of LDTA in Chapter 3, an adder made of majority gates only can be developed. The resultant circuit has a depth of  $O(\log N)$  and a complexity of  $O(N \log N)$ .

Note that the LDTA adder uses only three input majority gates in all levels except the top level. The threshold gates in the top level result from steps 1 and 2 of the procedure given. We now show that these gates can also be decomposed in 3-input majority gates.

Step 1 of the procedure to compute  $c_i$ ,  $0 \leq i < M/2$  can be modified to compute each of these  $c_i$  from  $c_{i-1}$ . Similarly, in step 2, one computes  $G$  and  $T$  over the range  $[jM/2 + k : jM/2]$ ,  $0 \leq k < M/2$ ,  $1 \leq j < 2N/M$  directly from operand bits over these ranges. Instead, these may be obtained by combining  $G$  and  $T$  over the ranges  $[jM/2 + k - 1 : jM/2]$  and  $[jM/2 + k, jM/2 + k]$ . Note that  $G$  and  $T$  over the range  $[jM/2 + k, jM/2 + k]$  are simply  $a_{jM/2+k}b_{jM/2+k}$  and  $a_{jM/2+k} + b_{jM/2+k}$  respectively. Thus, using (3.22) one gets,

$$\begin{aligned} G_{jM/2+k:jM/2} &= (a_{jM/2+k}b_{jM/2+k}) + (a_{jM/2+k}b_{jM/2+k})G_{jM/2+k-1:jM/2} \\ &= \mathbf{MAJ}(a_{jM/2+k}, b_{jM/2+k}, G_{jM/2+k-1:jM/2}). \end{aligned}$$

The last step in this equation is obtained from Theorem 1(P6). Similarly, using (3.24),

$$\begin{aligned} T_{jM/2+k:jM/2} &= (a_{jM/2+k}b_{jM/2+k}) + (a_{jM/2+k}b_{jM/2+k})T_{jM/2+k-1:jM/2} \\ &= \mathbf{MAJ}(a_{jM/2+k}, b_{jM/2+k}, T_{jM/2+k-1:jM/2}). \end{aligned}$$

Steps 1 and 2 of the LDTA design can now be modified to read:

## 6.5. CONCLUSION

1. Obtain  $c_i = \mathbf{MAJ}(a_i, b_i, c_{i-1})$ ,  $0 \leq i < M/2$ .
2. For  $1 \leq j < 2N/M$ , obtain  $G_{jM/2:jM/2} = \mathbf{MAJ}(a_{jM/2}, b_{jM/2}, 0)$  and  $T_{jM/2:jM/2} = \mathbf{MAJ}(a_{jM/2}, b_{jM/2}, 1)$ . Then, for each  $1 \leq j < 2N/M$ , obtain for  $0 \leq k < M/2$ ,  $G_{jM/2+k:jM/2} = \mathbf{MAJ}(a_{jM/2}, b_{jM/2}, G_{jM/2+k-1:jM/2})$  and  $T_{jM/2+k:jM/2} = \mathbf{MAJ}(a_{jM/2}, b_{jM/2}, T_{jM/2+k-1:jM/2})$ .

With this transformation, the repetition of inputs in the top level threshold gates that was present in the earlier LDTA is eliminated. This results in a smaller number of total inputs. This result is summarized by the following theorem.

**Theorem 23** *One can compute all the carries of an  $N$  bit adder using only 3-input majority gates. This circuit has a depth of  $\log_2(2N/M) + (M/2)$  and a hardware complexity of  $3N \log_2(4N/M) - (4N/M) + 2$  inputs.*

Fig.6.9 shows the carry computation circuit of an 8-bit LDTA using only majority gates. Note that the 3-input majority gates with one input 0 or 1 are really two input ANDs and ORs respectively.

Since the carry computational circuit of this modified LDTA uses only 3-input majority gates, the fan-in of the circuit is no longer a limitation as long as it is 3 or larger. Thus in this design,  $M$  merely serves as a parameter that allows one to trade off the depth with the hardware complexity in implementations using the same 3-input majority gates. Smaller  $M$  is associated with a smaller depth but a larger complexity and larger  $M$  implies a smaller complexity but a larger depth. In particular, when  $M = 2N$ , the adder degenerates to a CPA.

## 6.5 Conclusion

This chapter provides new QCA architectures for the comparison function. This architecture reduces the delay of an  $n$  bit comparison to  $O(\log n)$  while maintaining the gate complexity to  $O(n)$ . Using this comparison function, one may be able to obtain better (low depth) implementations of some Boolean functions.



## 6.5. CONCLUSION

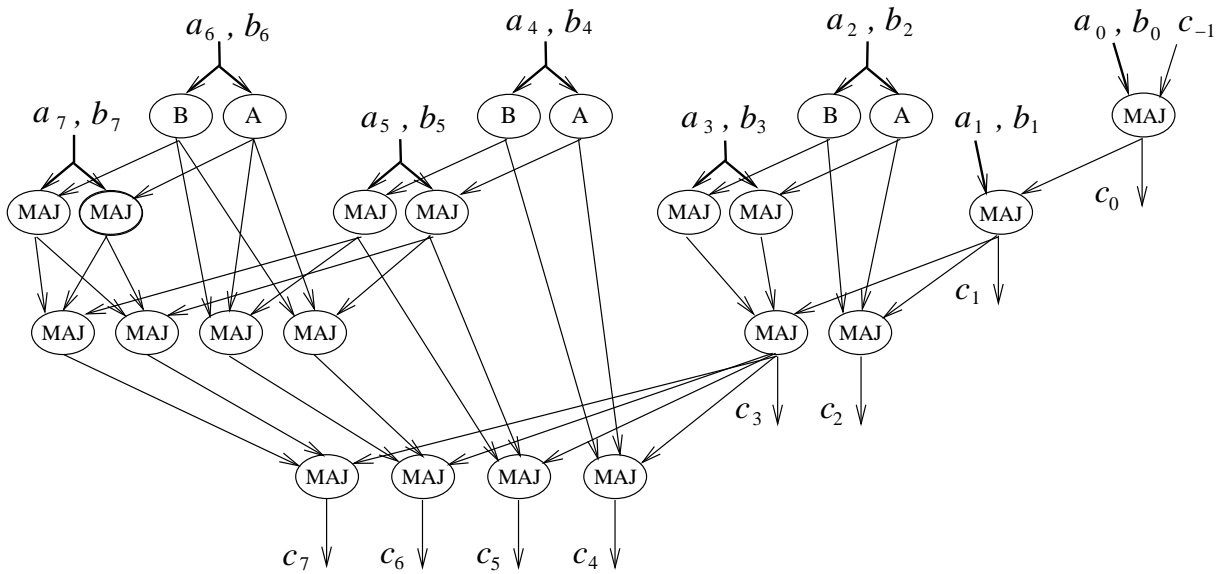


Figure 6.9: Calculating carries in an 8-bit LDMA using only 3 input majority gates.

Also this chapter derives from the design procedure of LDMA in Chapter 3 to develop an architecture of a low depth majority threshold adder. The architecture has a delay of  $O(\log N)$  and a complexity of  $O(N \log N)$ . The LDMA architecture has similar delay and complexity as the LDMA. Its two major advantages over LDMA are that it only uses majority gates (with fan-in 3 or 5) and it is able to trade off the delay for the complexity by varying the parameter  $M$  in the design. For example, in a 64 bit LDMA, by choosing  $M = 8$  rather than  $M = 4$ , one can decrease the gate count from 448 to 384 and the input complexity from 1410 to 1240 while increasing the adder delay from 8 to 9. When  $M = 2N$ , LDMA degenerates into a CPA.

# Chapter 7

## Conclusions

This dissertation has focused on computer architecture design using nanotechnology. For reliability purpose, all digital circuits that are developed in this work compose of threshold gates with bounded fan-in. We have provided general decomposition strategies that decompose any threshold gate into a network of threshold gates with bounded fan-in. We have also developed design algorithms for specific arithmetic blocks such as adders with interesting features of low depth, low complexity and flexibility to trade-off these parameters. Different implementation styles are also discussed in this dissertation. We have developed a novel systolic implementation for threshold function decomposition and combinational logic implementations that use only majority gates as demanded by the QCA nanotechnology.

Our general decomposition of any threshold function is an extension of the work of [44]. Instead of using a binary tree decomposition structure as described in [44], we generalize the structure to a  $k$ -ary tree. Though we use the same definitions and the critical concepts such as the error, the critical error, fragments, recombiners, our work extends the the hardware reduction Theorems to the  $k$ -ary tree structure. An application of these new concepts to the comparison function shows that the new architecture based on  $k$ -ary tree decomposes the function into a network of a  $O(\log N)$  depth and a  $O(N)$  gate count and interconnect complexity. The comparison also shows that this generalization of decomposition scheme provides a flexibility of design in the manner of choosing the suitable  $k$  according to the fan-in bound to

obtain a circuit with optimum depth and hardware complexity.

This work has discussed designs for the adder extensively. Adder is a key arithmetic unit in any computer architecture. We have developed a general design strategy for adders with the introduction of a new logic primitive that eliminates the complexity of implementing XOR functions with threshold gates. The strategy can be used to obtain adder architectures with low depth, low hardware complexity, or a trade-off between the two using different ways of combining the new primitives. We have illustrated the application of the strategy by developing low depth adders (LDTA) and low complexity adders (LCTA). We also have obtained an enhanced low depth adder (ELDTA), that simultaneously minimizes the complexity and the delay. We have compared ELDTA with the two common conventional adders, the carry propagation adder (CPA) and the group carry look-ahead adder (GCLA). As expected, this comparison with GCLA, also implemented with threshold gates, shows that the complexity and delay of ELDTA are both lower by about 40% as compared to that of GCLA. The CPA complexity is logarithmically lower than that of the ELDTA ( $O(N)$  versus  $O(N \log(N))$  where  $N$  denotes the adder length). However, as far as the delay is concerned, ELDTA is far superior ( $O(\log N)$  as against  $O(N)$ ) to CPA.

A systolic implementation scheme of the recombiners of a decomposition circuits is developed for applications requiring very low hardware complexity. Unfortunately the speed is sacrificed due to the serial output. It needs to be pointed out that there are no results in the current literature where sequential logic is implemented using nanoelectronic devices. This is because of the loops in sequential logic which run counter to the four phase clocking scheme mandated in all the nanoelectronic implementation of digital gates. In order to successfully create the systolic design, we had to modify the clocking to a new six phase scheme. In the application to decomposition, We use a novel recombiner scheme combines errors from two fragments at the first systolic stage and add errors from one more fragment in each of the following systolic stage. Both the hardware complexity and the time complexity of this architecture are directly related to the critical error of the threshold function and the bound of the fragments. Thus the systolic implementation scheme is best suitable

## 7.1. FUTURE WORK

for threshold functions with smaller weights. A generalized majority function and a pattern matching function is shown with the systolic implementation. With certain increase in the time complexity, the gate counts are tremendously saved using the systolic system.

Finally, this dissertation has also developed better designs for some applications using only three and five input majority gates. Such designs are immediately applicable to QCA technology.

The design and implementation strategies developed in this dissertation are not limited to the applications that are provided but to a boarder general digital world. To summarize, this work develops reliable design schemes suitable for all digital circuits implemented in nanotechnology.

## 7.1 Future Work

With the design and implementation schemes that have been developed in this work, we believe that it is possible to develop flexible design algorithms and implementation styles following the methodology. Adder architecture design initiates our interest in developing design algorithms for addition related functions such as multiplication. One can expect to extend the design tool developed in this work to those that are suitable for more complex arithmetic functions. One more interesting future work is the design and implementation of sequential logic using nanotechnology. In this work a new clocking scheme is introduced that breaks the original design paradigm of nanotechnology circuits when data flow only goes in one direction. This could prove to be a new direction to explore sequential logic implementation with nanotechnology.

# Bibliography

- [1] V. Beiu, J. Peperstraete, J. Vandewalle, and R. Lauwereins, “Efficient decomposition of comparison and its applications,” in *In M. Verleysen (ed.): European Symp. Artif. Neural Networks ESANN’93*, (Dfacto, Brussels), pp. 45–50, April 1993.
- [2] V. Beiu, J. Peperstraete, J. Vandewalle, and R. Lauwereins, “Overview of some efficient threshold gate decomposition algorithms,” in *Proc. of 9th Intl. Conf. Control Systems and Comp. Sci. CSCS’93*, (Bucharest, Romania), pp. 458–469, May 1993.
- [3] “The international technology roadmap for semiconductors: Emerging research devices.” <http://www.itrs.net/>, 2005.
- [4] C. Pacha and K. Goser, “Design of arithmetic circuits using resonant tunneling diodes and threshold logic,” in *Proc. of the 2nd Workshop on Innovative Circuits and Systems for Nanoelectronics*, (Delft, NL), pp. 83–93, Sep. 1997.
- [5] C. Lageweg, S. Cotofana, and S. Vassiliadis, “A linear threshold gate implementation in single electron technology,” in *Proc. IEEE-CS Annual Workshop on VLSI*, (Orlando, FL), pp. 93–98, Apr. 2001.
- [6] A. Schmid and Y. Leblebici, “Robust circuit and system design methodologies for nanometer-scale devices and single-electron transistors,” *IEEE Trans. on VLSI Systems*, vol. 12, pp. 1156–1166, Nov. 2004.

## BIBLIOGRAPHY

- [7] I. Amlani, A. O. Orlov, G. Toth, G. H. Bernstein, C. S. Lent, and G. L. Snider, “Digital logic gate using quantum-dot cellular automata,” *Science*, vol. 284, pp. 289–291, April 1999.
- [8] N. J. A. Sloane and S. Plouffe, *The Encyclopedia of Integer Sequences*. Academic Press, 1995.
- [9] W. Prost, U. Auer, F.-J. Tegude, C. Pacha, K. F. Gosser, G. Janssen, and T. van der Roer, “Manufacturability and robust design of nanoelectronic logic circuits based on resonant tunnelling diodes,” *Int. J. Circ. Theor. Appl.*, vol. 28, pp. 537–552, 2000.
- [10] J. G. Guimaraes, H. C. Carmo, and J. C. da Costa, “Basic subcircuits with single-electron tunneling devices,” in *Proc. of 17th Symp. on Technology and Devices*, 2002.
- [11] C. Lageweg, S. Cotofana, and S. Vassiliadis, “Evaluation methodology for single electron encoded threshold logic gates,” in *Proc. Int. Conf. on Very Large Scale Systems Systems on Chip*, pp. 258–262, Dec. 2003.
- [12] Y. Sun and M. D. Wagh, “A fan-in bounded low delay adder for nanotechnology,” in *Proc. of 2010 NanoTech Conf., vol. 2*, (Anaheim, CA), pp. 83–86, July 2010.
- [13] M. D. Wagh, Y. Sun, and V. Annampedu, “Implementation of comparison function using quantum-dot cellular automata,” in *Proc. of 2008 NanoTech Conf., vol. 3*, vol. 3, (Boston, MA), pp. 76–79, June 2008.
- [14] K. Gosser and C. Pacha, “System and circuit aspects of nanoelectronics,” in *24th European Solid-State Circuits Conf.*, (The Hague, NL), pp. 18–29, Sep. 1998.
- [15] D. Goldhaber-Gordon, M. S. Montemerlo, J. C. Love, G. J. Opteck, and J. C. Ellenbogen, “Overview of nanoelectronic devices,” tech. rep., MITRE Corporation, McLean, Virginia, Mar. 1997.

## BIBLIOGRAPHY

- [16] “Definition of software interfaces.” ANSWERS Tech Report (Autonomous Nanoelectronic Systems With Extended Replication and Signalling), Jan. 1999.
- [17] A. Sellai, H. Al-Hadhrami, S. Al-Harthy, and M. Henini, “Resonant tunneling diode circuits using PSPICE,” *Microelectronics Journal*, vol. 34, no. 5–8, pp. 741–745, 2003.
- [18] Z. Yan and M. Deen, “New RTD large-signal DC model suitable for PSPICE,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, pp. 167–171, Feb. 1995.
- [19] C. Moffat, “The resonant tunnelling transistor,” tech. rep., Image Processing Group, University of College London, July 1996.
- [20] K. Maezawa and T. Mizutani, “A new resonant tunneling logic gate employing monostable-bistable transition,” *Japan J. Appl. Phys.*, vol. 32, pp. L42–L44, 1993.
- [21] K. J. Chen, K. Maezawa, and M. Yamamoto, “InP-based high performance monostable-bistable transition logic elements (MOBILE’s) using integrated multiple-input resonant-tunneling devices,” *IEEE Elec. Dev. Lett.*, vol. 17, pp. 127–129, Mar. 1996.
- [22] T. Akeyoshi, K. Maezawa, and T. Mizutani, “Weighted sum threshold logic operation of MOBILE (monostable-bistable transition logic element) using resonant-tunneling transistors,” *IEEE Elec. Dev. Lett.*, vol. 14, pp. 475–477, Oct. 1993.
- [23] C. Pacha, K. Goser, A. Brennemann, and W. Prost, “A threshold logic full adder based on resonant tunneling transistors,” *24th European Solid-State Circuits Conf.*, pp. 428–431, 1998.
- [24] W. Prost, U. Auer, J. Degenhardt, A. Brennemann, C. Pacha, K. F. Goser, and F.-J. Tegude, “A depth-2 full-adder circuit using the InP RTD/HFET

## BIBLIOGRAPHY

- MOBILE,” in *Proc. Indium Phosphide and Related Materials Conference (IPRM'01)*, pp. 5045–5046, May 2001.
- [25] W. Prost, U. Auer, F. J. Tegude, C. Pacha, K. F. Gosser, R. Duschl, K. Eberl, and O. G. Schmidt, “Tunnelling diode technology,” in *31st IEEE Intl. Symp. on Multiple Valued Logic*, pp. 49–58, May 2001.
- [26] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, “Quantum cellular automata,” *Nanotechnology*, vol. 4, pp. 49–57, Jan. 1993.
- [27] G. L. Snider, A. O. Orlov, I. Amlani, G. H. Bernstein, C. S. Lent, J. L. Merz, and W. Porod, “Quantum-dot cellular automata: Line and majority logic gate,” *Jpn. J. Appl. Phys.*, vol. 38, pp. 7227–7229, 1999.
- [28] A. O. Orlov, I. Amlani, G. Toth, C. S. Lent, G. H. Bernstein, and G. L. Snider, “Experimental demonstration of a binary wire for quantum-dot cellular automata,” *Applied Physics Letters*, vol. 74, pp. 2875–2877, May 1999.
- [29] J. R. Janulis, P. D. Tougaw, S. C. Henderson, and E. W. Johnson, “Serial bit stream analysis using quantum-dot cellular automata,” in *IEEE Transactions on Nanotechnology*, vol. 3, pp. 158–164, Mar. 2004.
- [30] S. Roy and B. Saha, “Minority gate oriented logic design with quantum-dot cellular automata,” in *Cellular Automata, 7th International Conference on Cellular Automata, for Research and Industry, ACRI 2006, Perpignan, France, September 20-23, 2006, Proceedings*, vol. 4173 of *Lecture Notes in Computer Science*, pp. 646–656, 2006.
- [31] S. Muroga, *Threshold logic and its applications*. New York: Wiley-Interscience, 1971.
- [32] V. Beiu, J. M. Quintana, and M. J. Avedillo, “VLSI implementations of threshold logic - a comprehensive survey,” *IEEE trans. Neural Networks*, vol. 14, pp. 1217–1243, Sep. 2003.



## BIBLIOGRAPHY

- [33] “Definition of software interfaces.” ANSWERS (Autonomous Nanoelectronic Systems With Extended Replication and Signalling) Project Report, Jan. 1999.
- [34] P. Gupta and N. K. Jha, “An algorithm for nano-pipelining of RTD-based circuits and architectures,” *IEEE Trans. on Nanotechnology*, vol. 4, pp. 159–167, Mar 2005.
- [35] M. Goldmann and M. Karpinski, “Simulating threshold circuits by majority circuits,” *SIAM J. Computing*, vol. 27, pp. 230–246, Feb 1998.
- [36] E. Allender, “Circuit complexity before the dawn of the new millennium,” in *Lecture Notes in Computer Science*, vol. 1180, pp. 1–18, Springer-Verlag, 1996.
- [37] A. Maciel and D. Thérien, “Threshold circuits of small majority-depth,” *Info. and Computation*, vol. 146, pp. 55–83, Oct 1998.
- [38] K.-Y. Siu and J. Bruck, “On the power of threshold circuits with small weights,” *SIAM J. on Disc. Math.*, vol. 4, pp. 423–435, Aug 1991.
- [39] N. Alon and J. Bruck, “Explicit construction of depth-2 majority circuits for comparison and addition,” *SIAM J. Discrete Math*, vol. 7, no. 1, pp. 1–8, 1994.
- [40] S. Cotofana and S. Vassiliadis, “Signed digit addition and related operations with threshold logic,” *IEEE Trans. on Computers*, vol. 49, no. 3, pp. 193–207, 2000.
- [41] R. Katz, *Contemporary Logic Design*. Benjamin/Cummings, 1994.
- [42] V. Annampedu and M. D. Wagh, “Approximate pattern matching in nanotechnology,” in *Proc. of Nanotech 2006*, vol. 3, (Boston, MA), pp. 316–319, May 7–11 2006.
- [43] G. S. Glinski and C. K. Yue, “Decomposition of n-variable threshold function into p-variable threshold functions, where  $p < n$ ,” Tech. Rep. 63-10, Dept. of EE, Univ. of Ottawa, Canada, June 1963.

## BIBLIOGRAPHY

- [44] V. Annampedu and M. D. Wagh, “Reconfigurable approximate pattern matching architectures for nanotechnology,” *Microelectronics*, vol. 38, pp. 430–438, 2007.
- [45] V. Bohossian, M. Riedel, and J. Bruck, “Trading weight size for circuit depth: An  $\widehat{LT}_2$  circuit for comparison,” Tech. Rep. Paradise, ETR028, California Institute of Technology, Nov. 1998.
- [46] P. A. Jackson, C. P. Chan, J. E. Scalera, C. M. Rader, and M. M. Vai, “A systolic fft architecture for real time fpga systems,” tech. rep., MIT Lincoln Laboratory, Lexington, MA, 2005.
- [47] G. Sal and M. Ari, “Fpga-based customizable systolic architecture for image processing applications,” in *Reconfigurable Computing and FPGAs, 2005. International Conference on*, pp. 8 pp. –3, Feb. 2005.
- [48] H. T. Kung, “Why systolic architectures?,” *Computer Magazine*, vol. 15, pp. 37–46, Jan. 1982.
- [49] K. Walus, G. A. Jullien, and V. S. Dimitrov, “Computer arithmetic structures for quantum cellular automata,” in *Proc. 37th Asilomar Conf. Signals, Systems and Computers*, (Pacific Grove, CA), pp. 9–12, Nov. 9–12 2003.
- [50] I. Hanninen and J. Takala, “Binary multipliers on quantum-dot cellular automata,” *Facta Universitatis Ser.: Elec. Energ.*, vol. 20, pp. 541–560, December 2007.
- [51] H. Cho and E. E. Swartzlander, “Adder designs and analyses for quantum-dot cellular automata,” *IEEE Trans. Nanotechnology*, vol. 6, pp. 374–383, May 2007.

# Vita

Yichun Sun was born in Shanghai, China on June 24th, 1983. She received her Bachelor of Engineering degree in Electrical Engineering from Shanghai Jiaotong University, in June 2005. She obtained her Master of Science degree in Electrical Engineering from Lehigh University, in May 2007.

Since August 2005, Yichun has been studying Electrical Engineering at Lehigh University and since 2006, she has worked as research assistant under the supervision of Dr. Meghanad. D. Wagh. She has worked on computer architectures and digital design with nanotechnology. From August 2006 to May 2008, she has also worked as teaching assistant in the Electrical Engineering department at Lehigh University. She has also worked as internship at LSI Corporation since May 2011 on IC design an verification.

She has 2 conference papers and presented them at Nanotech conferences.