5-1-2018

DESIGN AND TEST OF DIGITAL CIRCUITS AND SYSTEMS USING CMOS AND EMERGING RESISTIVE DEVICES

Seyed Nima Mozaffari Mojaveri
Southern Illinois University Carbondale, nima.mozaffari@siu.edu

DESIGN AND TEST OF DIGITAL CIRCUITS AND SYSTEMS USING CMOS AND EMERGING RESISTIVE DEVICES

by

Seyed Nima Mozaffari Mojaveri

B.S. in Electrical and Computer Engineering, University of Mazandaran, 2007
M.S. in Electrical and Computer Engineering, University of Tehran, 2010

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

Department of Electrical and Computer Engineering
in the Graduate School
Southern Illinois University Carbondale
May 2018

DISSERTATION APPROVAL

DESIGN AND TEST OF DIGITAL CIRCUITS AND SYSTEMS USING CMOS AND EMERGING RESISTIVE DEVICES

By

Seyed Nima Mozaffari Mojaveri

A Dissertation Submitted in Partial

Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in the field of Electrical and Computer Engineering

Approved by:

Dr. Spyros Tragoudas, Chair

Dr. Iraklis Anagnostopoulos

Dr. Themistoklis Haniotakis

Dr. Haibo Wang

Dr. Shahram Rahimi

Graduate School
Southern Illinois University Carbondale
March 8, 2018

AN ABSTRACT OF THE DISSERTATION OF

Seyed Nima Mozaffari Mojaveri, for the Doctor of Philosophy degree in ELECTRICAL AND COMPUTER ENGINEERING, presented on March 8, 2018, at Southern Illinois University Carbondale.

TITLE: DESIGN AND TEST OF DIGITAL CIRCUITS AND SYSTEMS USING CMOS AND EMERGING RESISTIVE DEVICES

MAJOR PROFESSOR: Dr. Spyros Tragoudas

The memristor is an emerging nano-device. Low power operation, high density, scalability, non-volatility, and compatibility with CMOS Technology have made it a promising technology for memory, Boolean implementation, computing, and logic systems. This dissertation focuses on testing and design of such applications. In particular, we investigate on testing of memristor-based memories, design of memristive implementation of Boolean functions, and reliability and design of neuromorphic computing such as neural network. In addition, we show how to modify threshold logic gates to implement more functions.

Although memristor is a promising emerging technology but is prone to defects due to uncertainties in nanoscale fabrication. Fast March tests are proposed in Chapter 2 that benefit from fast write operations. The test application time is reduced significantly while simultaneously reducing the average test energy per cell. Experimental evaluation in 45 nm technology show a speed-up of approximately 70% with a decrease in energy by approximately 40%. DfT schemes are proposed to implement the new test methods.

In Chapter 3, an Integer Linear Programming based framework to identify current-mode threshold logic functions is presented. It is shown that threshold logic functions can be implemented in CMOS-based current mode logic with reduced transistor count when the input weights are not restricted to be integers. Experimental results show that many more functions can be implemented with predetermined hardware overhead, and the hardware requirement of a large percentage of existing threshold functions is reduced when comparing to the traditional CMOS-based threshold logic implementation.

In Chapter 4, a new method to implement threshold logic functions using memristors is presented. This method benefits from the high range of memristor's resistivity which is used to define different weight values, and reduces significantly the transistor count. The proposed approach implements many more functions as threshold logic gates when comparing to existing implementations. Experimental results in 45 nm technology show that the proposed memristive approach implements threshold logic gates with less area and power consumption.

Finally, Chapter 5 focuses on current-based designs for neural networks. CMOS aging impacts the total synaptic current and this impacts the accuracy. Chapter 5 introduces an enhanced memristive crossbar array (MCA) based analog neural network architecture to improve reliability due to the aging effect. A built-in current-based calibration circuit is introduced to restore the total synaptic current. The calibration circuit is a current sensor that receives the ideal reference current for non-aged column and restores the reduced sensed current at each column to the ideal value. Experimental results show that the proposed approach restores the currents with less than 1% precision, and the area overhead is negligible.

DEDICATION

To my wife, Bahar Oladazimi Ghadikolaei, and my daughter Hanna Mozaffari, and my parents Seyed Ahmad Mozaffari Mojaveri, Mehrangiz Khosravi Koochaksaraei, and my sisters, Seyedeh Elham Mozaffari Mojaveri, Seyedeh Armaghan Mozaffari Mojaveri.

ACKNOWLEDGMENTS

My most sincere thanks go to my advisor, Dr. Spyros Tragoudas for his invaluable guidance, motivation and direction in my research.

I would like to take an opportunity to thank Dr. Themistoklis Haniotakis, Dr. Haibo Wang, Dr. Iraklis Anagnostopoulos and Dr. Shahram Rahimi for taking time from their busy schedule to serve on my committee and their guidance in writing my dissertation document.

I would like to thank current and past members of Design Automation Lab (E105, E238) with whom I had fruitful interactions. Especial thanks to Pavan Kumar Javvaji, Krishna Prasad Gnawali, Wisam Abdulrahman Al-Jubouri, Phaninder Alladi and Puneet Ramesh Savanur for their support.

I am also thankful to my family for their unconditional support in my career. I am thankful to all my friends who have been part of my life.

PREFACE

This research has been supported in part by grants NSF IIP 1432026, and NSF IIP 1361847 from the NSF I/UCRC for Embedded Systems at SIUC. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

The memristor is an emerging nano-device. Low power operation, high density, scalability, non-volatility, and compatibility with CMOS Technology have made it a promising technology for memory, Boolean implementation, computing, and logic systems. This dissertation focuses on testing and design of such applications. In particular, we investigate on testing of memristor-based memories, design of memristive implementation of Boolean functions, and reliability and design of neuromorphic computing such as neural network. In addition, we will show how to modify threshold logic gates to implement more functions.

Nano scale devices are prone to defects due to design marginalities, imperfection in fabrication process, and process variation [24]. Some defects impact the logical behavior of resistive memory cells. Others impact the temporal behavior. Chapter 2 focuses on testing of memristor-based memory, and introduces a novel approach to reduce the test application time using fast write operations. The proposed method benefits from the behavior of memristor device which is nonlinear and asymmetric. The approach is taking into consideration the random memristive behavior and sneak-paths in crossbar memory. A new Design for Testability (DfT) mechanism is required to implement the proposed fast write operation. The experimental results on DfT implementation in 45nm technology will be reported in Chapter 2 to evaluate the effectiveness of the proposed approach.

Threshold Logic Gate (TG) is a promising candidate for the future digital circuits.

A Boolean function that can be implemented as a single TG is called Threshold Logic Function (TF). In a TF, there is an integer weight for each input. An input pattern evaluates the function to logic one only when the sum of the active weights is greater than (or equal) to a predetermined integer weight value called the threshold weight. Otherwise, it evaluates the function to logic 0.

A small fraction of binary functions are TFs, and this limits the impact of TGs in digital circuit synthesis. Thus, our focus shifts on identifying more functions as TFs. Chapter 3 benefits from the higher order definition of TF and shows that a weight in a TF can be activated by a group of active inputs.

Moreover, TF implementations consist of three components: two differential networks (input networks) and a sensor (sense amplifier). The power dissipation of a TG depends primarily on two factors: the transistor count of the input networks which is the total number of unit size transistors that implements weights and the sensor size which is proportional to the transistor count of input networks. Therefore, Chapter 3 proposes a new method to reduce the transistor count of the input networks by introducing non-integer weights.

The transistor count reduces further when weights are implemented by resistive devices. This has been investigated in Chapter 4. Chapter 4 proposes a new method to implement efficiently the higher order weight components using non-volatile resistive memories (memristors). The resistance value of a memristor is called its memristance, and the range of memristance is used to define different weight values. This method of weight implementation reduces significantly the transistor count of the input networks. Chapter 4 is an extended version of the Chapter 5.

Artificial neural networks (ANNs) are machine-learning systems for pattern matching, character and speech recognition, and big data management among other applications. They consist of an input layer, an output layer and multiple hidden layers. Each layer consists of several neurons. Each neuron has multiple inputs that are typically real numbers and one output that is typically a real number. Every input signal is convolved with a predetermined weight value called the synaptic weight. A neuron calculates the weighted sum of all the convolved signals, and it is mapped to the output signal by a component called the activation function.

It is observed that analog-based ANNs may result in erroneous computations due to transistor aging. In particular, Bias Temperature Instability (BTI), and to a lesser extent, dielectric breakdown as well as Hot Carrier Injections (HCI) shift the threshold voltage of the CMOS transistor causing the reduction in the drain current. Therefore, the synaptic current reduces as CMOS component of the cell ages. This impacts the value of each convolution. It is experimentally shown in Chapter 5 that the aging impacts the computational accuracy of analog ANNs.

Chapter 5 introduces an enhanced memristive crossbar array (MCA) based ANN architecture to improve reliability due to the aging effect. The MCA is enhanced by an extra row (the calibration row) and an extra column (the spare column). A built-in current-based calibration circuit is introduced to restore the total synaptic current. The calibration circuit is a current sensor that receives the ideal reference current for non-aged column and restores the reduced sensed current at each column to the ideal value. It will be shown that the enhances MCA columns with a calibration circuit alleviates the aging effect and maintain invariant sum of synaptic currents.

CHAPTER 2

EFFICIENT TESTING OF METAL-OXIDE MEMRISTOR-BASED MEMORY

2.1 Introduction

The memristor is an emerging nano-device [1, 2]. Low power operation, high density, scalability, non-volatility, and compatibility with CMOS Technology have made it a promising technology for memory, computing, and logic systems [3-10]. Resistive memories are also appealing to 3D circuit designs.

Memristor-based memory is a hybrid memory where memory cells are implemented using memristors and access memory circuitry is implemented using CMOS transistors [11]. The performance of CMOS-based components, implemented in deep-submicron, is impacted by the variation of the process parameters which has become a serious design and test challenge [12]. Furthermore, variations in the length, area, and doping concentration of the memristors impact the write time in memory cells [19]. Memristor-based memory is also prone to short, open and bridging faults in metal lines [11].

This chapter focuses on testing memristor-based memory, and introduces a novel approach to reduce the test application time using fast write operations. March tests are presented, i. e., algorithms that consist of traversals of the whole memory array where the same read or write operations apply to each cell [13]. The proposed March tests benefit from the behavior of memristor device which is nonlinear and asymmetric. In particular, switching from the ON state to the OFF state is significantly slower than the inverse [14, 15]. Memristors exhibit a random behavior in resistivity change during read

and write operations [16, 17], and such variability must be taken into consideration when testing for manufacturing defects.

The approaches in [18-22] test memristor-based memories by reading simultaneously multiple cells. These cells are called the Region of Detection (RoD). This is done in order to reduce the impact of unwanted current paths, also called sneak paths, which may cause errors during test. However, these techniques use the imprecise linear symmetric model in [23], and also are not applicable to high density crossbars. The reduction on the number of read operations does not have a significant impact on the test application time since read operations are much faster than writes. March tests with fast write operations are presented where the effect of sneak paths is eliminated by applying appropriate voltages at the cells of the RoD. Such March tests reduce test application time and power.

This is the first time that write operations with different time duration are introduced in March tests. The non-March testing approaches in [24-27] use variable length write operations to detect faults which were undetectable by March tests but rely on the linear symmetric model and do not consider the random nature of memristors. We present March tests for such faults under the non-linear asymmetric model while taking into consideration the random memristive behavior.

In all the above cases, experimental evidence in 45nm technology shows that March tests with the new fast write operation reduce the test application time by at least 70% while reducing the average test energy per cell by at least 40%. This chapter considers blocked-based crossbar memory which is known to limit the side effect due to the sneak path [3, 28]. Such memory is implemented by a crossbar architecture which is

combination of CMOS transistors and memristors [3, 11]. A new Design for Testability (DfT) mechanism is presented in order to implement the proposed fast write operation in crossbar architecture. Its design uses ideas proposed in [19], [24] and [27], and implementation in 45nm technology shows that the area overhead of the DfT is only 8.875 μm^2 per column.

This chapter is organized as follows. Section 2.2 provides preliminaries on the metal-oxide memristor device which is used without loss of generality. It outlines recent advances on verilog-base modeling of its behavior, and existing fault models for defects in memristors. The proposed method and related March tests are introduced in Section 2.3. Experimental evaluation in 45nm technology is also presented. The DfT mechanism to implement the proposed method is presented in Section 2.4. Section 2.5 concludes the chapter.

2.2 Preliminaries

This work assumes that memristors are implemented as bipolar metal-oxide two-terminal devices [1, 3] because accurate models for its behavior have been developed [15]. However, the presented March tests apply to all types of memristors.

2.2.1 The bipolar metal-oxide memristor

This memristor is formed by a metal-oxide-metal thin film sandwiched between two electrodes [1, 3]. Bipolar metal-oxide memristors are variable resistors. Resistance switching in such memristors relates to the drift and diffusion direction of the mobile oxygen anions and oxygen vacancies created under Joule heating and electric fields [29]. One of the most common switching oxide is TiO_2 [30].

The memristor is written and read by biasing positive and negative voltages across the electrodes [21]. These voltages will be denoted by V_{write} and V_{read}, respectively. By applying a positive voltage across the device, the motion of anions and oxygen vacancies results in a nanoscale-conducting filament, and this decreases the total resistance of the device [30-33]. This is logic 1 and is referred to as state 1. On the other hand, in order to write logic 0 (also referred to as state 0), a negative voltage should apply across the memristor. In this case, oxygen vacancies and anions drift back, and this results in a partial dissolution of the filament, which increases the total resistance [30-33]. The magnitude of the voltage for write 0 (*w0*) may be different to the one for write 1 (*w1*).

The resistance value of a memristor is called its memristance, and the range of memristance is used to define different logic states. Let R_{ON} and R_{OFF} denote the minimum and maximum possible resistance value, respectively [34]. Logic 1 occurs when the memristance value is in the range [R_{ON} .. 0.4($R_{OFF} - R_{ON}$)] and logic 0 when the memristance value is in the range [0.6($R_{OFF} - R_{ON}$) .. R_{OFF}]. The pulse time should be long enough to change the resistance from one state into the boundary of the complementary state.

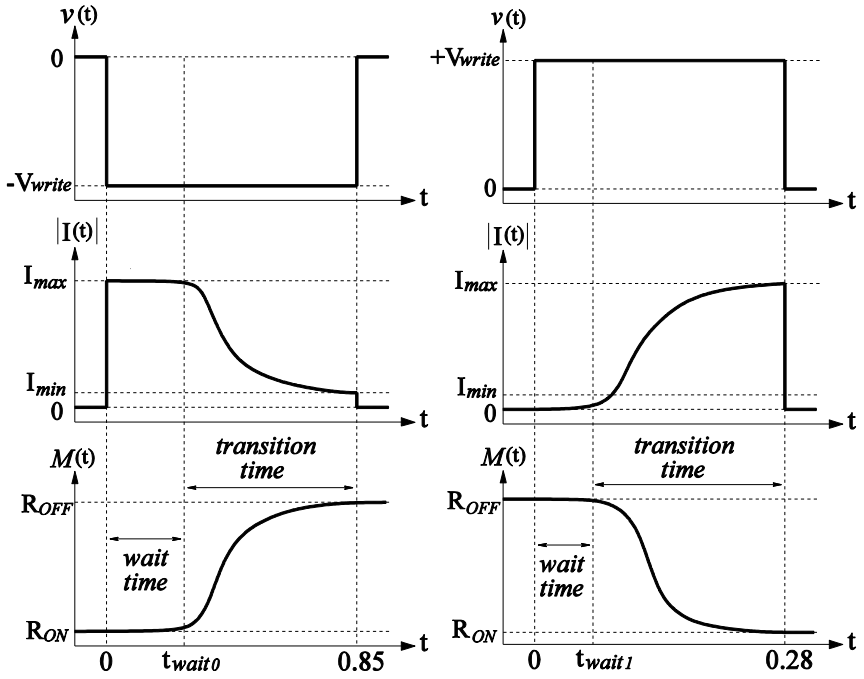
The behavior of any bipolar metal-oxide memristor device (including TiO_2-based) is nonlinear and asymmetric on transitions from one state to another [14, 15]. The ON switching (logic 0 to logic 1) is significantly faster than the OFF switching (logic 1 to logic 0). The ON switching is abrupt because the drift and diffusion of the oxygen vacancies have different directions for positive and negative voltages [14].

Figure 2.1 shows the conductivity and resistivity transition behavior of a memristor for *w1* and *w0* operations using the Voltage-controlled ThrEshold Adaptive Memristor

(VTEAM) model [15]. Manufactured bipolar metal-oxide memristors exhibit stochastic behavior during write operations [16]. In particular, when applying a pulse *w1* to the memristor, the device does not switch immediately, and waits for a time period t_{wait_1}. In [36], the difference between the application time of the write pulse in the normal mode of operation and the time that the device starts to switch is called the wait (switching) time. As shown in Figure 2.1 (b), the current does not change during the wait time t_{wait_1} and is considered to be zero [36]. On the other hand, when applying a *w0* pulse, the current is initially high and does not change during the wait time t_{wait_0}. (See also Figure 2.1 (a).) Times t_{wait_1} and t_{wait_0} may differ. Experimental results in [6, 16, 17, 30, 37, 38] on manufactured bipolar metal-oxide memristors show that the wait time varies from cycle-to-cycle. Its behavior is stochastic, and has been modeled by a Poisson distribution. This is due to the stochastic nature of the filament formation [16, 17]. It has been observed that the wait time is always less than the transition time [30, 39]. This is also shown in Figure 2.1.

The wait time strongly depends on the magnitude of the applied voltage, and decreases to at most 10% of the transition time when the when the applied voltage increases by 1-2 V [17, 36, 37, 40, 41]. For this reason, the proposed March tests use 3.3V for write operations instead of 1.2V that is used in the normal mode of operation.

A read in the normal mode of operation consists of two different stages: The first stage extracts the memristance value and determines the logic state. For this task, it is necessary to apply a stable pulse across the memristor (V_{read}). The sensed current I_{mem} through the memristor depends on the memristance value.



(a) The *w*0 operation. (b) The *w*1 operation.

Figure 2.1. Resistivity and current behavior for writes in the normal mode of operation for a bipolar metal-oxide memristor with R_{ON}=100Ω, R_{OFF}=100KΩ, $V_{write} = 1.2V$, $V_{read} = 0.8V$, using [15, 35].

The reference current in state 0 is denoted by I_{ref_0}, and the I_{ref_1}denotes the reference current in logic sate 1. Let I_{ref} be $(I_{ref_0} + I_{ref_1})/2$. The logic state is determined using a sense amplifier (implemented in CMOS technology) that compares the sensed current I_{mem} with the reference current I_{ref} of an ideal device. The duration of the first stage mostly depends on the CMOS sensing circuitry, and the duration of V_{read} must be long enough so that the sense amplifier can sense the current through the examined cell.

The first stage may change the data stored in the memory cell. However, for the data to remain intact, an inverse pulse must be applied as the second stage (also called recovery scheme) of the read operation [3, 11]. This stage is a short write operation whose duration depends mainly on the memristor. The total duration of normal read operation is

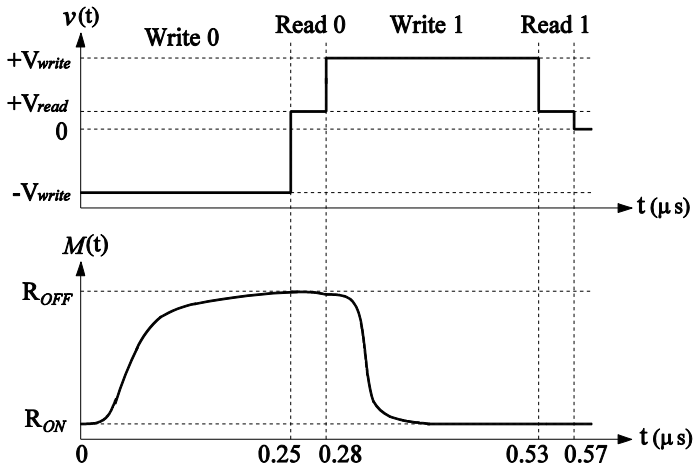


Figure 2.2. Resistivity behavior for all four operations (*w*0, *w*1, *r*0, and *r*1) during test for a bipolar metal-oxide memristor with R_{ON}=100Ω, R_{OFF}=100KΩ, $V_{write} = 3.3V$, $V_{read} = 0.8V$, using [15, 35].

sum of the duration of the two stages. The amplitude of V_{read} is 0.8V in the normal as well as the test mode. It is always lower than the write voltage to minimize the above-mentioned destructive effect of the read operation [6]. However, March tests use writes after reads, and therefore the second stage of read operation is not needed when testing.

Figure 2.2 shows the transition behavior of a memristor for all four operations (*w0*, *w1*, *r0*, and *r1*) during test using the VTEAM model in [15], with the parameter values in [35], and V_{write}=3.3V in a memristor with R_{ON}=100Ω and R_{OFF}=100KΩ. The value of 0 is stored within 255ns whereas the value of 1 is stored within 59ns. These are the minimum required access times during *w0* and *w1*. A single clock is used, and hence the access time is dominated by *w0*. A complete read operation requires 33.5ns.

2.2.2 Memristor-based crossbar memory

The crossbar memory consists of two perpendicular sets of wires. There is one memristor at the intersection of each vertical and horizontal lines which are called column and row, respectively. As shown in Figure 2.3, in order to limit the number of sneak paths,

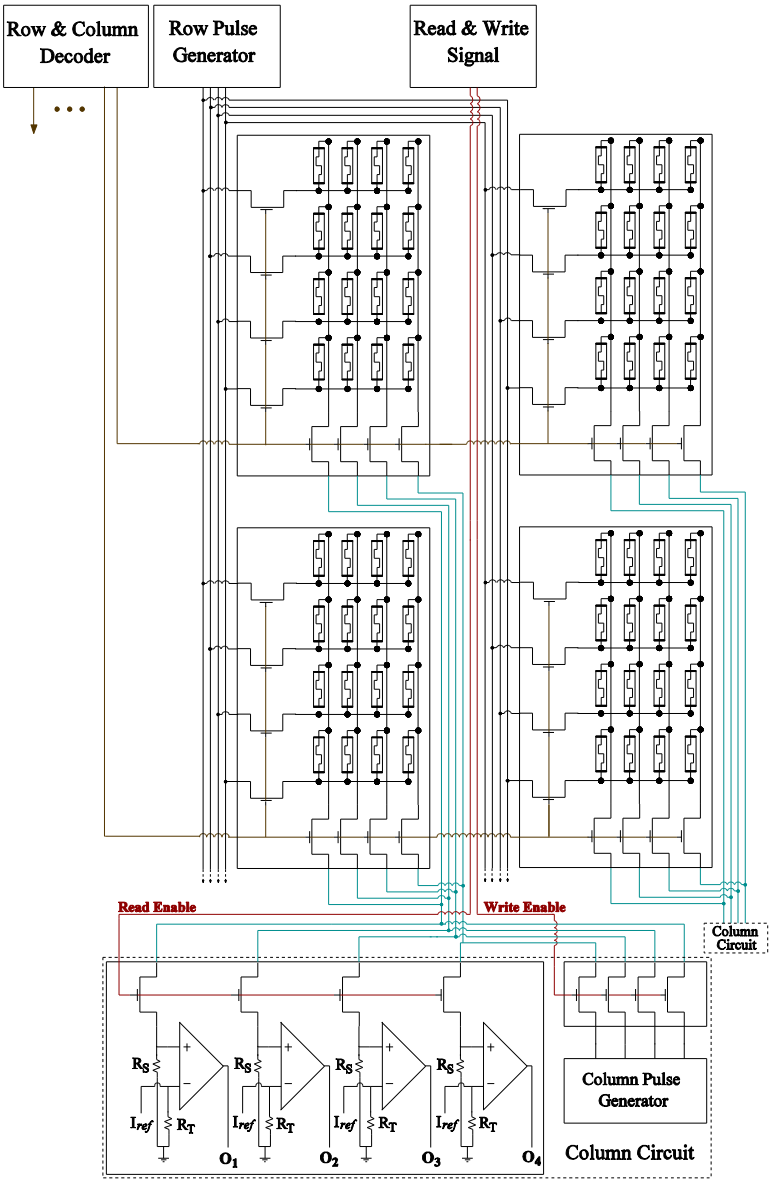


Figure 2.3. Hybrid crossbar architecture using the combination of memristor and isolating transistor [3, 28].

the larger array is divided into smaller regions, called tiles, which are isolated by transistors.

The crossbar architecture provides inherent parallelism for read and write operations on cells in different tiles for the normal mode of operation. Parallel reads and writes can be implemented using the row decoder, and the row and column pulse

generators. (See also [28].)

Read operation requires a sense amplifier which compares the output current to a predetermined threshold. Figure 2.3 illustrates how to test a memristor using a sense amplifier. The threshold is adjusted by the reference current I_{ref}. The read voltage across the memristor and R_S results into memristor current I_{mem} during the read operation, and I_{mem} is compared to I_{ref} and is compared to the current through the threshold resistor R_T [3, 25, 28]. If I_{mem} is greater than I_{ref} the output shows logic 1. Inversely, any current less than I_{ref} changes the output to logic 0.

As mentioned earlier, this chapter focuses on the testing mode of operation. In order to benefit from the crossbar architecture, appropriate enhancements must be made. Such DfT enhancements are described in Section 2.4.

2.2.3 Intrinsic and extrinsic faults

Nano scale devices are prone to defects due to design marginalities, imperfection in fabrication process, and process variation [24]. Some defects impact the logical behavior of resistive memory cells. Others impact the temporal behavior. Fault models have been defined to handle all types of defects. The following list existing fault models for memristor-based memory. Defects that impact the behavior of a memristor are modeled by faults which are called intrinsic [42]. However, there are defects that occur either between different memristors or in CMOS components that surround the memory cells. Such defects are modeled by faults which are called extrinsic [42]. A precise analysis for fault models and efficient DfT is required to keep the test time and cost low.

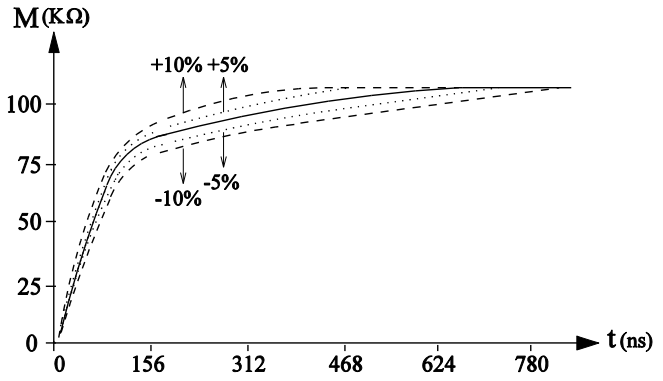
Intrinsic faults (also called self-faults) occur due to parametric variations of the memristor [18]. Variation in size (the ratio of length over area) and doping density may

impact the logic and temporal behavioral of the device. Size and doping variations have been modeled by changing the physical parameters of the device such as R_{on}, R_{off}, and the mobility of the charge [22]. For example, a variation in length causes a change in R_{on}, R_{off}. The change in resistivity, denoted by ΔR, is $\rho(\Delta L/A)$. A variation on the area impacts ΔR as $\rho[(L \cdot \Delta A)/A(A - \Delta A)]$. A variation in doping affects the mobility of the charge carriers. This changes the rate of the transitions.

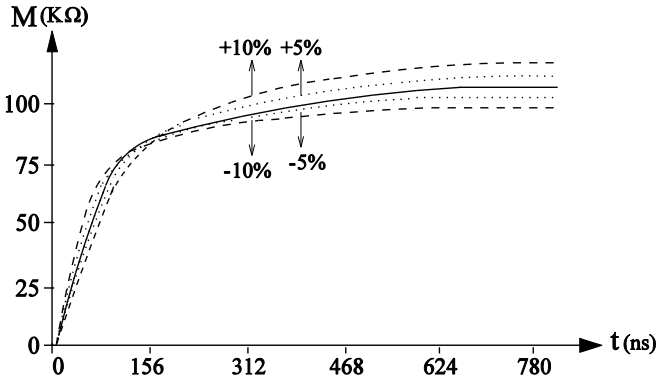
Figure 2.4 (a) shows the effect of -10%, -5%, 5% and 10% variation in doping. It can be seen that it affects the slope of the transition and therefore the transition time. The higher the doping, the faster the rate is, and the transition accelerates. Figure 2.4 (b) shows the impact of -10%, -5%, 5% and 10% size variation using the model in [15]. As the size increases, R_{OFF} increases, and therefore the transition time is increased.

The works in [11, 13, 18, 20, 24, 25, 43] consider different intrinsic fault models. They include the stuck-at fault (SAF), slow write fault (SWF), and deep state fault models. These fault models cover the range of size and doping variations as well as other types of defects such as open and shorts that may be caused by missing metal or extra metal. They are summarized in the following.

Single Stuck-at faults: When the memristor is undoped or fully doped, there is no transition in the memristance value. Undoped memristors are modeled as single stuck-at-0 (SA0) faults and fully doped memristors are modeled as single stuck-at-1 (SA1) faults [11, 19]. A single stuck-at fault corresponds to a fixed logic (0 or 1) at a single storing element in the memory. A SA0 is defined by <1/0>; where 0 is the unintended output while logic 1 is the fault-free output after a w1 operation. Similarly, a SA1 is defined by <0/1>; where 1 is the unintended output while logic 0 is the fault-free output after a w0



(a)



(b)

Figure 2.4. The effect of 5% and 10% variation on (a) doping concentration, and (b) the ratio of length over area. Results were obtained using [15] with R_{ON}=100Ω and R_{OFF}=100KΩ, $V_{write} = 1.2V$.

operation. The necessary condition for a test to detect all stuck-at faults requires that a 0 and a 1 must be read from each cell [11].

An open is an unintended defect due to imperfection in lithography and pattern process, missing materials, broken nanowires, and parameter fluctuation of CMOS-based devices [26]. Opens increase the resistivity of the affected rows or columns, and hence prevent writing the appropriate value inside the memristor. These defects are modeled by single stuck-at faults in memristors [19]. An open in a column may affect read and write operations to the cells along the column. Also an open in a row causes all cells after the

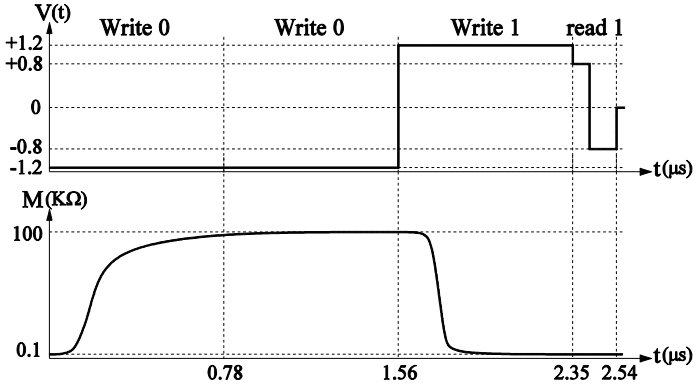
fault location to be inaccessible. These defects are also modeled by single stuck-at fault in memristors [19]. Note that the stuck-at fault model has been also proposed for conventional CMOS-based memory [13].

Slow write faults: A small decrease in dopant density (also called under-doping), is modeled as a slow write fault (SWF). During the write operation, the transition from one state to another is much slower than the expected behavior of memristor. Slow write 0 (SW0) is denoted in [24] by <1W0/X1>. X1 denotes the final state which can be either undefined (X) or 1. If the final state is 1 then the slow write fault is equivalent to the slow transition fault to state 0, also denoted as TF0. If the final state is X the slow write fault is also called the undefined state fault to state 0, also denoted as USF0 [24]. Similarly, the faults slow write 1 (SW1) denoted by <0W1/X0>, models dopant–related defects that impact the memristor's behavior when the final state is 1. They can be either TF1 or USF1.

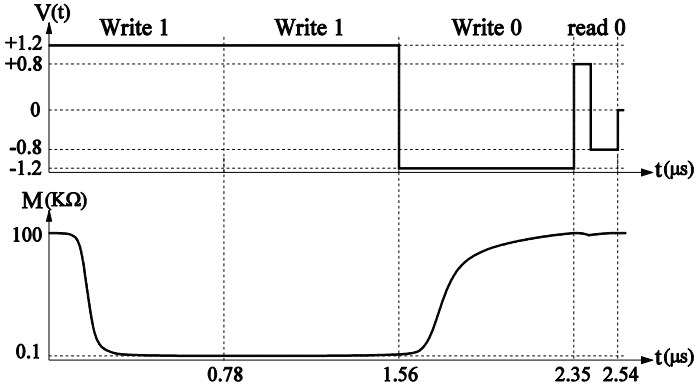
A test that sensitizes and detects all slow write faults must have the following necessary condition [11]: Each cell must undergo a rise transition and a fall transition, and be read after each, before undergoing any further transitions. For example, SW1 is sensitized by the w0 operation, followed by a *w1* operation. A subsequent *r1* operation will detect the fault.

Deep faults: Deep faults were introduced in [18] to model transitions that were believed to be slower than those modeled by slow write faults, occurring when many write operations precede a read. Simulations were presented that used a linear and symmetric I-V relation [3]. However, these faults are non-applicable to metal-oxide memristors whose behavior is nonlinear and asymmetric. Extensive experiments on different sizes with the accurate model in [15] were conducted. The results revealed that the transition

time from 0 to 1 and 1 to 0 were not impacted by the number of write operations prior to a read. As an example, Figure 2.5 (a) shows the transition time from 0 to 1 for the *w0*, *w0*, *w1*, *r1* sequence, and Figure 2.5 (b) shows the transition time from 1 to 0 for *w1*, *w1*, *w0*, *r0*. We observe strong similarity in the transitions shown in Figure 2.2 and Figure 2.5 (a). Likewise, the transition from 1 to 0 in Figure 2.2 and Figure 2.5 (b) are very similar. Therefore, tests for metal-oxide memristor defects should not consider deep faults, and only consider the slow write fault model.



(a)



(b)

Figure 2.5. (a) The transition time from 0 to 1 for the *w*0, *w*0, *w*1, *r*1 sequence and (b) the transition time from 1 to 0 for *w*1, *w*1, *w*0, *r*0 sequence. R_{ON}=100Ω and R_{OFF}=100KΩ, $V_{write} = \pm 1.2V$, $V_{read} = \pm 0.8V$.

Extrinsic faults occur due to defects in crossbar structure [19], address decoding error [13], imperfection in lithography, missing materials, broken nanowires [24] rather than the parametric variation in the memristor. This chapter considers shorted rows/columns (SRC) coupling fault as extrinsic faults.

Coupling fault: Coupling fault (CF) arises due to defects in crossbar structure rather than the parametric variation in the memristor [19]. A coupling fault (CF) means that a transition in memory cell j causes an unwanted change in memory cell i. A resistive short between adjacent rows or columns or between a row and a column is modeled as a SRC coupling fault [19]. A transition in aggressor cell results into the same transition in the victim cell which is horizontally or vertically adjacent to the aggressor cell.

Let x denote either value 0 or value 1. Let $\bar{x}$ denote the complement value of x. A SRC fault is defined by < xw1;0/1/-> or < xw0;1/0/->. The necessary condition for a test to detect all SRC faults is: The test must read x from cell 0, write $\bar{x}$ to cell 0, read x from cell 1, write $\bar{x}$ to cell 1, for the entire memory.

Notice that the above fault models are also adopted to conventional CMOS-based memory testing because defects also impact logical as well as temporal behavior.

2.3 The proposed test methods

2.3.1 March tests

The test operation *w0* is significantly slower than the *w1*, *r1* and *r0* [28]. Thus, the time performance of the existing March test is dominated by the *w0* operation. The main objective of the proposed March tests is to reduce the time duration of the *w0* operation. The test time is decreased by defining a new write operation $fw0$ that substitutes the *w0*

operation. As mentioned in Section 2.1, tests are done in elevated voltage 3.3V.

In defect free memory, the size of all memristors does not vary significantly because the amount of intra-chip variations typically does not exceed 10% [21, 22]. Let us denote the size of a memristor (length over area) as L/A. Since the geometrical variation induced by lithography process variations vary in the same way all over the chip, the average amount of variation in L/A can be determined by choosing randomly some cells and checking the upper bound in the resistance value. A March test that writes 0 and reads 0 at each cell will determine the smallest L/A among all cells.

In contrast to the normal mode of operation, during test the *r0* and *r1* operations use different reference currents which we call I_{ref_0} and I_{ref_1}, respectively. Their values depend on the logic state boundaries. As in [18-21], it is assumed that such currents have been determined. We note that $I_{ref_1} > I_{ref}$ and $I_{ref_0} < I_{ref}$. Let the respective resistive values be denoted as R_{ref_0} and R_{ref_1}. The resistive region between R_{ref_0} and R_{ref_1} is called the undefined region. (See also Figure 2.6) Testing considers the undefined region, as we mentioned earlier in the slow write fault model.

In analysis of a resistive cell, both size and doping parameters have been considered. The worst-case scenario occurs when the length increases while the area and doping concentration decreases. Doping variation affects only the rate of transition, and variation in L/A changes R_{OFF} whereas R_{ON} is practically invariant. (See also Figure 2.4) So the resistance of the cell in state 1 is practically invariant. A negative voltage (operation *w0*) across the memristor in state 1 may increase the memristance, and forces it to move out of state 1. Once this happens, the voltage is not further applied and a read operation detects whether the read resistivity has exceeded R_{ref_1}.

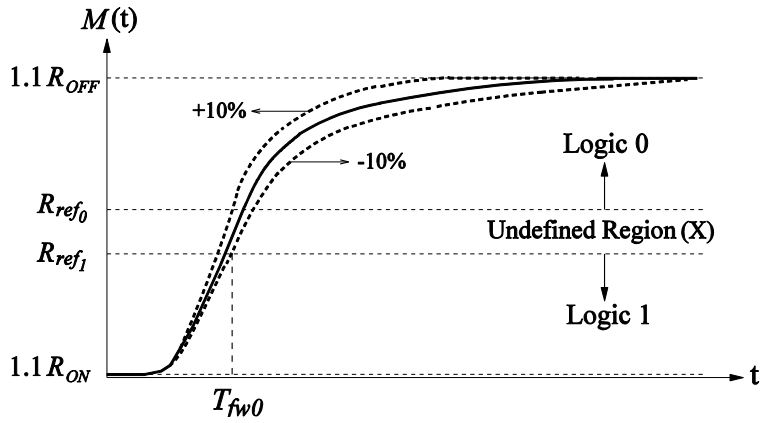


Figure 2.6. The undefined state for a memristor cell, and the value of T_{fw0} considering 10% doping variation. The memristor also suffers from 10% increment in size (L/A).

Let T_{fw0} denote the time for $fw0$. T_{fw0} must be long enough to move the cell out of state 1. Therefore, it should be longer than the sum of the t_{wait_0} and the time required to change the resistivity from R_{ON} to the boundary of state 1. (See also Figure 2.6)

The following presents March tests for different fault models. We assume that T_{w1} is as high as T_{fw0} (in practice it is less), and that $T_{r0} = T_{r1}$ (which is always the case). That way, we need only two different access times: T_r for read operations and $T_w \approx 2T_r$ for write operations. The access time for *w0* operation is the same as in normal mode of operation. All results are generated using the same write voltage to ensure a fair comparison between the proposed and existing tests.

Table 2.1 lists notations that will be used throughout this section. Notation || indicates parallel addressing of the cells in a tile. This operation is used to initialize all cells in a row to either 1 or 0. The driving current suffices as the number of cells per row is limited.

***Stuck-at Fault*:** The Existing March test for Stuck-At Fault (EMSAF) in equation (2.1) detects all stuck-at faults [11]:

Table 2.1. Notation for the March Tests.

Symbol	Operation
r	A read operation
w	A write operation
r0 (r1)	Read a 0 (1) from the memory location
w0 (w1)	Write a 0 (1) to the memory location
⇳	Addressing order can be either increasing or decreasing
||	parallel addressing of all cells in an entire row in a tile
$fw0$	Write a 0 to the memory location and quickly stop this operation after the initial resistivity changes, and the device conducts current.
r_{ref_1}	Read operation with reference current I_{ref_1}.
r_{ref_0}	Read operation with reference current I_{ref_0}.

EMSAF: {⇳ (*w1*) ; ⇳ (*r1*) ; ⇳ (*w0*) ; ⇳ (*r0*)} (2.1)

The test time is $2n(T_{w0} + T_r)$, where n denotes the number of memory cells.

The proposed March test does not apply *w0*. To detect SA1, we apply a $fw0$ in order to change the current state of the cell from 1 to a predetermined intermediate resistivity R_{ref_1}. Unlike the existing EMSAF, all cells are initialized to 1 by applying a parallel write operation for all cells in an entire row. A fast write $fw0$ from 1 to 0 followed by a read operation results in a resistivity value greater than R_{ref_1}. A read operation after $fw0$ will detect if there is slow transition from 1 to 0.

In summary, we test for SA1 with March {⇳ ($fw0$, r_{ref_1})}. The operations ($fw0$, r_{ref_1}) determine whether the cell is moved out of state 1. The reference current I_{ref_1} is the expected output current of the sense amplifier for resistivity value of R_{ref_1}. If there is a fault in the examined cell, the output current will be more than I_{ref_1} and the sense amplifier will show logic 1.

The proposed March test for all SA0 and SA1 is called Fast March test for Stuck-At Fault (FMSAF):

FMSAF: {‖ (*w1*)};{⇳ (*r1*) ; ⇳ ($fw0$, r_{ref_1})} (2.2)

The test time for the proposed FMSAF in an 8×8 tile is $(9n / 8)T_w + 2nT_r$. Open defects behave like SA0 and shorts to V_{dd} behave like SA1 [20]. Hence, the proposed FMSAF can also detect open and short defects.

***Transition Fault*:** The Existing March test for Transition Fault (EMTF) in equation (2.3) detects all transition faults [18]:

EMTF: {⇳ (*w0*, *w1*) ; ⇳ (*r1*) ; ⇳ (*w0*, *r0*)} (2.3)

EMTF uses $3n$ writes followed by $2n$ reads in a memory with n cells. The test time is $n(3T_{w0} + 2T_r)$.

The proposed March test to detect TF0 is {||(w1)};{⇳(fw0, r_{ref_1})}. All cells must be initialized to 1 by applying a parallel write operation in all rows. TF0 is sensitized by applying a $fw0$ in order to change the state from 1 to resistive value R_{ref_1}. One r_{ref_1} per cell will detect this fault since it tests whether the cell is fast enough to move from one state to another state. For an ideal cell, the output current of sense amplifier is expected to be less than I_{ref_1}. However, for a faulty cell, the memristance value is not in the expected range, and the output current is more than I_{ref_1}. Hence, the sense amplifier shows logic 1. The proposed March test for all TF0 and TF1 is called the Fast March test for Transition Fault (FMTF):

FMTF: {‖ (*w0*, *w1*)};{⇳ (*r1*) ; ⇳ ($fw0$, r_{ref_1})} (2.4)

The test time for the proposed FMTF in an 8×8 tile is $(n/8)(T_{w0} + 9T_w) + 2T_r$.

***Shorted Rows/Columns*:** The Existing March test for Shorted Rows/Columns (EMSRC) detect all SRC faults that propagate to a higher or lower address memory [19]:

EMSRC: {⇳(*w1*); ⇑(*r1*,*w0*); ⇓(*r0*,*w1*); ⇳(*w0*); ⇑(*r0*,*w1*); ⇓(*r1*,*w0*)} (2.5)

EMSRC uses $6n$ writes followed by $4n$ reads in a memory with n cells. The total test time is $n(6T_{w0} + 4T_r)$. EMSRC detects SRC faults by reading the value 1 (0) from cell i, writing 0 (1) to cell i and for the entire memory with higher (lower) address location. It tests whether a write operation to cell i changes the state of cell j.

The proposed March test for all SRC faults that propagate to a higher or lower address is called Fast March test for Shorted Rows/Columns (FMSRC). It is similar to EMSRC except that *w0* is substituted by $fw0$:

$$\text{FMSRC:}\quad \{\Vert(w1)\};\{\Uparrow(r1,fw0);\Downarrow(r_{ref_1},w1)\};\{\Vert(w0)\};\{\Uparrow(r0,w1);\Downarrow(r1,fw0)\} \tag{2.6}$$

The test time is $(n/8)(T_{w0} + 33T_w) + 4T_r$. Current I_{ref_1} that corresponds to resistivity r_{ref_1} distinguishes state 1 from resistivity R_{ref_1}.

The Existing March Test (EMT) in [19, 20, 22] detects all intrinsic and extrinsic faults:

$$\begin{aligned}\text{EMT:}\quad &\{\text{M0:}\Updownarrow(w0);\ \text{M1:}\Uparrow(r0,w1,r1);\ \text{M2:}\Downarrow(r1,w0,r0)\\ &\text{M3:}\Updownarrow(w1);\ \text{M4:}\Uparrow(r1,w0);\ \text{M5:}\Downarrow(r0,w1);\}\end{aligned} \tag{2.7}$$

The following explains how EMT sensitizes and detects the various faults. Faults SA1 are sensitized and detected by M2. Faults SA0 are sensitized and detected by M1. Faults TF1 are sensitized by M0 and the write operation *w1* of M1, and are detected by the read operation *r1* of M1. Faults TF0 are sensitized by the *w1* of M1 and the *w0* of M2, and are detected by the *r0* of M2. The SRC faults that propagate to a higher memory address are sensitized and detected by M1 and M4, and the SRC faults that propagate to a lower memory address are sensitized and detected by M2 and M5. Observe that the EMT is dominated by the *w0* operations.

The proposed March test to detect all SAFs, TFs, and SRCs is called the Fast

March Test (FMT):

$$\text{FMT:} \quad \{\Vert\,(w0)\};\{\Uparrow\,(r0, w1, r1); \Downarrow\,(r1, fw0, r_{ref_1})\};$$
$$\{\Vert\,(w1)\};\{\Downarrow\,(r1, fw0); \Uparrow\,(r_{ref_1}, w1)\} \tag{2.8}$$

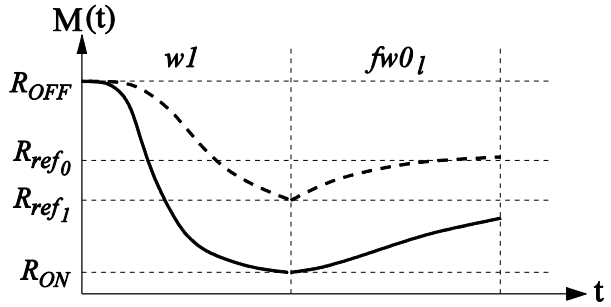
This test is much faster than the EMT. The total test time is $(n/8)(T_{w0} + 33T_w) + 6T_r$.

The March test of equation (2.8) has a wx followed by $\Uparrow(rx, \ldots, w\bar{x})$ and $\Downarrow(r\bar{x}, \ldots, wx)$ conditions, and thus also detects all address decoder faults (AFs) [13]. An address decoder fault (AF) occurs in any of the following scenarios: Multiple cells are accessed by a certain address, no cell is accessed by a certain address, a certain cell is not accessed by any address or a certain cell is accessed by multiple addresses [13].

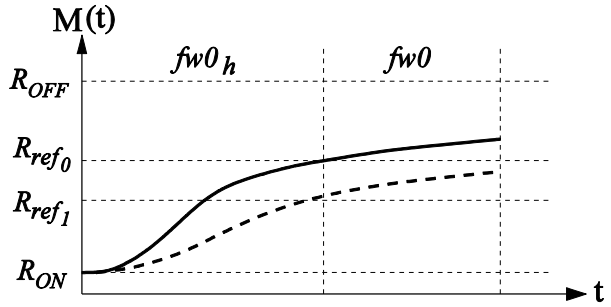
***Undefined State Fault*:** The proposed FMT can also detect USF1 which is explained in [25, 26]. USFs have emerged due to the analog nature of the memristor device. Traditional March tests such as the EMT in equation (2.7) cannot guarantee the detection of such a fault [26]. A normal *w1* operation will put the faulty cell into the undefined state X instead of state 1. A weak stress $fw0_l$ (such as $fw0$ with less duration) will change the state to 0. However, the time duration of $fw0_l$ is not enough to flip the state of a fault-free cell from 1 to 0. A read operation immediately after $fw0_l$ will detect the USF1. The reference current should be set according to the upper bound of the undefined region. (See also Figure 2.7 (a).) Any I_{mem} less than the I_{ref_0} results in detecting a USF1 in the cell. The proposed March test to detect all USF1s is called the Fast March test for Undefined State Fault 1 (FMUSF1):

$$\text{FMUSF1: } \{\Vert\,(w0)\};\{\Updownarrow\,(w1, fw0_l, r_{ref_0})\} \tag{2.9}$$

Changing 0 to 1 in FMUSF1 does not result to a robust test for USF0 since $fw1$ is



(a)



(b)

Figure 2.7. Simulation results for the fault free and faulty cell with (a) USF1 and (b) USF0.

not a robust operation. The following presents a test for USF0 that uses another fast write operation $fw0_h$ with more duration when compared to $fw0$. Its time duration should be enough to set the cell in the upper boundary of undefined region. (See also Figure 2.7 (b).) The proposed March test to detect all USF0s is called the Fast March test for Undefined State Fault 0 (FMUSF0):

$$\text{FMUSF0: } \{\Vert (w0)\};\{\Updownarrow (fw0_h, fw0, r_{ref_0})\} \tag{2.10}$$

2.3.2 Experimental evaluation

Let T_{r1}, T_{w1}, T_{r0}, T_{w0} denote the times for *r1*, *w1*, *r0*, *w0*, respectively. Table 2.2 compares T_{fw0} with T_{r1}, T_{w1}, T_{r0}, and T_{w0}. These times were obtained with SPICE simulation using [15] under different size and doping variations for the metal-oxide memristor. Simulations were done using 45nm CMOS technology with R_{ON} =100Ω,

Table 2.2. Time Duration (in ns) of $w1$, $r1$, $w0$, $r0$, $fw0$ for Different 45nm Instances when Size Increases and Doping Decreases. R_{ON} =100Ω, R_{OFF} =100KΩ, R_{ref_1} =40KΩ, R_{ref_0} =60KΩ, $V_{write} = \pm$3.3V, V_{read} =0.8V.

Variation	T_{r1}	T_{w1}	T_{r0}	T_{w0}	T_{fw0}
0%	33.5	59.0	33.5	255	61.7
1%	34.3	61.3	34.3	262	64.1
2%	35.2	63.2	35.2	269	66.7
5%	36.5	68.0	36.5	280	73.0
10%	38.4	71.9	38.4	288	**77.4**

R_{OFF} =100KΩ, R_{ref_1} =40KΩ, R_{ref_0} =60KΩ, $V_{write} = \pm$3.3V, V_{read} =0.8V, and the temperature was set to 27°C. All memristor parameters were set as in [35]. SPICE simulation revealed that $I_{ref_1} = 82$µA and $I_{ref_0} = 55$µA.

Table 2.2 shows the impact of doping and size (L/A) variations. The first row represents the nominal condition without variation. The second row represents the instance when L/A increased by 1% and doping decreased by 1%. The last row represents the worst case instance which corresponds to a 10% increase in L/A with a 10% decrease in doping. Columns two to five show the values of T_{r1}, T_{w1}, T_{r0}, and T_{w0} for the listed instances when the wait time is maximized and equal to 10% of the transition time. Observe that T_{w0} is always very high compared to T_{fw0}. Also observe that T_{fw0}, T_{w1}, T_{r1}, and T_{r0} are always in the same order. Similar characteristics are observed for different values of R_{OFF} / R_{ON}. All read operations are reported at 0.8V. Observed that T_{r1} and T_{r0} are very fast because the second pulse that nullifies the read disturbance is not needed during the test.

The following simplifies test operation times by assuming that T_{w0} is approximately

Table 2.3. Average Energy Needed for $w1$, $r1$, $w0$, $r0$, and $fw0$ in a Random Cell Using 45 nm CMOS Technology with R_{ON} =100Ω, R_{OFF} =100KΩ, V_{write} = ±3.3V, and V_{read} =0.8V.

Operation	$w0$	$r0$	$w1$	$r1$	$fw0$
Energy (pJ/cell)	29.07	0.23	3.74	0.23	3.98

7.5 times slower than any read operation, and that T_{w1} is approximately two times slower than any read operation. See also Table 2.2.

Based on the above assumptions, in a memory with n cells using 8×8 tiles, the test time of EMSAF is $19nT_r$ whereas the time of the proposed FMSAF is $4.25nT_r$. This is almost a 4.5X speed-up over EMSAF. The test time of EMTF is $24.5nT_r$ but the test time of the proposed FMTF is $5.18nT_r$. This is almost a 5X speed-up over EMTF. Furthermore, the EMSRC detects SRCs in $49nT_r$ time while the test time for FMSRC is $13.18nT_r$. This is a 4X speed-up over EMSRC. (See also Table 2.4.)

Table 2.3 shows the average energy per operation in a random cell during test. Observe that $fw0$ requires significantly less energy when compared to *w0*.

Table 2.4 summarizes and compares the proposed FMT March test to the EMT.

Table 2.4. Time and Average Energy of Existing and Proposed March Tests Using 45nm CMOS Technology with $R_{ref_1} = 0.4R_{off}$, $R_{ref_0} = 0.6R_{off}$, V_{write} = ±3.3V, V_{read} =0.8V.

Existing March Test			Proposed Fast March Test			% improvement in test time using tile (8×8)		% improvement in average test energy per cell
Name	Test Time (*n* cells)	Energy per cell (PJ)	Name	Test Time (*n* cells)	Energy per cell (PJ)	R_{ON}=100Ω R_{OFF}=100kΩ	R_{ON}=100Ω R_{OFF} =150kΩ	
EMSAF	$19nT_r$	33.25	FMSAF	$4.25nT_r$	8.18	77.6	79.0	75
EMTF	$24.5nT_r$	61.7	FMTF	$5.18nT_r$	36.4	78.8	80.5	41
EMSRC	$49nT_r$	98.1	FMSRC	$13.18nT_r$	49.3	73.5	74.4	50
EMT	$51nT_r$	99.3	FMT	$15.18nT_r$	50.2	70.1	71.3	49

Column 7 shows the improvement of the test time with $R_{ON} = 100\Omega$ and $R_{OFF} = 100\text{K}\Omega$. Similar results with $R_{ON} = 100\Omega$ and $R_{OFF} = 150\text{K}\Omega$ are given in Column 8. An 70% improvement in test application time is observed. Columns 6 and 7 list the time of the existing March tests minus the time of the proposed fast March tests over the time of the existing March tests. The last column also lists the average energy improvement to test a memory cell. The average test energy per cell is reduced at least by 40%.

2.3.3 More reliable March tests considering sneak paths

This section introduces March tests for testing the crossbar architecture of Figure 2.3 in the presence of sneak paths. The sneak path error may occur during a *r0* operation from a cell at logic 0 when there is also a parallel path that has several cells at logic 1. Let $\mathrm{M}_{i,j}$ be the memristor in row i and column j. The proposed fast write operation $fw0$ is applied to reduce the test time of existing sneak path March tests. We use the analysis in [44-46] to determine the number of cells in the RoD.

Figure 2.8 shows a sneak path with ordered cells $\mathrm{M}_{1,4}$, $\mathrm{M}_{3,4}$ and $\mathrm{M}_{3,2}$, it is labeled as Path 2. Assume an *r0* for cell $\mathrm{M}_{1,2}$ in logic 0, and let $\mathrm{M}_{1,4}$, $\mathrm{M}_{3,4}$ and $\mathrm{M}_{3,2}$ be in logic 1. Instead of the current relating to Path 1 through $\mathrm{M}_{1,2}$, the output current I_{output} is the sum of the current of Path 1 through $\mathrm{M}_{1,2}$ and the current of Path 2 through $\mathrm{M}_{1,4}$, $\mathrm{M}_{3,4}$ and $\mathrm{M}_{3,2}$. Therefore, I_{output} erroneously determines logic 1 in cell $\mathrm{M}_{1,2}$. This section presents reliable tests that avoid this side-effect.

To avoid sneak paths during test, [45] proposed to ground all rows expect the one being read. However, this technique is power consuming due to lower equivalent resistance that affects the measured current. Instead, this paper uses the symmetric and asymmetric grounding techniques proposed in [44] which are more efficient.

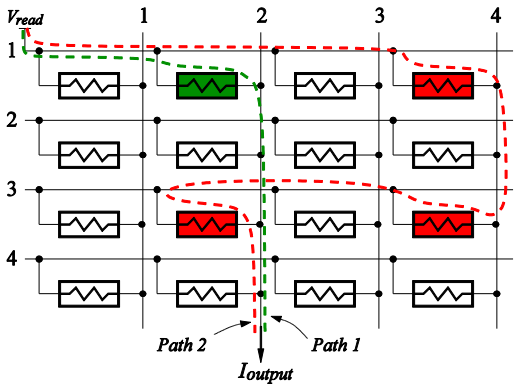


Figure 2.8. Sneak-path testing in a 4×4 high density crossbar memory.

An array A is a sneak-path-free array if and only if the “1”s in every two rows have either full-overlap or no-overlap [44]. The proposed method uses the above-mentioned conditions.

Let b be an integer. The proposed method grounds all rows except: The examined row i, b rows above the row i, and b rows below the row i. This operation controls the power consumption due to reduced grounding, and sneak paths are limited to the set of the $2b + 1$ ungrounded rows which must be controlled during the test.

During the *r0* operation, all the cells in the set of ungrounded rows are initialized to logic 0 (high resistance) before applying *r0* to any cells in row i. This reduces the effect of the bounded number of sneak paths in the set and decreases significantly the currents through the existing sneak paths. The RoD is the union of all cells in ungrounded rows. Therefore, the RoD for an $n \times n$ array contains $n(2b + 1)$ cells. The reference currents in this algorithm are the same as the ones used for the March tests in equation (2.1) through equation (2.10).

The following notation is used at the March tests of this section.

$\updownarrow_{i,\pm b}$: All cells in rows between $(i - b)$ and $(i + b)$ are accessed in either increasing

or decreasing order.

$\downarrow_{i,\pm b}$ (resp. $\uparrow_{i,\pm b}$): All cells in rows between $(i-b)$ and $(i+b)$ are accessed in decreasing (resp., increasing) order.

$\downarrow_{i,+b}$ (resp. $\uparrow_{i,+b}$): All cells in rows between i and $(i+b)$ are accessed in decreasing (resp., increasing) order.

$\downarrow_{i,-b}$ (resp. $\uparrow_{i,-b}$): All cells in rows between $(i-b)$ and i are accessed in decreasing (resp., increasing) order.

$\updownarrow_i$: All cells in rows i are accessed in either increasing or decreasing order.

***Stuck-at Fault*:** The proposed FMSAF in equation (2.5) is modified to reduce the effect of unwanted sneak paths during the test by grounding rows. The proposed test to detect all SA1 and SA0 is called the Fast Sneak Path Test for SAF (FSPTSAF):

$$\text{FSPTSAF: } \{\Vert\ (w1)\};\{\Updownarrow(r1)\};\ \{\updownarrow_{i,\pm b}(fw0);\updownarrow_i(r_{ref_1})\} \tag{2.11}$$

The test time is the same as that of FMSAF. The value of b controls the energy per cell. Table 2.5 shows that the energy per cell increases as b decreases. FSPTSAF is sneak path free when $b=0$.

***Transition Fault*:** The proposed test to detect all TFs is called the Fast Sneak Path Test for TF (FSPTTF):

$$\text{FSPTTF:}\{\Vert\ (w0,w1)\};\{\Updownarrow\ (r1)\};\ \{\updownarrow_{i,\pm b}(fw0);\ \updownarrow_i(r_{ref_1})\} \tag{2.12}$$

Test time is the same as in FMTF. Table 2.5 shows a reduction in energy as b increases, and the test is sneak path free when $b=0$.

***Shorted Rows/Columns*:** The proposed Fast Sneak Path Test for SRC is called FSPTSRC:

$$\text{FSPTSRC: } \{\Vert(w1)\};\{\Uparrow(r1,fw0)\};\{\downarrow_{i,+b}(r_{ref_1},w1)\};\{\Downarrow(r1,\ fw0)\};\{\uparrow_{i,-b}(r_{ref_1},w1)\} \tag{2.13}$$

Table 2.5. The Average Energy of the Pulse Applied to a Row and a Column in Order to Test One Cell Using Tile (8×8) for the Proposed Sneak-path March Tests.

b	FSPTSAF	FSPTTF	FSPTSRC	FSPT	FSPTSAF	FSPTTF	FSPTSRC	FSPT
0	12.3	43.0	54.9	55.3	12.7	44.3	57.1	58.0
1	11.4	40.7	53.1	53.6	12.0	42.1	53.9	54.5
2	8.9	39.1	52.1	52.1	9.3	40.1	53.2	53.2

The test time is the same as in FMSRC. Table 2.5 shows that the energy per cell increases as b decreases, and when $b = 0$, the test is free sneak paths effects.

The proposed test for all SAF, TF, SRC is called Fast Sneak Path Test (FSPT):

$$\text{FSPT: } \{\Vert(w1)\};\{\Uparrow(r1, fw0); \downarrow_{i,\pm b}(r_{ref_1}); \downarrow_{i,+b}(r_{ref_1}, w1)\}$$
$$\{\Vert(w0)\};\{\uparrow_{i,-b}(r0,w1); \Downarrow(r1, fw0)\} \quad (2.14)$$

Table 2.5 shows the benefits in energy as b increases and when $b = 0$, the test is free sneak paths effects. Similar observations hold when experimenting with different combinations of R_{OFF} and R_{ON}.

2.4 DfT for the proposed tests

Memristor operations either in normal mode or testing mode rely mainly on the duration of the access time on columns and rows. Every operation requires a specific access time [11, 25]. A programmable DfT scheme is presented in order to be able to assign different access times for the proposed test operations.

Figure 2.9 depicts the proposed DfT to implement the FMT March tests in equation (2.8). The description assumes 45nm technology. The DfT circuit contains one timer to control the access time duration of the write operation called the W-Timer. At any time, one of the timers is selected. The timer is shown on the left part of Figure 2.9, and the

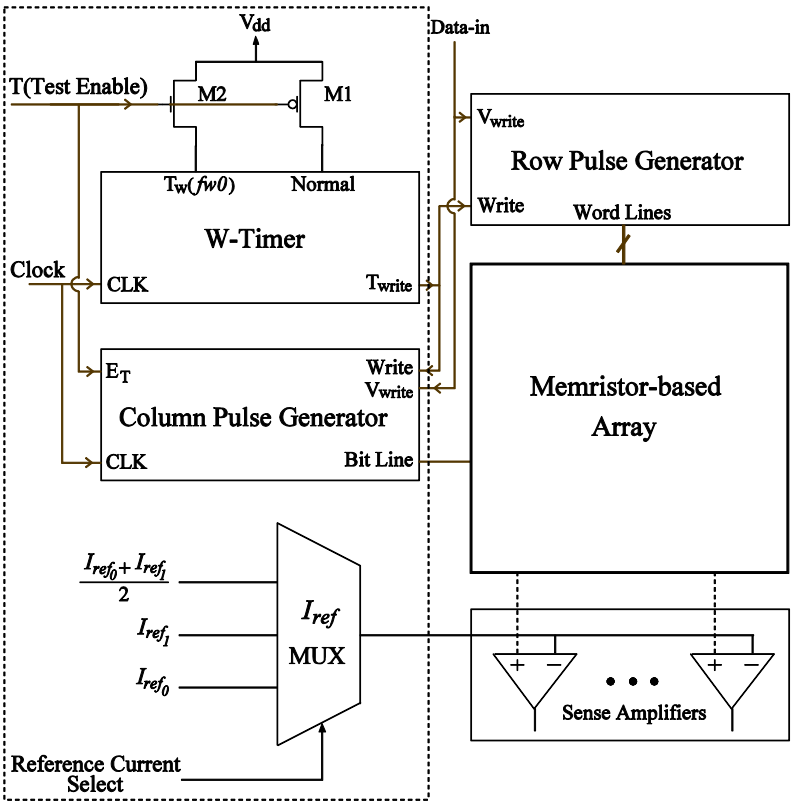


Figure 2.9. Schematic of the proposed DfT.

associated selection hardware is shown above them. The timer supplies appropriate time duration to the row and column pulse generations. The DfT also consists of a reference current selection scheme and associated sense amplifiers. (See bottom of Figure 2.9.) Therefore, the write and read operations are mode dependent.

The W-Timer sets two different access times for write: The normal mode write time which is set to *max*{T_{w1}, T_{w0}} for *w0* and *w1* in normal mode operation, and a time T_w which is equal to T_{w1} and T_{fw0}. In the test mode, T_{w0} is the same as in the normal mode.

Test signal T can activate the appropriate access time by using two different transistors: M1 (PMOS) and M2 (NMOS). The timer activates the row and column pulse generators which can supply the voltage to the rows and columns, respectively. M1 and M2 switch between the normal mode and the test mode. When T is low, M1 is switched

on and the normal mode is activated. The output of the timer (T_{write}) is activated for nominal access time T_{write}=T_{w0}=288ns. On the contrary, when the T is high, M2 is switched on. This activates the fast write operation by supplying the pulse generator for T_{write}=T_w=77.4 ns. Time T_{w0} is also used for *w1* in test mode.

During the read operation, the sense amplifier compares the I_{mem} with the I_{ref}. The sense amplifier has three different reference currents. During the normal mode, I_{ref} is set to $(I_{ref_1} + I_{ref_0})/2$. During the test mode, I_{ref_1}=82μA and I_{ref_0}=55μA are used as reference currents.

In order to calibrate the access times during testing, [24] and [26] proposed a programmable DfT with different access time settings. This circuit copes with the unexpected effect of process variations during post-silicon test. Therefore, the timer can be tuned during the test [26].

Figure 2.10 shows the programmable version of the proposed DfT of equation (2.8). Depending on the selection signals S[3:1], the decoder will set W-Timer to activate one of the time durations. The timer in turn activates the row and column pulse generators which can supply the voltage to the memristor-based array. The circuitry inside the dotted line can be enhanced with the peripheral circuitry in [19, 47] or the BIST architecture in [22] to accommodate in FSPT March test in equation (2.14) that considering sneak path effects.

Figure 2.11 shows the proposed DfT for detecting USFs using FMUSF1 and FMUSF0 of Section 2.3.1. The W-Timer sets four different time durations for write operations: A write time which is set to *max*{T_{w1}, T_{w0}} for *w0* and *w1* in the normal mode of operation, and three different times T_{fw0}, T_{fw0_l}, T_{fw0_h} for the fast writes $fw0$, $fw0_l$,

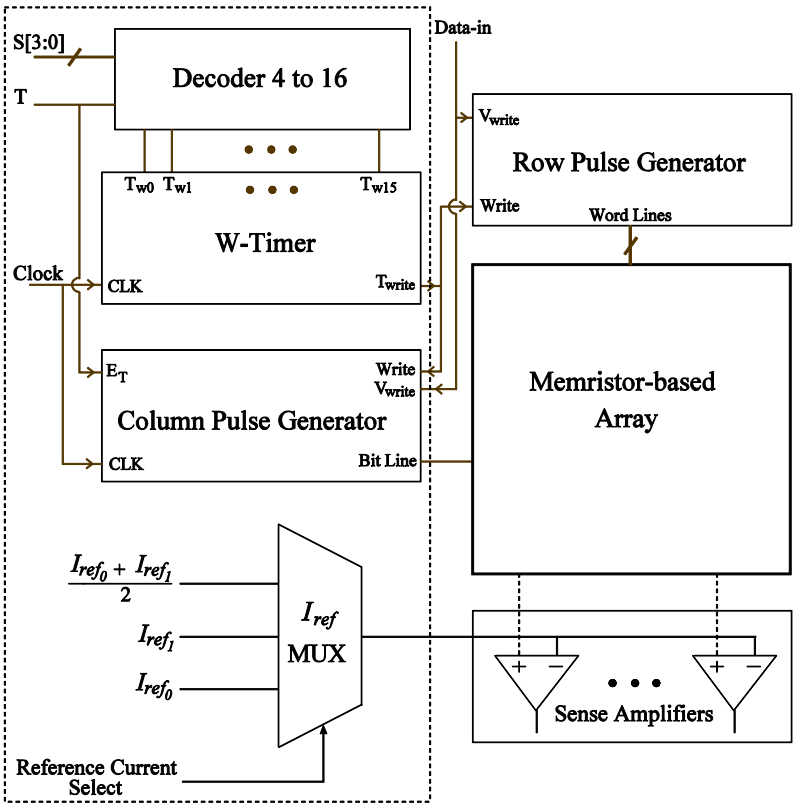


Figure 2.10. Schematic of the proposed programmable DfT.

$fw0_h$, respectively. Depending on the selection signals S0 and S1, the decoder will set the W-Timer to activate one of the time durations. During the test mode, the reference current I_{ref_0} is set to 55μA.

The total area overhead of the proposed DfT depends only on the number of columns in the two dimensional memory because the timer, the decoder, the column pulse generator (that implements parallel writes in a row), and the reference current multiplexer must access each column. These components are shown inside a dotted line in Figure 2.10 and Figure 2.11. The DfT overhead does not depend on the number of rows and the number of tiles.

We implemented the proposed DfT in Cadence using 45nm CMOS technology. The timer was implemented by a 5-bit asynchronous counter which contains five J-K Flip-

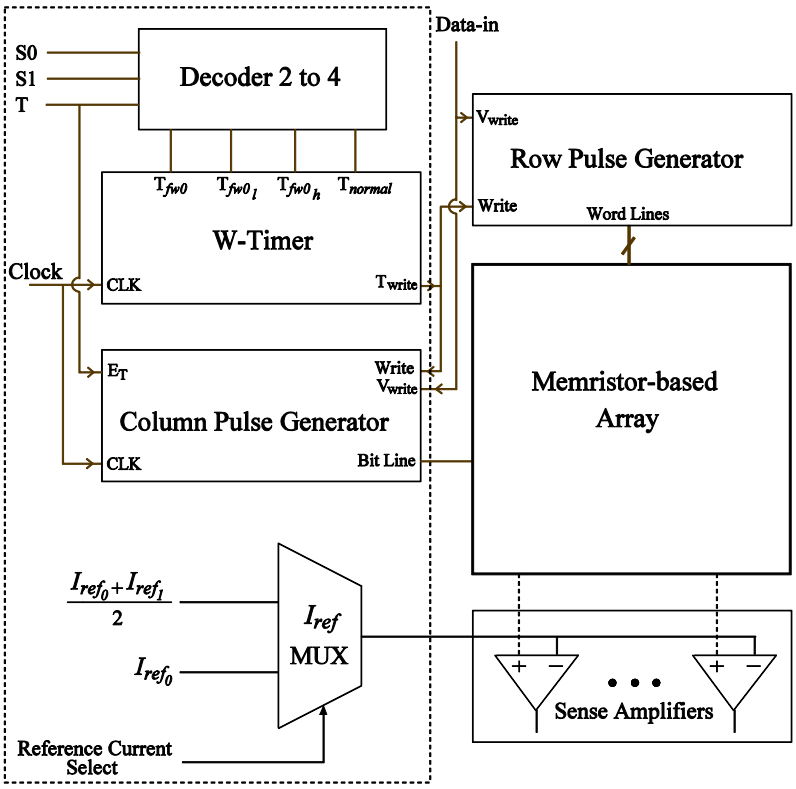


Figure 2.11. Schematic of the proposed USF DfT.

flops and three NAND gates. The column pulse generator consists of two NMOS transistors which connect the appropriate voltages to the column during test and normal mode of operations. The area overhead of the designed DfT was found to be only 8.875µm2 per column.

2.5 Conclusions

A methodology for testing the bi-state hybrid crossbar architecture in [28] has been presented. The proposed March test used a new fast write operation and reduced the test application time by 70% and the test energy by 40%. The method was extended to consider sneak paths in order to increase reliability with similar test application time. A programmable DfT Scheme has been proposed to implement the methods, and its

overhead was analyzed.

The proposed methods have been presented assuming that each memory cell is a bipolar metal-oxide memristor which is a popular technology. The methods can be generalized to other types of memristors as long as they have nonlinear and asymmetric characteristics in the switching parameters.

CHAPTER 3

A GENERALIZED APPROACH TO IMPLEMENT EFFICIENT CMOS-BASED THRESHOLD LOGIC FUNCTIONS

3.1 Introduction

Threshold Logic Gate (TG) is a promising candidate for the future digital circuits. Recent synthesis approaches show that TG-based circuits exhibit less delay, power dissipation, and silicon area [48-52]. A Boolean function that can be implemented as a single TG is called Threshold Logic Function (TF). In a TF, there is an integer weight for each input. When the input is set to binary value 1 then the weight is active. An input pattern evaluates the function to logic one only when the sum of the active weights is greater than (or equal) to a predetermined integer weight value called the threshold weight [57-59]. Otherwise, it evaluates the function to logic 0. In this paper, such a TF function is called a 1^{st}-order TF and will be denoted as a 1-TF. A n-input 1-TF $f(x_1, x_2, \dots, x_n)$ is formally defined as [58]:

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & if \quad \sum_{i=1}^{n} w_i \cdot x_i \geq w_T \\ 0 & otherwise \end{cases}, \tag{3.1}$$

where x_i, $i = 1, \dots, n$, are binary input variables, w_i is the weight corresponding to the i^{th} input, and w_T is the threshold weight (threshold value). The 1-TF $f(x_1, x_2, \dots, x_n)$ is also uniquely identified by the set of threshold and input integer weights $[w_1, w_2, \dots, w_n; w_T]$.

1-TF implementations consist of three components: two differential networks (input networks) and a sensor (sense amplifier) [51-57]. One input network implements positive

input weights and negative threshold weight and the other one implements the negative input weights and the positive threshold weight. The sensor evaluates the output by comparing and amplifying the difference between the sum of active weights of the two input networks.

In CMOS-based TGs, the power dissipation depends primarily on two factors: the transistor count of the input networks which is the total number of unit size transistors that implements weights and the sensor size which is proportional to the transistor count of input networks [52]. Input networks implement the weights with NMOS (or PMOS) transistors connected in parallel. Let X denote the width of a minimum size transistor which implements the unit weight. Each transistor implements a weight w and its width is $w \cdot X$, and the gate of the transistor is connected to an input. (The area of a transistor with width $w \cdot X$ is practically the same as w minimum size transistor.) The gate of the NMOS transistor for the threshold is connected to the power supply (it is active for all input patterns). The length of all transistors is the same and is determined by the used technology. The less transistor count in input networks, the lower the power dissipation in the differential network of a TG is. Furthermore, a reduced transistor count subsequently reduces the sensor size, which, in turn, decreases further the power dissipation [52].

This chapter proposes a new method to reduce the transistor count of the input networks by introducing non-integer weights. This method is applicable to all existing TG implementations. In order to demonstrate the impact of non-integer weights on TGs in terms of area, power dissipation, and delay, this chapter focuses on the current-mode TGs (CTGs) implementation which is a popular PMOS- and NMOS-based

implementation. (See [52-53], among others.)

A small fraction of binary functions are 1-TFs [58], and this limits the impact of TGs in digital circuit synthesis. Thus, our focus shifts on identifying TFs that are not 1-TF. A higher order TF, also called k^{th}-order TF (k-TF) in this chapter, was introduced in [60]. For each input pattern, a weight in a k-TF can be activated by a group of k active inputs, $1 \leq k \leq n$. Such a weight is called a k-weight.

Let x_{i_1}, $i_1 = 1, \ldots, n$, be binary input variables, w_T denote the threshold weight, and integer weight $w_{i_1,i_2,\ldots,i_m}$ denote the m-weight, $1 \leq m \leq k$, that is activated by the group of m inputs i_1, i_2, …, and i_m. The k-TF formulation for an n-input function is [60]:

$$f(x_1, x_2, \cdots, x_n) =$$

$$\begin{cases} 1 & if \quad \sum_{i_1=1}^{n} w_{i_1} x_{i_1} + \cdots + \sum_{i_1=1}^{n-m+1} \sum_{i_2=i_1+1}^{n-m} \ldots \sum_{i_m=i_{m-1}+1}^{n} w_{i_1,i_2,\ldots,i_m} x_{i_1} x_{i_2} \ldots x_{i_m} \\ & \quad + \cdots + \sum_{i_1=1}^{n-k+1} \sum_{i_2=i_1+1}^{n-k} \ldots \sum_{i_k=i_{k-1}+1}^{n} w_{i_1,i_2,\ldots,i_k} x_{i_1} x_{i_2} \ldots x_{i_k} \geq w_T \\ 0 & otherwise \end{cases}, \quad (3.2)$$

Let us consider the recently proposed synthesis approach in [51] where a flip-flop at the output of the circuit together with a portion of the predecessor combinational logic is replaced by a single TG. Since more functions can be identified as a single k-TG by equation (3.2), each output cone will more likely be implemented with fewer gates and one k-TG as the final gate.

This chapter presents an Integer Linear Programming (ILP) formulation to assign efficiently weights to a k-TF so that the CTG implementation has low transistor count, and, subsequently, low power and delay. Weight variations due to CMOS-aging, circuit parasitics, and process variations are also taken into consideration.

It will be shown that many more TFs can be implemented as CTG without increased transistor count when compared to CTGs using traditional 1-TF definition. Moreover, the cost of certain CTGs that are 1-TF is reduced when considering the k-TF definition.

This chapter is organized as follows. Section 3.2 provides preliminaries on 1-TFs, including a scalable integer linear programming (ILP) based method to identify 1-TF and assign weights. Section 3.3 proposes an ILP capable of reducing the transistor count using rational weight values. A new method to identify and implement higher order TFs is presented in Section 3.4. An efficient approach to implement higher order TFs using rational weights is presented in Section 3.5. Section 3.6 provides experimental results. Section 3.7 concludes the paper and outlines future work that includes fast synthesis of complex circuit specifications and beyond CMOS k-TF implementations. We will build upon existing 1-TF based synthesis methods [51-72] and 1-TF resistive-based implementations [50, 75-79].

3.2 Preliminaries on the algorithmic infrastructure

This section provides with definitions and properties of 1-TF implemented using integer weights. It is also outlines some algorithmic aspects for scalable identification of threshold logic functions.

Positive (negative) function: Assume that function f is expressed in a disjunctive form. Function f is positive (negative) in variable x_i if the variable $\bar{x}_i$ (x_i) does not appear in the expression of f. Function f is a positive (negative) function if it is positive (negative) in all variables [58].

Unate Function (UF): Assume function f is expressed in a disjunctive form. f is unate in variable x_i if f is either positive or negative in variable x_i [58]. A function is a UF if it is unate in all variables. In other words, a function f is a UF, if and only if, for each variable x_i, $f_{x_i}\ (f_{\bar{x}_i}) \supseteq f_{\bar{x}_i}(f_{x_i})$.

Non-unate functions are called Binate Functions (BFs). All 1-TFs are UF [58]. However, higher order TFs may be BF [60].

Modified Chow's Parameters [58]: For an n-input function, the Modified Chow's parameter (m_i) of variable x_i, $1 \leq i \leq n$, is defined as the difference between the number of fully specified product terms in the onset of the function that have $x_i = 1$, and the fully specified product terms in the onset of that have $x_i = 0$.

Let $\vec{m}_f = (m_1, \dots, m_n)$ be the set of Modified Chow's parameters of all n variables for an n-input UF f. Function f is positive (negative) in variable x_i, if and only if, $m_i > 0$ ($m_i < 0$). Therefore, function f is a positive (negative) UF if all members of set $\vec{m}_f$ are non-zero and positive (negative) [58].

Negation property [58]: The negation of any variable results into a new set of weights. Let $f(x_1, x_2, \dots, x_i, \dots, x_n)$ be $[w_1, w_2, \dots, w_i, \dots,\ w_n; w_T]$. The negation of variable x_i, $(x_i \rightarrow \bar{x}_i)$, changes the weight configuration to $[w_1, w_2, \dots, -w_i, \dots,\ w_n; w_T - w_i]$.

In order to determine if a function is a 1-TF, we form an ILP constraint per input pattern using the right-hand side of equation (3.1) according to the binary evaluation of the function for the pattern [58]. However, the ILP requires more variables in order to implement TFs with low hardware overhead. The objective function in the ILP must minimize the absolute value of the weights, some of which may be negative. Negative weights can be assigned using three variables for each weight. (Details are omitted for

brevity.) However, such an approach introduces unnecessarily many variables and impacts ILP scalability. The following present an improved ILP where only one variable per weight is needed.

Let f be the examined non-positive UF. First, we find the Modified Chow's parameter for every variable. Every variable x_i that corresponds to negative weight w_i has a negative Modified Chow's parameter. Such variables must be complemented. This implies that the weight w_i will be activated when $x_i = 0$. The new set of input variables ensures that all weights are positive.

Next, we form an ILP for function f with the new set of variables, one variable per weight and an additional variable for the threshold weight. There is one constraint per input pattern to satisfy the functionality, and one constraint per variable that bind the range of that variable. The ILP objective minimizes the sum of the variables. The weight configuration is assigned from the ILP solution and the negation property. The following example illustrates the approach.

***Example 1*:** Consider the three-input function $F_1 = x_2 + x_1 x_3'$. In F_1, $\vec{m}_{F_1} = (1,3,-1)$. Parameter m_3 is negative, and hence variable x_3 must be complemented before forming the ILP. Table 3.1 lists the inequalities extracted from the truth value of F_1 using equation (3.1). The first three columns show the truth table of function F_1. Column four shows the linear inequalities. Weights w_1 and w_2 are activated when $x_1 = 1$ and $x_2 = 1$, while weight w_3 is activated when $x_3 = 0$. The last row shows the objective function that minimizes quantity $w_T + \sum_{i=1}^{3} w_i$. For the set of constraints in Table 3.1, $[w_1, w_2, w_3; w_T] = [1,2,1;2]$ is an optimum solution. Hence, F_1 is a 1-TF and its weight configuration, using the negation property, is $w_{F_1} = [w_1, w_2, w_3; w_T] = [1,2,-1;1]$. □

Table 3.1. The Linear Inequalities for $F_1 = x_2 + x_1 x_3'$ with Activation Signals x_1, x_2, and x_3'.

Truth Table			Inequalities
Input Value $(x_1 x_2 x_3)$		F_1	
P_0	000	0	$w_3 < w_T$
P_1	001	0	$0 < w_T$
P_2	010	1	$w_2 + w_3 \geq w_T$
P_3	011	1	$w_2 \geq w_T$
P_4	100	1	$w_1 + w_3 \geq w_T$
P_5	101	0	$w_1 < w_T$
P_6	110	1	$w_1 + w_2 + w_3 \geq w_T$
P_7	111	1	$w_1 + w_2 \geq w_T$
Minimize:			$w_T + \sum_{i=1}^{3} w_i$

The performance of the CMOS transistor which implements a weight is impacted by circuit parasitics, process variations, and aging. Transistor aging is caused by Bias Temperature Instability (BTI), dielectric breakdown, and hot career injections [72-74] that shift the threshold voltage and decrease the current through the transistors [80-81]. This is called weight aging. Process variations impact transistor width and length, and therefore they may modify the designed weight. However, all CTG weights will be shifted by the same factor and in the same direction, and therefore weight assignment is not that sensitive to process variations. Finally, parasitics are evaluated accurately with post-layout simulations.

Let value C denote the maximum weight deviation due to the above factors. It is obtained with SPICE simulations on the predetermined technology as explained in Section 2.6. Let $|w|$ denote the absolute value of weight w. The pattern dependent inequalities of the ILP are rewritten as $\sum_i (w_i - C \cdot |w_i|) \cdot x_i > w_T + C \cdot |w_T|$ when function

evaluates to 1, and as $\sum_i (w_i + C \cdot |w_i|) \cdot x_i < w_T - C \cdot |w_T|$ for the remaining input patterns. The above may only change the total sum of weights. For example, the weight configuration in Example 1 considering $C = 5\%$ becomes $w = [w_1, w_2, w_3; w_T] = [2,4,-2;1]$. This weight assignment results in a reliable CTG implementation. □

3.3 Efficient design of first-order threshold functions based on rational weights

The previous section described an ILP formulation that implements 1-weight by assigning an integer value w to each input variable and the variable for the threshold. This section shows how to form an ILP capable of implementing weights that are fractions w/l, where w and l are integers, and is an extension of the preliminary results presented in [82] for the special case where $l = 2$.

In contrast to the ILP method in the previous section, each weight value assigned by the new ILP is different from the implemented weight. The novelty is that the ILP assigns an integer value w to each variable so that the value of the respective weight when evaluating the TF (as in Equation 1) is w/l for some predetermined integer l. The flexibility of implementing rational weights results into significant reduction in the transistor count of CTGs with subsequent reduction in power and delay.

In CTGs that have been identified as 1-TF, each integer weight is implemented by a component that is connected to the components for the other weights, including the threshold. Each input weight component is controlled by an input variable. Such weights and components are called 1^{st}-order weights (or 1-weight) and 1^{st}-order components, respectively.

Let X denote the width of a minimum size transistor and I the active current through this transistor. The CTG implementation in [53] implements an integer 1-weight with value w using either a single NMOS (or PMOS) transistor with width $w \cdot X$ or w minimum size transistors which are connected in parallel. The active current through this component is $w \cdot I$. A transistor (weight) is active when its corresponding input is set to 1 (or 0 in case of using PMOS).

The proposed approach implements a rational 1-weight using multiple minimum size transistors connected in series. A 1-weight with value w/l is implemented with l NMOS (or PMOS) transistors which are connected in series. The transistor gates are connected to each other and are controlled by the corresponding input. The width of each transistor is $w \cdot X$ and the active current through them is $\frac{w}{l} \cdot I$. The transistor count of this component is l times the transistor count of a 1^{st}-order component that implements an integer 1-weight with value w.

Figure 3.1 shows the 1^{st}-order components that implement rational 1-weights with value $1/j$, for $1 \leq j \leq l$, given a predetermined integer l. The figure also shows the active current through them. The active current decreases when j increases. It is true that adding a transistor increases the capacitance of that component, and this increases the power dissipation. However, when a TF is implemented by rational components of Figure 3.1, the total transistor count of the gate reduces, and this reduces significantly the power dissipation of the gate.

Let x_i, $i_1 = 1, \ldots, n$, be binary input variables, j, w_i^j, and w_T^j integer values. w_i^j the 1-weight component with value w_i^j/j corresponding to the i^{th} input, $1 \leq j \leq l$, and w_T^j the

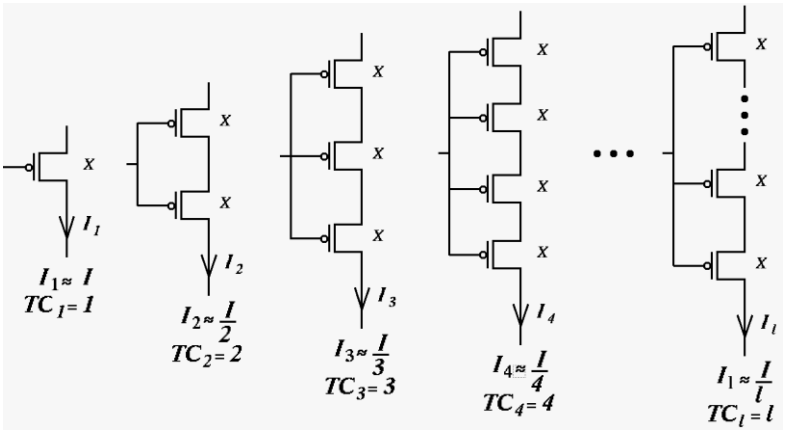


Figure 3.1. 1^{st}-order components that implement rational 1-weights with value $^1/_j$, for $1 \leq j \leq l$.

threshold weight component with value w_T^j/j. In the proposed method, the ILP assigns l different integer weights w_i^j for each input i, $1 \leq j \leq l$, but the value of each w_i^j is $\frac{1}{j} w_i^j$ when the TF is evaluated. When the TF is evaluated, the total value for input variable x_i is $\sum_{j=1}^{l} \frac{1}{j} w_i^j$. Likewise, the ILP assigns l different integer weights w_T^j for the threshold, $1 \leq j \leq l$. However, the value of each w_T^j is $\frac{1}{j} w_T^j$ when the TF is evaluated. The total value for the threshold weight w_T is $\sum_{j=1}^{l} \frac{1}{j} w_T^j$ when the TF is evaluated.

Based on the above, an n-input 1-TF $f(x_1, x_2, \ldots, x_n)$ is defined as

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & if \quad \sum_{i=1}^{n} ((\sum_{j=1}^{l} \frac{1}{j} w_i^j) \cdot x_i) \geq \sum_{j=1}^{l} \frac{1}{j} w_T^j \\ 0 & otherwise \end{cases}, \tag{3.3}$$

Although a 1-weight component with value $^w/_l$ requires a significant number of transistors, the flexibility of allowing rational weights allows the ILP to assign values to the respective variables so that the sum of the values assigned is much lower than the sum of values for the ILP that is restricted to integer weights. That way, the transistor count for

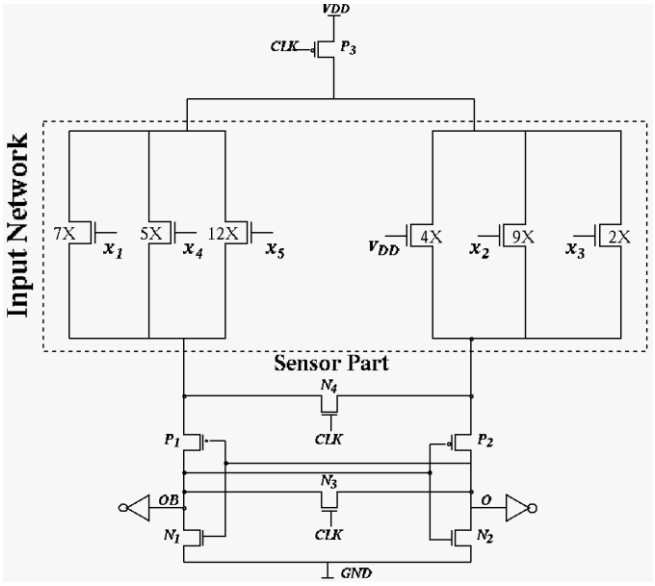
the CTG may be reduced.

Before we elaborate on the details in forming the ILP of a function, we show that we have identified 1-TFs whose transistor count is reduced when considering rational weights due to the flexibility in selecting the appropriate weight values. The following shows that a 1-TF with integer weights can be implemented with less number of transistors when considering rational weights when $l = 2$, i.e., non-integer weights that are multiples of 0.5. It also shows that the transistor count reduces further when $l = 4$, i.e., non-integer weights that are multiples of 0.25.

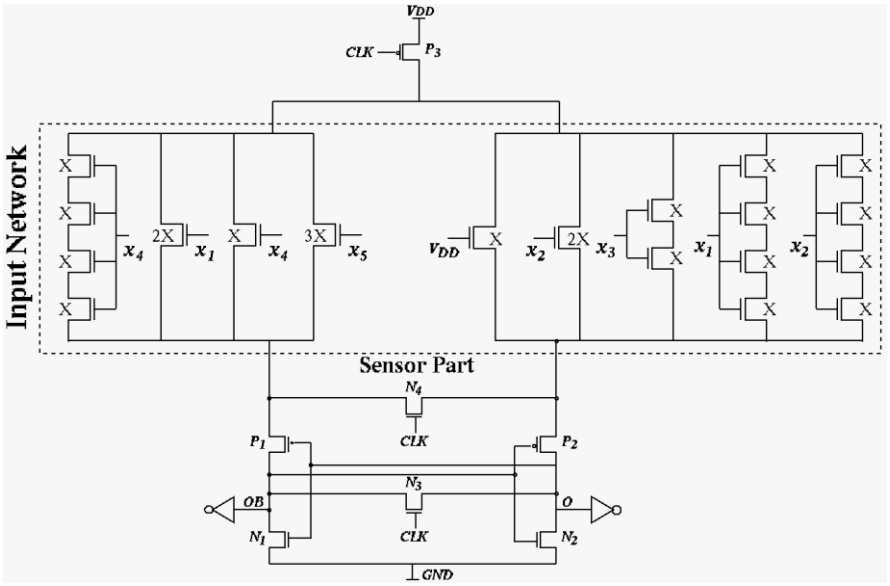
***Example 2*:** Consider the 5-variable function $F_2 = x_2x_5' + x_1'x_2x_4' + x_1'x_3x_5' + x_1'x_4'x_5'$. It is a TF with optimum integer weight configuration $w = [w_1, w_2, w_3, w_4, w_5; w_T] = [-7,9,2,-5,-12;-4]$ using the ILP in Section 3.2 and considering $C = 5\%$. The transistor count for implementing input weights is 39. However, when $l = 2$ (weights are multiple of 0.5), this function can be implemented as $w' = [w_1^1, w_2^1, w_3^1, w_4^1, w_4^2, w_5^1,\ w_5^2; w_T^1] = [-4,5,1,-2,-1,-6,-1;-2]$. The total transistor count reduces to 24. Moreover, when $l = 4$, weight set changes to $w'' = [w_1^1, w_1^4, w_2^1, w_2^4, w_3^2, w_4^1, w_4^4, w_5^1; w_T^1] = [-2,1,2,1,1,-1,-1,-3;-1]$, and the total transistor count reduces to 23.

Figure 3.2 shows the CTG implementation of F_2 with integer weights [53] and the proposed method when $l = 4$. In Figure 3.2, X denotes the size of a minimum width transistor to implement a unit integer 1-weight. □

The proposed ILP formulation is an extension of the one in [58] that was explained in Section 3.2. It has $2^n + l(n+1)$ constraints with $l(n+1)$ unknown variables. There is a constraint per input pattern to satisfy the functionality, and $l(n+1)$ constraints that determine the range of each variable. The weight at input i and the threshold weight are



(a)



(b)

Figure 3.2. The CTG implementation for function F_2 in example 3 when using (a) integer weights [53] (b) rational 1-weights with value $^{w}/_{j}$, for when $1 \leq j \leq l$ and $l = 4$.

$\sum_{j=1}^{l} \frac{1}{j} w_i^j$ and $\sum_{j=1}^{l} \frac{1}{j} w_T^j$, respectively, for some integer value j and predetermined l.

In addition, the ILP must minimize the transistor count considering that any rational 1-weight with value $^{w}/_{l}$ requires l times more transistors than any integer 1-weight with

value w. Therefore, the ILP must minimize quantity

$$\sum_{j=1}^{l} j w_T^j + \sum_{i=1}^{n} \left(\sum_{j=1}^{l} j w_i^j \right) \tag{3.4}$$

The example below illustrates the ILP-based approach to identify a 1-TF and assign optimum weights that are multiples of 0.5 ($l = 2$).

***Example 3*:** Consider UF $F_3 = x_1 + x_2 x_3$. Table 3.2 lists the ILP inequalities of F_3 based on the proposed ILP framework and 1-TF formulation in equation (3.3) considering $l = 2$. The last row shows the objective function of ILP-solver introduced in equation (3.4). For the set of constraints listed in Table 3.2, $w = [w_1^1,\ w_2^1,\ w_3^1,\ w_4^1;\ w_T^1,\ w_T^2] = [2,1,1;1,1]$ is an optimum solution for F_3. □

For non-positive UF f, in order to avoid implementing the negative weights, we

Table 3.2. The Truth Table and the ILP Constraints for UF F_3.
($0 < w_T^1 + 0.5 \cdot w_T^2 \iff 0 < 2 \cdot w_T^1 + w_T^2$)

Truth Table			Inequalities
Input Pattern $(x_1 x_2 x_3)$		F_3	
P_0	000	0	$0 < 2w_T^1 + w_T^2$
P_1	001	0	$2w_3^1 + w_3^2 < 2w_T^1 + w_T^2$
P_2	010	0	$2w_2^1 + w_2^2 < 2w_T^1 + w_T^2$
P_3	011	1	$2w_2^1 + w_2^2 + 2w_3^1 + w_3^2 > 2w_T^1 + w_T^2$
P_4	100	1	$2w_1^1 + w_1^2 > 2w_T^1 + w_T^2$
P_5	101	1	$2w_1^1 + w_1^2 + 2w_3^1 + w_3^2 > 2w_T^1 + w_T^2$
P_6	110	1	$2w_1^1 + w_1^2 + 2w_2^1 + w_2^2 > 2w_T^1 + w_T^2$
P_7	111	1	$2w_1^1 + w_1^2 + 2w_2^1 + w_2^2 + 2w_3^1 + w_3^2 > 2w_T^1 + w_T^2$
$\forall\ w_x^y \in \{w_T^1, w_1^1, w_2^1, w_3^1\}$: $0 \le w_x^y \le 10$			
$\forall\ w_x^y \in \{w_T^2, w_1^2, w_2^2, w_3^2\}$: $w_x^y \in \{0,1\}$			
minimize:			$w_T^1 + 2 \cdot w_T^2 + \sum_{i=1}^{3} (w_i^1 + 2 \cdot w_T^2)$

form the above mentioned ILP by using the method explained in Section 3.2.

3.4 Higher-order implementation of threshold functions using integer weights

A small fraction of binary functions are 1-TFs [50] and can be implemented as a single TG. In order to identify more threshold functions and increase the impact of TFs in digital synthesis, we consider the generalized k-TF definition described in equation (3.2). Preliminary results for the special case where $k = 2$ were presented in [83]. This section shows TF implementations using integer weights. The next section shows that the transistor count can be further reduced using rational weights.

A k-weight is implemented with k NMOS (or PMOS) transistors of the same size (according to the respective weight) which are connected in series, and the transistor gates are connected to k CTG inputs. The size of each transistor in a component that implements a k-weight with value w is set to $k \cdot w \cdot X$ to keep the active current of a unit k-weight equal to the active current of a unit 1-weight. Thus, the total transistor count of a k-weight component is k^2 times more than the transistor count of a 1-weight that implements the same weight value. This transistor count ratio is called the penalty factor and is taken into consideration to force the ILP-solver to find the minimum possible transistor count. Such gates are called k-CTGs. Figure 3.3 shows the k-weight components, their transistor count, and active current through them for $1 \leq k \leq 4$.

The approach is presented in two steps. First, we consider UFs for which the ILP does not use many variables. Then, we present an ILP for BFs. The latter requires more variables and is less scalable.

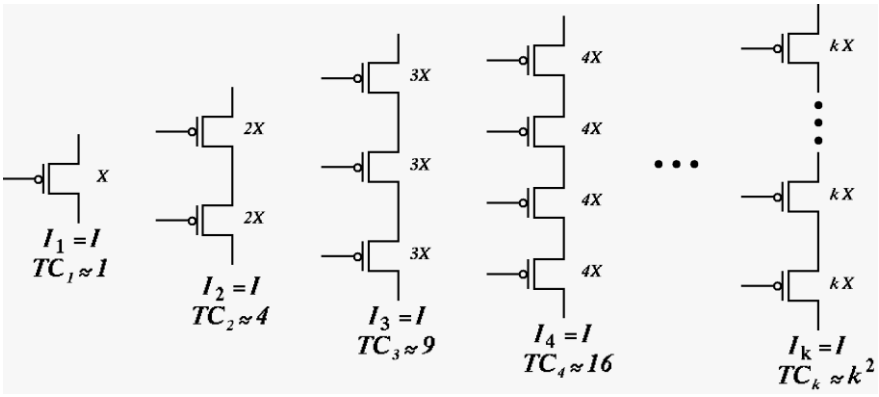


Figure 3.3. k-weight components for $1 \leq k \leq 4$.

An ILP formulation is presented to implement a UF as a k-CTG, as in equation (3.2) with minimum transistor count. The proposed ILP is an improvement of [60]. The ILP contains $2^n + n' + 1$ constraints with $n' + 1$ variables, where $n' = \sum_{m=1}^{k} \binom{n}{m}$ is the total number of m-weights, $1 \leq m \leq k$. There is a constraint per input pattern to satisfy the functionality, and $n' + 1$ constraints that bind the range of threshold and each input weight. Once a UF f is given, the Modified Chow's parameters [58] for all inputs and groups of inputs determine the negative weights. To form an efficient ILP, every product of m inputs $x_{i_1} \cdot x_{i_2} \cdot \ldots \cdot x_{i_m}$, $1 \leq m \leq k$, that activates an m-weight with a negative Modified Chow's parameter must be complemented. A 1-weight (m-weight when $m = 1$) with negative Modified Chow's parameter is activated as in 1-TF. A m-weight, $2 \leq m \leq k$, with negative Modified Chow's parameter will be activated when at least one of its input is set to 0. The ILP with the appropriate activation signals determines whether function f is a k-TF. The penalty factors appear as the coefficient of weights in the objective function of the ILP. The following objective function minimizes the k-CTG transistor count:

$$w_T + 1 \cdot \sum_{i_1=1}^{n} w_{i_1} + 4 \cdot \sum_{i_1=1}^{n-1} \sum_{i_2=2}^{n} w_{i_1,i_2} + \cdots + k^2 \cdot \sum_{i_1=1}^{n-k+1} \sum_{i_2=i_1+1}^{n-k} \ldots \sum_{i_k=i_{k-1}+1}^{n} w_{i_1,i_2,\ldots,i_k} \quad (3.5)$$

The weight configuration will then be assigned using the ILP solution and the negation property. The following examples illustrate the concept of k-TF and the ILP-based method to identify a k-TF.

***Example 4*:** Consider a 4-input UF $F_4 = x_1' + x_3x_2' + x_3x_4'$ with a set of all unknown weights $w = [w_1, w_2, w_3, w_4, w_{1,2}, w_{1,3}, w_{1,4}, w_{2,3}, w_{2,4}, w_{3,4}; w_T]$. The set of Modified Chow's parameters of either an input or a pair of inputs that activates a weight is $\vec{m}_{F_1} = (-5, -1, +3, -1, -9, -5, -9, -5, -7, -5)$. To form an efficient ILP, first every product term (activation signal) with negative m_i must be complemented. In this example, all inputs and pairs of inputs must be complemented except x_3.

Table 3.3 lists the inequalities of F_4 based on the 2-TF definition and by considering positive weights. For an input pattern, weight w (either 1-weight or 2-weight) appears in the inequality when its activation signal evaluates to 1. The objective function for a 2-TF is to minimize quantity $w_T + 1 \cdot \sum_{i=1}^{4} w_i + 4 \cdot \sum_{i=1}^{3} \sum_{j=2}^{4} w_{i,j}$. For the set of constraints listed in Table 3.3, an optimum solution is $w = [3, 0, 2, 0, 0, 0, 0, 1, 0, 1; 2]$. Using the negation property, $F_4 = [-3,0,2,0,0,0,0,-1,0,-1;-2]$. When considering $C = 5\%$ weight variation, the weight configuration becomes $w = [-6,0,4,0,0,0,0,-2,0,-2;-5]$. □

***Example 5*:** Function $F_5 = x_1x_2' + x_1'x_2 + x_1x_2x_3'$ is neither an 1-TF nor 2-TF. It is a 3-TF and the weight configuration to implement as a 3-CTG is $w = [w_1,\ w_2,\ w_{1,2,3};\ w_T] = [2\ ,2\ ,-4\ ;1]$ considering 5% weight variation ($C = 5\%$). □

The solutions for functions F_4 and F_5 in Examples 4 and 5 illustrate that many 1-weight, 2-weights, and 3-weights are assigned to zero. This means that a k-TF does not

Table 3.3. The Truth Table and ILP Constraints for F_4 Considering All Inputs and Pairs of Inputs Are Complemented Except x_3.

Truth Table			Inequalities
Input Pattern $(x_1x_2x_3x_4)$		F_4	
P_0	0000	1	$w_1 + w_2 + w_4 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,3} + w_{2,4} + w_{3,4} \geq w_T$
P_1	0001	1	$w_1 + w_2 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,3} + w_{2,4} + w_{3,4} \geq w_T$
P_2	0010	1	$w_1 + w_2 + w_3 + w_4 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,3} + w_{2,4} + w_{3,4} \geq w_T$
P_3	0011	1	$w_1 + w_2 + w_3 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,3} + w_{2,4} \geq w_T$
P_4	0100	1	$w_1 + w_4 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,3} + w_{2,4} + w_{3,4} \geq w_T$
P_5	0101	1	$w_1 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,3} + w_{3,4} \geq w_T$
P_6	0110	1	$w_1 + w_3 + w_4 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,4} + w_{3,4} \geq w_T$
P_7	0111	1	$w_1 + w_3 + w_{1,2} + w_{1,3} + w_{1,4} \geq w_T$
P_8	1000	0	$w_2 + w_4 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,3} + w_{2,4} + w_{3,4} < w_T$
P_9	1001	0	$w_2 + w_{1,2} + w_{1,3} + w_{2,3} + w_{2,4} + w_{3,4} < w_T$
P_{10}	1010	1	$w_2 + w_3 + w_4 + w_{1,2} + w_{1,4} + w_{2,3} + w_{2,4} + w_{3,4} \geq w_T$
P_{11}	1011	1	$w_2 + w_3 + w_{1,2} + w_{2,3} + w_{2,4} \geq w_T$
P_{12}	1100	0	$w_4 + w_{1,3} + w_{1,4} + w_{2,3} + w_{2,4} + w_{3,4} < w_T$
P_{13}	1101	0	$w_{1,3} + w_{2,3} + w_{3,4} < w_T$
P_{14}	1110	1	$w_3 + w_4 + w_{1,4} + w_{2,4} + w_{3,4} \geq w_T$
P_{15}	1111	0	$w_3 < w_T$
minimize:			$w_T + 1 \cdot \sum_{i=1}^{4} w_i + 4 \cdot \sum_{i=1}^{3} \sum_{j=i+1}^{4} w_{i,j}$

necessarily need all k-weight components when implementing as a k-CTG. Moreover, the transistor count, and hence, the hardware requirement of many existing threshold functions (1-TFs) is reduced using higher order components. The following show that a 1-TF can be implemented as proposed 2-CTG with lower cost than the respective 1-CTG.

***Example 6*:** Consider the 4-variable function $F_6 = x_4x_3 + x_3x_2 + x_3x_1 + x_2x_1$. It is a 1-TF with optimum weight configuration $w = [w_1, w_2, w_3, w_4; w_T] = [4,4,6,2; 7]$ using the ILP in [58]. The transistor count or the total sum of threshold and input weights of 1-CTG is 23. Let X denote the area of a unit 1-weight transistor. The total area of the input

networks is $23 \cdot X$.

However, this function can be implemented as a 2-CTG with weight configuration $w = [w_1, w_2, w_3, w_4, w_{1,2}, w_{1,3}, w_{1,4}, w_{2,3}, w_{2,4}, w_{3,4}; w_T] = [2, 2, 2, 0, 0, 0, 0, 0, 0, 2; 3]$. Each 2-weight requires 4 more times the transistor count of a 1-weight that implements the same weight. The transistor count reduces to $(2 + 2 + 2) + 4 \cdot (2) + (3) = 17$, and thus, the total area of the input networks of the 2-CTG reduces to $17 \cdot X$.

Figure 3.4 shows the 1-CTG [53] and proposed 2-CTG implementations of F_6 and the size of transistors that implements 1-weights and 2-weights considering X as the size of a minimum width transistor to implement a unit 1-weight. The length of all the PMOS and NMOS transistors is the same and determined by the used technology. □

The remaining of the section considers BFs. The correlation between the sign of the Modified Chow's parameters and the sign of weights only holds for UFs. The ILP for

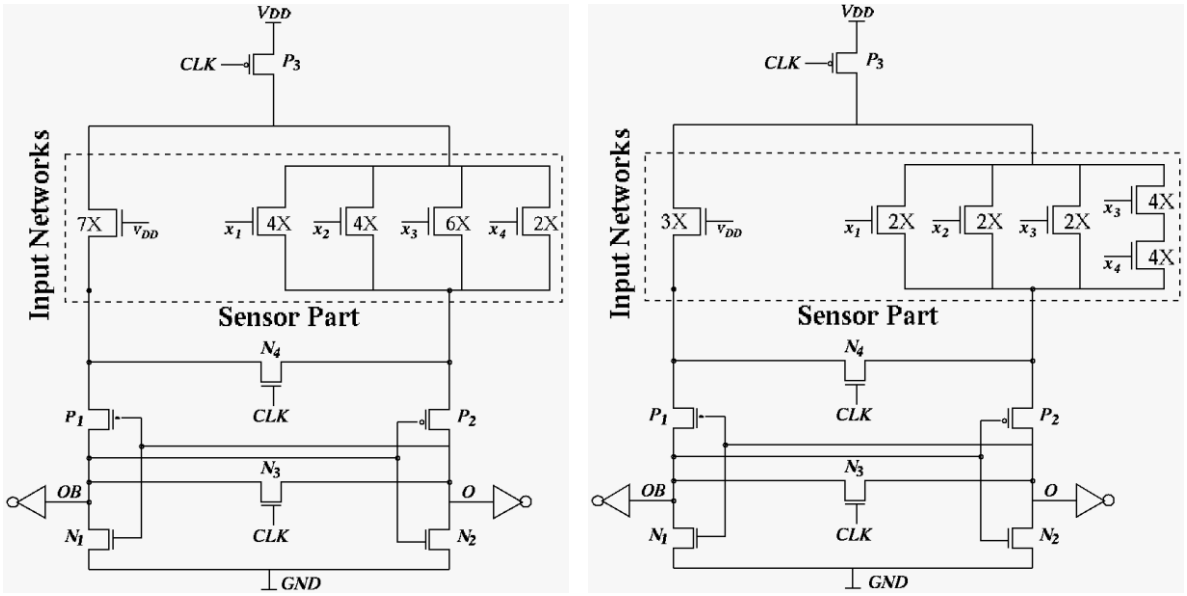


Figure 3.4. The CTG implementation for function $F_6 = x_4x_3 + x_3x_2 + x_3x_1 + x_2x_1$ with (a) 1-CTG as 1-TF [53] (b) proposed 2-CTG as 2-TF [82].

a BF works as in [60] and contains $2^n + 3(n + n' + 1)$ constraints with $3(n + n' + 1)$ variables, where $n' = \binom{n}{2}$ is the total number of 2-weights. Each weight can be either positive or negative. The objective function is to minimize quantity

$$|w_T| + 1 \cdot \sum_{i_1=1}^{n} |w_{i_1}| + 4 \cdot \sum_{i_1=1}^{n-1} \sum_{i_2=2}^{n} |w_{i_1,i_2}| + \cdots + k^2 \cdot \sum_{i_1=1}^{n-k+1} \sum_{i_2=i_1+1}^{n-k} \ldots \sum_{i_k=i_{k-1}+1}^{n} |w_{i_1,i_2,\ldots,i_k}| \quad (3.6)$$

where $|w_x|$, denotes the absolute value for each weight w_x, and $w_x \in \{w_T, w_i, w_{i,j}\}$. Let y_x be a binary variable, and U denote a predetermined upper bound of $|w_x|$ for each weight w_x. For each w_x, two variables ${w_x}^+$ and ${w_x}^-$ are used, and the bound on the absolute value w_x is enforced using the following constraints:

$$\begin{cases} 0 \le {w_x}^+ \le U \cdot y_x \\ 0 \le {w_x}^- \le U \cdot (1 - y_x) \\ y_x \in \{0,1\} \end{cases} \quad (3.7)$$

Then $w_x = {w_x}^+ - {w_x}^-$. In addition, for CTG the ILP should minimize quantity

$$ {w_T}^+ + {w_T}^- + \sum_{i_1=1}^{n} ({w_{i_1}}^+ + {w_{i_1}}^-) + 4 \cdot \sum_{i_1=1}^{n-1} \sum_{i_2=2}^{n} ({w_{i_1,i_2}}^+ + {w_{i_1,i_2}}^-) + \cdots + $$

$$ k^2 \cdot \sum_{i_1=1}^{n-k+1} \sum_{i_2=i_1+1}^{n-k} \ldots \sum_{i_k=i_{k-1}+1}^{n} ({w_{i_1,i_2,\ldots,i_k}}^+ + {w_{i_1,i_2,\ldots,i_k}}^-) \quad (3.8)$$

The example below illustrates the ILP-based approach to identify a BF as a 2-TF and assign optimum weights to implement 2-CTG with minimum possible transistor count.

***Example 7*:** Consider BF $F_7 = x_1 x_3' + x_2 x_3$. Table 3.4 lists the ILP inequalities of F_7 based on the k-TF formulation in equation (3.2) when $k = 2$. The last two rows show the constraints and the objective function of ILP-solver introduced in equations (3.7) and (3.8) to assign negative weights and minimize the sum of weights. For the set of constraints listed in Table 3.4, $w = [w_1, w_2, w_3, w_{1,2}, w_{1,3}, w_{2,3}; w_T] = [2,0,0,0,-2,2;1]$ is an

Table 3.4. The Truth Table and the ILP Constraints for BF F_7.

Truth Table			Inequalities
Input Pattern $(x_1x_2x_3)$		F_7	
P_0	000	0	$0 < {w_T}^+ - {w_T}^-$
P_1	001	0	${w_3}^+ - {w_3}^- < {w_T}^+ - {w_T}^-$
P_2	010	0	${w_2}^+ - {w_2}^- < {w_T}^+ - {w_T}^-$
P_3	011	1	${w_3}^+ - {w_3}^- + {w_2}^+ - {w_2}^- + {w_{2,3}}^+ - {w_{2,3}}^- \geq {w_T}^+ - {w_T}^-$
P_4	100	1	${w_1}^+ - {w_1}^- \geq {w_T}^+ - {w_T}^-$
P_5	101	0	${w_3}^+ - {w_3}^- + {w_1}^+ - {w_1}^- + {w_{1,3}}^+ - {w_{1,3}}^- < {w_T}^+ - {w_T}^-$
P_6	110	1	${w_2}^+ - {w_2}^- + {w_1}^+ - {w_1}^- + {w_{1,2}}^+ - {w_{1,2}}^- \geq {w_T}^+ - {w_T}^-$
P_7	111	1	${w_3}^+ - {w_3}^- + {w_2}^+ - {w_2}^- + {w_1}^+ - {w_1}^- + {w_{2,3}}^+ - {w_{2,3}}^- + {w_{1,3}}^+ - {w_{1,3}}^- + {w_{1,2}}^+ - {w_{1,2}}^- \geq {w_T}^+ - {w_T}^-$
$\forall\, {w_x}^+, {w_x}^- \in \{{w_T}^+, {w_T}^-, {w_1}^+, {w_1}^-, {w_2}^+, {w_2}^-, {w_3}^+, {w_3}^-, {w_{1,2}}^+, {w_{1,2}}^-, {w_{1,3}}^+, {w_{1,3}}^-, {w_{2,3}}^+, {w_{2,3}}^-\}$:			$\begin{cases} 0 \leq {w_x}^+ \leq U \cdot y_x \\ 0 \leq {w_x}^- \leq U \cdot (1 - y_x) \\ y_x \in \{0,1\} \end{cases}$
minimize:			${w_T}^+ + {w_T}^- + \sum_{i=1}^{3}({w_i}^+ + {w_i}^-) + 4 \times \sum_{i=1}^{2}\sum_{j=i+1}^{3}({w_{i,j}}^+ + {w_{i,j}}^-)$

optimum solution for F_7 when considering 5% weight variation. □

3.5 Efficient design of higher-order threshold functions using rational weights

This section applies the method of Section 3.3 on high order TFs so that their implementation has less transistor count because of rational weight assignment.

In k-TFs, each weight component may be controlled by more than one input. When $l \leq k$, a k-weight with value w/l is implemented with k transistors connected in series each with size $\frac{w \cdot k}{l} \cdot X$. In this case, the transistor count is $\frac{w}{l} \cdot k^2$ times the transistor count of a unit integer 1-weight component.

For the case when $l > k$, a k-weight can be implemented with l transistors connected in series each with size $w \cdot X$. In this case, the transistor count of this component is $l \cdot w$ times the transistor count of a unit integer 1-weight component.

The ILP will select the implementation with the smallest penalty factor in order to find the minimum possible transistor count. Figure 3.5 considers $l = 5$, and shows the k-weights for $k = 2$ and $k = 3$. In particular, it shows all minimum penalty factor components for $k = 2$ and $1 < j \leq 5$ as well as for $k = 3$ and $1 < j \leq 5$.

Let I denote the active current through a minimum size transistor that implements a unit integer 1-weight. The active current through a rational k-weight component with value w/l is $\frac{w}{l} \cdot I$.

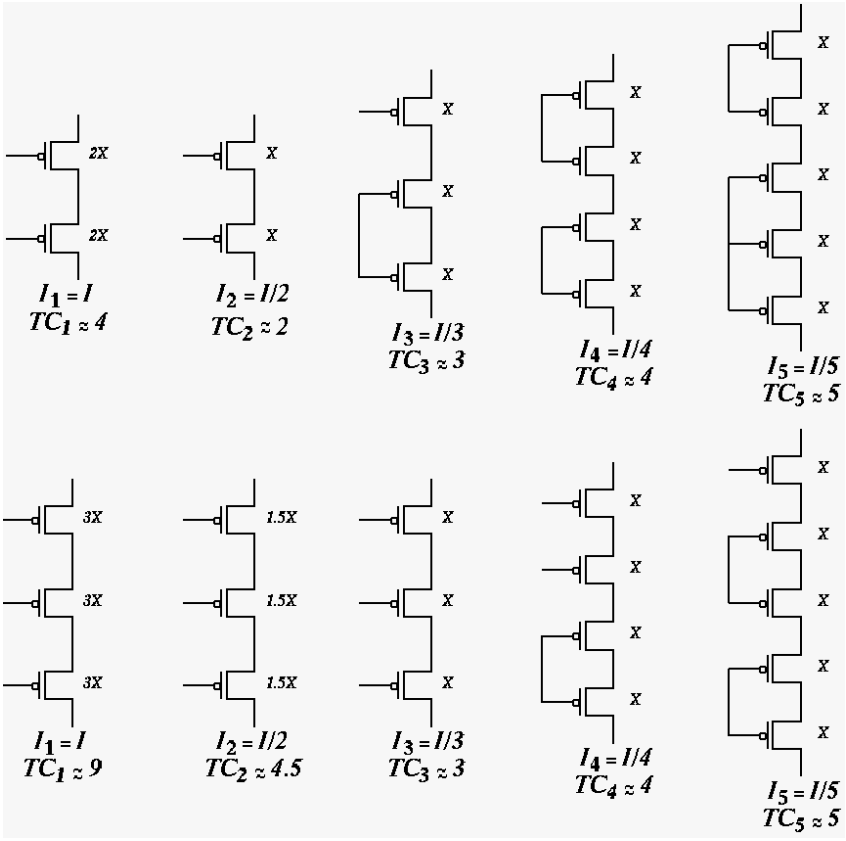


Figure 3.5. Rational k-weights components with values $1/j$ for $2 \leq k \leq 3$ and $1 \leq j \leq 5$.

Let $w^1_{i_1,i_2,\ldots,i_m}$ be an integer value for a weight component that is activated by m inputs $i_1, i_2, \ldots$, and i_m. Let also $w^j_{i_1,i_2,\ldots,i_m} \in \{0,1,\ldots,j-1\}$, $2 < j \leq l$, denote a rational m-weight with value $w^j_{i_1,i_2,\ldots,i_m}/j$ corresponding to the group of m inputs $i_1, i_2, \ldots$, and i_m, and $1 \leq m \leq k$. The m-weight $w_{i_1,i_2,\ldots,i_m}$ is represented by $\sum_{j=1}^{l} \frac{1}{j} w^j_{i_1,i_2,\ldots,i_m}$ in the definition of the TF. Based on the above,

$$f(x_1, x_2, \cdots, x_n) =$$

$$\begin{cases} 1 & if \quad \sum_{i_1=1}^{n} \left(\left(\sum_{j=1}^{l} \frac{1}{j} w^j_{i_1} \right) \cdot x_{i_1} \right) + \cdots + \\ & \sum_{i_1=1}^{n-k+1} \sum_{i_2=i_1+1}^{n-k} \ldots \sum_{i_k=i_{k-1}+1}^{n} \left(\left(\sum_{j=1}^{l} \frac{1}{j} w^j_{i_1,i_2,\ldots,i_k} \right) \cdot x_{i_1} x_{i_2} \ldots x_{i_k} \right) \geq \sum_{j=1}^{l} \frac{1}{j} w^j_T \\ 0 & otherwise \end{cases}, \quad (3.9)$$

where x_{i_1}, $i_1 = 1, \ldots, n$, are binary input variables, j, $w^j_{i_1,i_2,\ldots,i_m}$, and w^j_T are integer values, and w^j_T is the threshold weight with value w^j_T/j.

Before we elaborate on the details in forming the ILP of a function, we show that we have identified k-TFs whose transistor count is reduced when considering rational weights due to the flexibility in selecting the appropriate weight values.

***Example 8*:** Consider again function F_5 in Example 5. The weight configuration changes to $w = [w^1_1,\ w^2_1,\ w^1_2,\ w^2_2,\ w^1_{3,4,5}; w^1_T] = [1,1,1,1,1;2]$ with transistor count 17 and $w = [w^1_1, w^1_2, w^3_{3,4,5}; w^1_T, w^3_T] = [1,1,2;1,1]$ with transistor count 12 when l is 2 and 3, respectively. The transistor count may decrease when l increases. □

When comparing to a minimum size transistor as a unit integer 1-weight, the component that implements any rational k-weight with value w/l, for $k > 1$ or $l > 1$,

imposes more transistor count. Therefore, an effective ILP-based framework is needed to identify a TF and assign appropriate weights using minimum possible number of non-integer higher order weights.

The ILP formulations to identify and implement either a UF or a BF as a k-CTG with minimum transistor count is an extension of the formulations presented in Section 3.4. In particular, we start with UF, and then we present the ILP for BF.

The ILP formulation to identify and implement a UF as a k-CTG using non-integer weights has $2^n + l(n' + 1)$ constraints with $l(n' + 1)$ variables, where $n' = \sum_{i=1}^{k} \binom{n}{i}$ is the total number of m-weights, $1 \leq m \leq k$. The Modified Chow's parameters of all groups of m inputs determine the negative weights (including all m-weights). To form an efficient ILP, every product of m inputs ($x_{i_1} \cdot x_{i_2} \cdots x_{i_m}$), that activates a m-weight, with a negative Modified Chow's parameter must be complemented. An m-weight with negative Modified Chow's parameter will be activated when at least one of x_{i_1}, x_{i_2}, and x_{i_m} is set to 0. The ILP with the appropriate activation signals determines whether function f is a k-TF.

For each input pattern, any m-weight $w_{i_1,i_2,\ldots,i_m}$, $1 \leq m \leq k$, in Section 3.4 is substituted with l unknown variables so that $w_{i_1,i_2,\ldots,i_m} = \sum_{j=1}^{l} \frac{1}{j} w^{j}_{i_1,i_2,\ldots,i_m}$. Likewise, the threshold weight is replaced by equation (3.5). In addition, the ILP must minimize the transistor count considering that any m-weight with value w/j requires $j \cdot f(j-k) + \frac{m^2}{j} \cdot f(k-j)$ multiplied by the minimum size transistor that implements a unit integer 1-weight, where $f(t)$ is the unit step function that evaluates to 1 when $t \geq 0$, and evaluates to 0 when $t < 0$. Therefore, the ILP must minimize quantity

$$\sum_{j=1}^{l} j \cdot w_T^j + \sum_{i_1=1}^{n} \left(\sum_{j=1}^{l} j \cdot w_{i_1}^j \right) + \cdots +$$

$$+ \sum_{i_1=1}^{n-k+1} \sum_{i_2=i_1+1}^{n-k} \ldots \sum_{i_k=i_{k-1}+1}^{n} \left(\sum_{j=1}^{l} \left(j \cdot f(j-k) + \frac{k^2}{j} \cdot f(k-j) \right) \cdot w_{i_1,i_2,\ldots,i_k}^j \right) \tag{3.10}$$

The weight configuration will then be assigned using the solution from ILP and the negation property. The example below illustrates the ILP-based approach to identify a 2-TF and assign optimum half integer weights ($l = 2$).

***Example 9*:** Consider again the UF F_4 in Example 4 with a set of non-zero integer weights $w = [w_1, w_3, w_{2,3}, w_{3,4}; w_T] = [-6,4,-2,-2;-5]$. Let each input weight variable in weight set w is replaced by $\sum_{j=1}^{2} \frac{1}{j} w_{i_1,i_2,\ldots,i_m}^j$. Likewise, the threshold variable w_T is replaced by $\sum_{j=1}^{2} \frac{1}{j} w_T^j$. Table 3.5 lists the inequalities of F_4. The last row shows the objective function of ILP-solver introduced in equation (3.10). For the set of constraints listed in Table 3.5, $w = [w_1^1,\ w_2^1,\ w_3^1,\ w_4^1;\ w_T^1,\ w_T^2] = [2,1,1;1,1]$ is an optimum solution for F_4. □

Many of the constraints in the ILP are redundant. We use the simplification method which is the extension of the one in [70] to eliminate redundant constraints which makes the ILP formulation smaller and possibly faster to solve. As an example consider the UF in Example 9. If $2w_1^1 + w_1^2 > 2w_T^1 + w_T^2$, any constraint containing $2w_1^1 + w_1^2$ must be greater than $2w_T^1 + w_T^2$. Therefore, the last 3 constraints are redundant and can be removed from ILP.

The correlation between the sign of the Modified Chow's parameters and the sign

Table 3.5. The Truth Table and the ILP Constraints for UF F_4.
$(0 < w_T^1 + 0.5 \cdot w_T^2 \iff 0 < 2 \cdot w_T^1 + w_T^2)$

Truth Table			Inequalities
Input Pattern $(x_1x_2x_3)$		F_4	
P_0	000	0	$0 < 2w_T^1 + w_T^2$
P_1	001	0	$2w_3^1 + w_3^2 < 2w_T^1 + w_T^2$
P_2	010	0	$2w_2^1 + w_2^2 < 2w_T^1 + w_T^2$
P_3	011	1	$2w_2^1 + w_2^2 + 2w_3^1 + w_3^2 + 2w_{2,3}^1 + w_{2,3}^2 > 2w_T^1 + w_T^2$
P_4	100	1	$2w_1^1 + w_1^2 > 2w_T^1 + w_T^2$
P_5	101	1	$2w_1^1 + w_1^2 + 2w_3^1 + w_3^2 + 2w_{1,3}^1 + w_{1,3}^2 > 2w_T^1 + w_T^2$
P_6	110	1	$2w_1^1 + w_1^2 + 2w_2^1 + w_2^2 + 2w_{1,2}^1 + w_{1,2}^2 > 2w_T^1 + w_T^2$
P_7	111	1	$2w_1^1 + w_1^2 + 2w_2^1 + w_2^2 + 2w_3^1 + w_3^2 + 2w_{1,2}^1 + w_{1,2}^2 + 2w_{1,3}^1 + w_{1,3}^2 + 2w_{2,3}^1 + w_{2,3}^2 > 2w_T^1 + w_T^2$
minimize:			$w_T^1 + 2 \cdot w_T^2 + \sum_{i=1}^{n} (w_i^1 + 2 \cdot w_i^2) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(4 \cdot w_{i,j}^1 + 2 \cdot w_{i,j}^2\right)$

of weights only holds for UFs. The ILP formulation to identify and implement a BF as a k-CTG using non-integer weights contains 3 times more unknown variables that the one for a UF. Each weight can be either positive or negative. The objective function is to minimize the sum of absolute value of variables that are implemented using equation (3.7). In addition, for k-CTG the ILP should minimize quantity

$$\sum_{j=1}^{l} j(w_T^{j^+} + w_T^{j^-}) + \sum_{i_1=1}^{n} \left(\sum_{j=1}^{l} j \left(w_{i_1}^{j^+} + w_{i_1}^{j^-} \right) \right) + \cdots$$
$$+ \sum_{i_1=1}^{n-k+1} \sum_{i_2=i_1+1}^{n-k} \dots \sum_{i_k=i_{k-1}+1}^{n} \left(\sum_{j=1}^{l} \left(j \cdot f(j-k) + \frac{k^2}{j} \cdot f(k-j) \right) \left(w_{i_1,i_2,\dots,i_k}^{j^+} + w_{i_1,i_2,\dots,i_k}^{j^-} \right) \right) \quad (3.11)$$

The simplification method in [70] is not extendable to BFs. Therefore, the ILP for a BF slower than the one for a UF. The example below illustrates the ILP-based approach to identify a BF as a 2-TF and assign optimum integer and non-integer weights to

implement efficient 2-CTG.

***Example 10*:** Consider again the BF F_7 in Example 7 with set of integer weight values $w = [w_1, w_{1,3}, w_{2,3}; w_T] = [2, -2, 2; 1]$ and transistor count 19. Table 3.6 lists the ILP inequalities of F_7 based on the k-TF formulation described in equation (3.9). For simplicity, we consider $k = 2$ and $l = 2$. The last two rows show the ILP constraints and the ILP objective function of introduced in equations (3.7) and (3.11) to assign negative weights and minimize the sum of weights. For the set of constraints listed in Table 3.6,

Table 3.6. The Truth Table and the ILP Constraints for BF F_7.

Truth Table			Inequalities
Input Pattern $(x_1x_2x_3)$		F_7	
P_0	000	0	$0 < 2w_T^{1^+} - 2w_T^{1^-} + w_T^{2^+} - w_T^{2^-}$
P_1	001	0	$2w_3^{1^+} - 2w_3^{1^-} + w_3^{2^+} - w_3^{2^-} < 2w_T^{1^+} - 2w_T^{1^-} + w_T^{2^+} - w_T^{2^-}$
P_2	010	0	$2w_2^{1^+} - 2w_2^{1^-} + w_2^{2^+} - w_2^{2^-} < 2w_T^{1^+} - 2w_T^{1^-} + w_T^{2^+} - w_T^{2^-}$
P_3	011	1	$+2w_3^{1^+} - 2w_3^{1^-} + w_3^{2^+} - w_3^{2^-} + 2w_2^{1^+} - 2w_2^{1^-} + w_2^{2^+} - w_2^{2^-}$ $+2w_{2,3}^{1\ ^+} - 2w_{2,3}^{1\ ^-} + w_{2,3}^{2\ ^+} - w_{2,3}^{2\ ^-} \geq 2w_T^{1^+} - 2w_T^{1^-} + w_T^{2^+} - w_T^{2^-}$
P_4	100	1	$2w_1^{1^+} - 2w_1^{1^-} + w_1^{2^+} - w_1^{2^-} \geq 2w_T^{1^+} - 2w_T^{1^-} + w_T^{2^+} - w_T^{2^-}$
P_5	101	0	$2w_1^{1^+} - 2w_1^{1^-} + w_1^{2^+} - w_1^{2^-} + 2w_3^{1^+} - 2w_3^{1^-} + w_3^{2^+} - w_3^{2^-}$ $+2w_{1,3}^{1\ ^+} - 2w_{1,3}^{1\ ^-} + w_{1,3}^{2\ ^+} - w_{1,3}^{2\ ^-} < 2w_T^{1^+} - 2w_T^{1^-} + w_T^{2^+} - w_T^{2^-}$
P_6	110	1	$2w_1^{1^+} - 2w_1^{1^-} + w_1^{2^+} - w_1^{2^-} + 2w_2^{1^+} - 2w_2^{1^-} + w_2^{2^+} - w_2^{2^-}$ $+2w_{1,2}^{1\ ^+} - 2w_{1,2}^{1\ ^-} + w_{1,2}^{2\ ^+} - w_{1,2}^{2\ ^-} \geq 2w_T^{1^+} - 2w_T^{1^-} + w_T^{2^+} - w_T^{2^-}$
P_7	111	1	$2w_1^{1^+} - 2w_1^{1^-} + w_1^{2^+} - w_1^{2^-} + 2w_2^{1^+} - 2w_2^{1^-} + w_2^{2^+} - w_2^{2^-} + 2w_3^{1^+}$ $-2w_3^{1^-} + w_3^{2^+} - w_3^{2^-} + 2w_{1,2}^{1\ ^+} - 2w_{1,2}^{1\ ^-} + w_{1,2}^{2\ ^+} - w_{1,2}^{2\ ^-} + 2w_{1,3}^{1\ ^+}$ $-2w_{1,3}^{1\ ^-} + w_{1,3}^{2\ ^+} - w_{1,3}^{2\ ^-} + 2w_{2,3}^{1\ ^+} - 2w_{2,3}^{1\ ^-} + w_{2,3}^{2\ ^+} - w_{2,3}^{2\ ^-}$ $\geq 2w_T^{1^+} - 2w_T^{1^-} + w_T^{2^+} - w_T^{2^-}$
$\forall\, w_x^{\mp} \in \{w_T^{1\mp}, w_T^{2\mp}, w_1^{1\mp}, w_1^{2\mp}, w_2^{1\mp}, w_2^{2\mp}, w_3^{1\mp}, w_3^{2\mp} w_{1,2}^{1\ \mp}, w_{1,2}^{2\ \mp}, w_{1,3}^{1\ \mp}, w_{1,3}^{2\ \mp}, w_{2,3}^{1\ \mp}, w_{2,3}^{2\ \mp}\}$:			$\begin{cases} 0 \leq w_x^{\ +} \leq U \cdot y_x \\ 0 \leq w_x^{\ -} \leq U \cdot (1 - y_x) \\ y_x \in \{0,1\} \end{cases}$
minimize:			$w_T^{1^+} + w_T^{1^-} + 2w_T^{2^+} + 2w_T^{2^-} + \sum_{i=1}^{3}\sum_{j=1}^{2} j \cdot (w_i^{j\ ^+} + w_i^{j\ ^-}) +$ $\sum_{a=1}^{2}\sum_{b=a+1}^{3}\sum_{j=1}^{2} (j \cdot f(j-2) + \frac{4}{j} \cdot f(2-j)) \cdot (w_{a,b}^{j\ \ ^+} + w_{a,b}^{j\ \ ^-})$

$w = [w_1^1,\ w_1^2, w_{1,3}^1, w_{1,3}^2, w_{2,3}^1, w_{2,3}^2; w_T^1,\ w_T^2] = [1, 0,\ -1, 0, 1, 0; 0, 1]$ is an optimum solution for F_7. The transistor count reduces to 11. □

3.6 Experimental results

The proposed ILP-based approach has been implemented in the C++ language on an Intel Xenon 2.4GHz with 8GB memory. To evaluate its impact, we examined non-scalable functions with up to fifteen inputs. An n-input non-scalable function is a function that requires non-empty levels of variables in the Binary Decision Diagram (BDD) representation for some ordering of the variables [83]. In another word, in a non-scalable function, no input variable is don't care, and, therefore, all variables (and/or their complements) appear in the minimum sum-of-product expression of the function.

Table 3.7 presents non-scalable n-input k-TFs using rational k-weights with value w/l for different n, k, and l that were derived using the ILP of Section 3.5. For each value of n we found the 1-TFs with maximum transistor count. This was set as a bound to the objective function for any ILP formulation for $l \geq 4$ and $k \geq 4$. The first column in Table 3.7 shows the number of inputs (value of n). The goal in this paper is to provide an indication of the percentage of all n-input functions that benefit from the proposed method. When n is large it is impossible to examine all functions and, therefore, functions are selected randomly. For functions with $n \geq 4$, the entries in Table 3.7 were obtained by sampling randomly 100 thousand functions. For $n \geq 4$, the 2^n bit output vector of an n-input function was filled with either 0 or 1 at randomly selected positions (determined by randomly selecting an integer mod 2^n), and so that the number of ones in the function obeyed the distribution of functions based on this property. (For example, the number of

5-input functions with 16 ones in the output bit vector is approximately 10 times more than the number of 5-input functions with 10 ones.) In order for the experiment to have more statistical significance, we only considered non-scalable functions, and when a function is generated we applied the procedure described earlier in this section to determine that it is non-scalable. (It is asserted that the distribution of non-scalable n-input functions based on the number of ones in their output bit vector is the same as the one described earlier for n-input functions.)

The second column in Table 3.7 lists the number of 1-TFs identified by using the existing ILP-based method in [58] considering integer weights. These functions are implementable with existing CTGs. The third column shows the examined values of l, $l \in \{1, 2, 3, 4\}$. The fourth column shows the number of 2-TFs that do not have transistor count higher than any of the 1-TF in column two. The fifth column shows the percentage increase over the number of 1-TFs in column two. Columns six to nine show similar results for $k \in \{3, 4\}$. For all examined functions, the value of C was set to 11% to take into consideration that weights may vary primarily due to aging. This value for C was obtained by performing SPICE simulations on a single transistor that implements a unit integer 1-weight in corner cases using 45nm technology [76] while the transistor was continuously under stress (worst-case aging scenario). We used the static aging model in [84] and we found that the transistor threshold voltage shifted by $50mV$ under continuous stress. This increase in threshold voltage amounted to 11% decrease in the current.

The results in Table 3.7 show that for higher value of k and l, the ILP has more flexibility to assign weights so that the total transistor count decreases. Therefore, when k and l increase more functions can be implemented as current mode gates using a

transistor count similar to that for the significantly less 1-TFs that were implemented as 1-CTG. In particular, when $k = 4$ and $l = 4$, about 24.9 times more functions can be implemented as CTGs with similar or less transistor count.

Table 3.8 lists the average execution time by the proposed ILP method to determine whether an examined n-input function (BF and UF) was implementable as proposed k-TF described in equation (3.9) for different values of k and l, and C set to 11%. In our experimental evaluation, we set up an execution time upper bound of 60 second per TF, at which point the function was aborted. Character "-" indicates that no results were obtained due to violation of the execution time upper bound. Observe, however, that the approach can handle all the UFs with at most 12 inputs. They can be implemented to up to the 4^{th} order when considering rational weight with value up to 4. These results show that the average execution time increases as the values of n, k and l increases. This is due to the increase in the number of unknown variables in the ILP. For higher input functions (functions with more fan-ins), heuristic approaches as in [61-63] can be used to implement UFs. However, they will not ensure that all TFs can be identified, and the weight configuration of the identified TFs is not necessarily optimum. Furthermore, they do not apply to BFs.

Table 3.9 lists the number of 1-TFs identified by using the existing ILP-based method in [58] considering integer weights. Let Δ denote the percentage reduction in CTG transistor count. Columns three to eight show the number of TFs listed in second column that were implemented with less transistor count when considering the proposed k-CTG described in equation (3.9) for $k, l \leq 4$ non-integer weights. These columns were generated based on different ranges of Δ, and C set to 11%.

Table 3.7. The Number of k-CTGs with Rational k-weights of Value w/l Whose Transistor Count is no More Than 1-CTGS with Integer Weights.

n	1-TF	l	$k=2$	INCREASE RATIO	$k=3$	INCREASE RATIO	$k=4$	INCREASE RATIO
1	2	1	2	1	2	1	2	1
		2	2	1	2	1	2	1
		3	2	1	2	1	2	1
		4	2	1	2	1	2	1
2	8	1	8	1	10	1.25	10	1.25
		2	10	1.25	10	1.25	10	1.25
		3	10	1.25	10	1.25	10	1.25
		4	10	1.25	10	1.25	10	1.25
3	72	1	72	1	72	1	72	1
		2	188	2.61	192	2.66	192	2.66
		3	214	2.97	214	2.97	214	2.97
		4	216	3	218	3.02	218	3.02
4	1536	1	2566	1.7	4454	2.9	4761	3.1
		2	9012	5.87`	11059	7.2	11366	7.4
		3	9872	6.43	11063	7.2	12748	8.3
		4	11464	7.46	21043	13.7	21196	13.8
5*	367	1	4514	12.3	8110	22.1	9395	25.6
		2	7743	21.1	10202	27.8	10716	29.2
		3	7964	21.7	10312	28.1	11046	30.1
		4	8514	23.2	11450	31.2	12698	34.6
6*	105	1	1176	11.2	1827	17.4	1848	17.6
		2	1659	15.8	2467	23.5	2740	26.1
		3	1690	16.1	2478	23.6	2772	26.4
		4	1827	17.4	2887	27.5	3318	31.6
7*	89	1	1593	17.9	1877	21.1	1993	22.4
		2	2607	29.3	2919	32.8	3262	36.7
		3	2776	31.2	3017	33.9	3271	36.7
		4	2860	36.2	3506	39.4	3871	43.5
8*	66	1	1498	22.7	3438	52.1	3517	53.3
		2	3095	46.9	4468	67.7	4659	70.6
		3	3590	54.4	4481	67.9	4699	71.2
		4	3973	60.2	4765	72.2	4870	73.8
9*	306	1	2234	7.3	3610	11.8	3855	12.6
		2	5385	17.6	5905	19.3	6671	21.8
		3	5393	17.6	6089	19.9	6762	22.1
		4	7160	23.4	9057	29.6	9394	30.7
10*	119	1	1499	12.6	1880	15.8	1892	15.9
		2	2034	17.1	2983	25.1	3058	25.7
		3	2094	17.6	2991	25.1	3082	25.9
		4	2094	17.6	3177	26.7	3344	28.1
11*	65	1	886	14.3	1241	19.1	1521	23.4
		2	1670	25.7	2054	31.6	2132	32.8
		3	1813	27.9	2073	31.9	2190	33.7
		4	2067	31.8	2216	34.1	2314	35.6
12*	45	1	513	11.4	1053	23.4	1206	26.8
		2	801	17.8	1624	36.1	1773	39.4
		3	869	19.3	1643	36.5	1778	39.5
		4	886	19.7	1818	40.4	1939	43.1
13*	92	1	1343	14.6	2658	28.9	-	-
		2	2907	31.6	5042	54.8	-	-
		3	2925	31.8	5048	54.8	-	-
		4	2930	31.8	5981	65.0	-	-
14*	45	1	810	18.0	1404	31.2	-	-
		2	1219	27.1	2448	54.4	-	-
		3	1242	27.6	2583	57.4	-	-
		4	1480	32.9	3289	73.1	-	-
15*	97	1	1571	16.2	-	-	-	-
		2	2473	25.5	-	-	-	-
		3	2475	25.5	-	-	-	-
		4	2774	28.6	-	-	-	-
TOTAL	**3014**		48257	16.01	72193	23.7	75218	**24.9**

* Out of 100 thousand randomly selected non-scalable functions.

Table 3.8. Average Execution Time (ms) Per Function For n-input k-TFs (UF and BF), $6 \leq n \leq 15$, $1 \leq k \leq 4$, $C = 11\%$ considering Rational k-weights with Value w/l, $l \in \{1,4\}$.

TF Type \ n	6	7	8	9	10	11	12	13	14	15
1-TF, l=1	50	54	72	115	250	490	1108	2e3	3e3	7e3
1-TF, l=4	56	60	78	122	258	505	1617	22e2	36e2	8e3
UF 2-TF, l =1	54	60	80	125	275	508	1200	23e2	4e3	8e3
UF 2-TF, l =4	60	67	113	178	320	670	1420	28e2	5e3	1e4
BF 2-TF, l=1	63	96	150	290	540	1e3	2e3	5e3	9e3	2e4
BF 2-TF, l=4	88	103	189	502	596	18e2	39e2	1e4	2e4	-
UF 3-TF, l=1	225	270	324	561	1e3	22e2	5e3	85e2	2e4	-
UF 3-TF, l=4	289	450	511	887	15e2	35e2	67e2	13e3	35e3	-
BF 3-TF, l=1	460	605	617	11e2	21e2	5e3	8e3	-	-	-
BF 3-TF, l=4	481	619	705	16e2	29e2	9e3	13e3	-	-	-
UF 4-TF, l =1	13e2	14e2	2e3	3e3	65e2	14e3	3e4	-	-	-
UF 4-TF, l =4	17e2	17e2	25e3	7e3	91e2	17e3	4e4	-	-	-
BF 4-TF, l =1	2e3	2e3	4e3	8e3	13e3	-	-	-	-	-
BF 4-TF, l =4	23e3	3e3	51e2	15e3	-	-	-	-	-	-

The results in Table 3.9 show that by increasing either k or l, many 1-TFs were implemented as CTG with lower transistor count, and hence with lower area and power dissipation. In particular, in particular, 93% of selected 1-TFs were implemented with approximately 45% lower transistor count.

The following compares the transistor count of input networks, power dissipation, and delay of the CTG implementation of randomly selected 1-TFs and 2-TFs described by the weight configuration set. All functions were implemented using the CTG in [53, 83] considering only integer weights and using the proposed k-CTG described in Equation 9 considering $k \leq 4$ and $l \leq 4$, and C set to 11%. SPICE simulation took place for each function using the Berkeley Predictive Technology Models (PTM) for 45nm CMOS

Table 3.9. The Number of 1-TFs With Lower Transistor Count When Considering Rational k-Weights with Value w/l, $k \leq 4$, $l \leq 4$, $C = 11\%$.

n	1-TF	Δ (percentage reduction in CTG transistor count)			
		Δ<40%	40%≤Δ<50%	50%≤Δ<60%	Δ≥60%
1	2	2	0	0	0
2	8	8	0	0	0
3	72	72	0	0	0
4	1536	112	1242	176	6
5*	367	3	263	101	0
6*	105	0	67	31	7
7*	89	0	50	37	2
8*	66	0	57	9	0
9*	306	0	233	73	0
10*	119	0	102	14	3
11*	65	0	53	10	2
12*	45	0	41	4	0
TOTAL	2780	197	2108	455	20

* Out of 100 thousand randomly selected non-scalable functions.

transistors [85]. The V_{DD} was set to 1.1V. The applied voltages for clk were 1.1V and 0V for high voltage and low voltage, respectively. The applied load was a minimum size CMOS inverter which had a PMOS transistor with width 240nm, and a NMOS transistor with width 120nm. The length of all the PMOS and NMOS transistors were set to 45nm. The optimum sensor size was obtained using the approach in [52].

The first Column in Table 3.10 lists some randomly selected functions. They are denoted by the integer weight assignment that reflects minimum transistor count. Columns two to four list the transistor count of the input networks using the traditional approach, the power dissipation, and the delay of the CTG implementation for each function, respectively, using the approaches in [53, 83]. These values were obtained with SPICE simulations while considering corner cases by simultaneously varying the width and length of all transistors in input networks as well as the sensor part. Let d_V denote the voltage difference between two output nodes. Due to the clock enable in CTGs the

Table 3.10. Simulation Results: Transistor Count, Power Dissipation, and Delay of Randomly Selected TFs in $45nm$ Technology Using the CTG in [53, 83] and the Proposed k-CTG Based on equation (3.9) Using k-weights with Value w/l, $k \leq 4$, $l \leq 4$, $C = 11\%$.

CTG Implementation [53] and [83]				Proposed k-CTG with rational weights				% Reduction		
Function with Integer Weights	TC	Power (μW)	Delay (ps)	Optimized Function for $k \leq 4$ and $l \leq 4$	TC	Power (μW)	Delay (ps)	TC	Power	Delay
$[w_1, w_2, w_3; w_T] = [4,2,2; 3]$	11	1.98	98	$[w_1^1, w_2^1, w_3^1; w_T^1, w_T^2] = [2,1,1; 1,1]$	7	1.40	66	36%	30%	32%
$[w_1, w_3, w_{2,3}, w_{3,4}; w_T] = [-6,4,2,-2; -3]$	29	3.73	113	$[w_1^1, w_3^1, w_{2,3}^1, w_{3,4}^2; w_T^1, w_T^2] = [-3,2,1,-1; -1,-1]$	14	1.98	87	51%	47%	23%
$[w_1, w_2, w_3, w_4; w_T] = [-4,4,-2,2; 3]$	15	2.80	142	$[w_1^1, w_2^1, w_3^1, w_4^1; w_T^1, w_T^2] = [-2,2,-1,1; 1,1]$	10	1.74	105	33%	38%	26%
$[w_1, w_2, w_3, w_4, w_5; w_T] = [-7,9,2,-5,-12; -4]$	39	4.25	158	$w_1^1, w_1^4, w_2^1, w_2^4, w_3^2, w_4^1, w_4^4, w_5^1; w_T^1 = [-2,1,2,1,1,-1, -1,-3; -1]$	22	2.12	122	43%	50%	23%
$[w_1, w_2, w_3, w_4, w_5, w_6; w_T] = [4,4,7,-4,-11,-11; 5]$	43	4.08	129	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1, w_6^1; w_T^1, w_T^4] = [1,1,2,-1,-3,-3; 1,1]$	16	1.92	103	62%	53%	20%
$[w_1, w_2, w_3, w_{1,4}; w_T] = [2,2,4,2; 3]$	19	3.21	118	$[w_1^1, w_2^1, w_3^1, w_{1,4}^1; w_T^1, w_T^2] = [1,1,2,1; 1,1]$	11	1.93	87	42%	40%	26%
$[w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9; w_T] = [-5,-5,-5, -5,-5,-2,2,3,3; 4]$	39	3.30	183	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1, w_6^1, w_6^3, w_7^1, w_7^3, w_8^1, w_9^1; w_T^1, w_T^3] = [-2,-2, -2,-2,-2,-1,1,1,-1,1,1; 1,1]$	24	2.15	143	38%	35%	22%
$[w_1, w_2, w_3, w_4, w_5; w_T] = [5,5,3,3,-3; 4]$	23	3.76	114	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1; w_T^1, w_T^3] = [2,2,1,1,-1; 1,1]$	11	1.88	80	52%	50%	30%
$[w_1, w_2, w_3, w_4, w_5; w_T] = [-2,-2,4,6,6; 1]$	21	3.67	173	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1; w_T^2] = [-1,-1,2,3,3; 1]$	12	1.83	130	42%	50%	25%
$[w_1, w_{1,3}, w_{2,3}; w_T] = [2,-2,2; 1]$	19	3.21	151	$[w_1^1, w_{1,3}^1, w_{2,3}^1; w_T^2] = [1,-1,1; 1]$	11	1.80	96	42%	44%	36%
$[w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}; w_T] = [2,2,2, 2,2,2,2,2,2,2; 1]$	21	4.37	163	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1, w_6^1, w_7^1, w_8^1, w_9^1, w_{10}^1; w_T^2] = [1,1,1,1,1,1, 1,1,1,1; 1]$	12	1.53	112	43%	65%	31%

delay is calculated as the time difference between the time that clock is at 50% of its final value and the time that d_V is at 50% of its final value [52]. The power value reported in column three is an average value that includes leakage and dynamic power dissipation [12, 86].

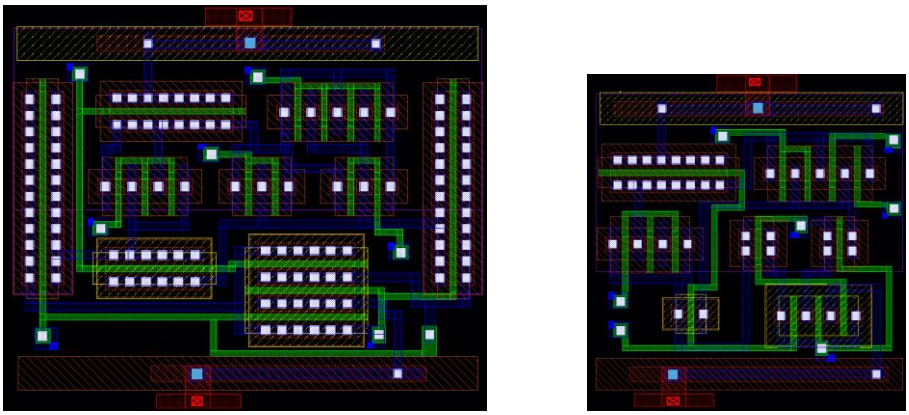
Columns five to eight show similar results when each function is implemented by the proposed k-CTG of Section 3.4 based on the formulation described in equation (3.9).

In particular, column five lists the obtained weights, column six lists the transistor count, column seven the power, and column eight the delay. The values in columns seven and eight were obtained by SPICE simulations.

The last three columns in Table 3.10 list the percentage reduction in transistor count, power dissipation, and delay, respectively, when compared to the traditional 1-TF current-mode implementation as in [53] or the 2-TF current-mode implementation as in [83]. The results show a significant decrease in power dissipation as well as a significant decrease in the delay due to the rational higher-order weights.

After the functions in Table 3.10 were synthesized by both the proposed method and the traditional in [53, 83], we proceeded to obtaining their layouts in 45nm using the Berkeley PTM model, and we derived the silicon area. Furthermore, we conducted post-layout simulation to determine the power and delay (leakage and dynamic). This experiment was conducted in order to confirm that optimizing the transistor count at the input networks (as obtained by proposed ILP method) results into area reduction when compared to the traditional CTG methods in [53, 83], and that delay and power are also reduced proportion to the saving shown by simulation at the synthesis level. Note that post-layout simulation taken to consideration the circuit parasitics which were extracted from the layout of the CTG.

As an example, Figure 3.6 (a) shows the layout of the first function listed in Table 3.10 with integer weights $[w_1, w_2, w_3; w_T] = [4,2,2; 3]$ as in [53], and Figure 3.6 (b) shows the layout using rational weights $[w_1^1, w_2^1, w_3^1; w_T^1, w_T^2] = [2,1,\ 1; 1,1]$ by the proposed method. In this case, the reduction in the area of the layout is 40%. For the remaining functions in Table 3.10, we observed that the reduction in area is even more significant.



(a) (b)

Figure 3.6. (a) The layout for 1-CTG implementation of function $[w_1, w_2, w_3; w_T] = [4,2,2; 3]$ as in [53] (b) the layout of the same function implemented using rational weights $[w_1^1, w_2^1, w_3^1; w_T^1, w_T^2] = [2,1,1; 1,1]$.

Table 3.11 provides with details on the layout area savings for all functions in Table 3.10. Please see the listed results in columns two, six, and nine. It is important to observe that the reduction in layout area is similar to the transistor count reduction in the input networks of those functions, as obtained by the proposed ILP-based synthesis method. These results show the impact of using rational weights in synthesis.

Table 3.11 also lists detailed results on power dissipation and delay obtained from post-layout simulations using the traditional approaches in [53, 83] and the proposed method. For power-related results please see columns three, seven, and ten. For delay-related results please see columns four, eight, and eleven. Again, observe that the reduction in power and delay by the proposed method, reflect the savings that were shown earlier in Table 3.10. In fact, the post-layout simulation showed that the saving in power is even higher than what was shown at the synthesis level.

The results in Tables 3.10 and 3.11 clearly demonstrate the significance of using rational weights. Furthermore, the value of the C that was set to 11% accommodates circuit parasitics due to interconnections, and all functions operate correctly.

Table 3.11. Post-Layout Results: Chip Area, Power Dissipation, and Delay of Randomly Selected TFs, in 45nm Technology Using the CTG in [53, 83] and the Proposed k-CTG Based on equation (3.9) Using k-weights with Value w/l, $k \leq 4$, $l \leq 4$, $C = 11\%$.

CTG Implementation [53] and [83]				Proposed k-CTG with rational weights				% Reduction		
Function with Integer Weights	Area (μm^2)	Power (μW)	Delay (ps)	Optimized Function for $k \leq 4$ and $l \leq 4$	Area (μm^2)	Power (μW)	Delay (ps)	Area	Power	Delay
$[w_1, w_2, w_3; w_T] = [4,2,2; 3]$	10.50	4.40	167	$[w_1^1, w_2^1, w_3^1; w_T^1, w_T^2] = [2,1,1; 1,1]$	6.25	1.98	129	40%	55%	23%
$[w_1, w_3, w_{2,3}, w_{3,4}; w_T] = [-6,4,2, -2; -3]$	16.80	5.90	196	$[w_1^1, w_3^1, w_{2,3}^1, w_{3,4}^2; w_T^1, w_T^2] = [-3,2,1, -1; -1, -1]$	8.25	1.94	160	51%	67%	18%
$[w_1, w_2, w_3, w_4; w_T] = [-4,4, -2,2; 3]$	13.10	9.32	169	$[w_1^1, w_2^1, w_3^1, w_4^1; w_T^1, w_T^2] = [-2,2, -1,1; 1,1]$	6.80	3.63	133	48%	61%	21%
$[w_1, w_2, w_3, w_4, w_5; w_T] = [-7,9,2, -5, -12; -4]$	20.00	8.44	326	$w_1^1, w_1^4, w_2^1, w_2^4, w_3^2, w_4^1, w_4^4, w_5^1; w_T^1 = [-2,1,2,1,1, -1, -1, -3; -1]$	11.40	2.95	270	43%	65%	17%
$[w_1, w_2, w_3, w_4, w_5, w_6; w_T] = [4,4,7, -4, -11, -11; 5]$	23.80	11.81	470	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1, w_6^1; w_T^1, w_T^4] = [1,1,2, -1, -3, -3; 1,1]$	9.10	5.07	399	62%	57%	15%
$[w_1, w_2, w_3, w_{1,4}; w_T] = [2,2,4,2; 3]$	13.00	9.21	283	$[w_1^1, w_2^1, w_3^1, w_{1,4}^1; w_T^1, w_T^2] = [1,1,2,1; 1,1]$	6.90	4.32	223	47%	53%	21%
$[w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9; w_T] = [-5, -5, -5, -5, -5, -2,2,3,3; 4]$	18.40	9.16	351	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1, w_6^1, w_6^3, w_7^1, w_7^3, w_8^1, w_9^1; w_T^1, w_T^3] = [-2, -2, -2, -2, -2, -1,1,1, -1,1,1; 1,1]$	11.40	4.67	284	38%	49%	19%
$[w_1, w_2, w_3, w_4, w_5; w_T] = [5,5,3,3, -3; 4]$	13.80	8.25	490	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1; w_T^1, w_T^3] = [2,2,1,1, -1; 1,1]$	6.90	3.05	357	50%	63%	27%
$[w_1, w_2, w_3, w_4, w_5; w_T] = [-2, -2,4,6,6; 1]$	12.90	9.79	275	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1; w_T^2] = [-1, -1,2,3,3; 1]$	7.10	3.91	223	45%	60%	19%
$[w_1, w_{1,3}, w_{2,3}; w_T] = [2, -2,2; 1]$	12.50	8.80	415	$[w_1^1, w_{1,3}^1, w_{2,3}^1; w_T^2] = [1, -1,1; 1]$	6.90	4.05	298	45%	54%	28%
$[w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}; w_T] = [2,2,2, 2,2,2,2,2,2,2; 1]$	12.90	8.45	317	$[w_1^1, w_2^1, w_3^1, w_4^1, w_5^1, w_6^1, w_7^1, w_8^1, w_9^1, w_{10}^1; w_T^2] = [1,1,1,1,1,1, 1,1,1,1; 1]$	7.10	2.45	253	45%	71%	20%

3.7 Conclusion

It has been demonstrated that the presented approach can implement many more functions as current mode threshold logic gates with similar or less transistor count when compared to existing method. Also a significant percentage of existing threshold functions can be implemented as current mode threshold gates with approximately 60% less power dissipation, and 20% less delay when considering higher order non-integer weights in the

presence of aging and circuit parasitics.

In future work, heuristic approaches will be investigated to implement higher input k-TFs with rational weights. In addition, we will investigate the impact of emerging technology on resistive devices such as memristors and spin torque transfer devices. Synthesis of complex circuit specifications will also be built upon existing 1-TF based synthesis methods.

CHAPTER 4

MAXIMIZING THE NUMBER OF THRESHOLD LOGIC FUNCTIONS USING RESISTIVE MEMORY

4.1 Introduction

The generalized definition of TF in equation (3.2) is called k^{th}-order TF (k-TF). A TG that implements a k-TF is also called k-TG. Observe that when $k = 1$, equation (3.2) simplifies to equation (3.1). The latter is also called 1-TF, and its implementations have been studied extensively in the literature [50-55] and [57-59], among others. As mentioned in Chapter 3, a TG generally contains two input networks and a sensor [104]. Each input network consists of several components connected in parallel. Every component implements a weight value.

For each input pattern, some input components are active while the threshold component is always active. The binary output of the TG is determined by the sensor, which compares the currents (current mode TG) or voltages (differential mode TG) of the sum of the active components of the two input networks, and amplifies the difference.

In TG, the power dissipation depends primarily on two factors: the number of parallel components of the input networks, i.e., the transistor count of the input networks which is the total number of unit size transistors that implements weights, and the sensor size which is proportional to the transistor count of input networks [52]. The less the transistor count in input networks, the lower the power dissipation in a TG is. A reduced transistor count subsequently reduces the sensor size which, in turn, decreases further the power dissipation [52].

Several interesting CMOS-based circuit concepts have been proposed in [50-55], [57-59], among others, for TG implementations for the special case of 1-TF. In these designs, each weight component is a single NMOS (or PMOS) transistor which is controlled by one input. This limits the number of functions that can be implemented as TGs. Authors in [83] showed recently that more functions can be implemented as TGs if a component contains multiple transistors (connected in series) where each is controlled by either one or two inputs (2-TFs). In fact, they showed in [83] that the transistor count of several 1-TGs can be reduced when implemented as 2-TGs.

This chapter is the extention of Chapter 3, and proposes a new method to implement efficiently the k-weight components in k-TGs using non-volatile resistive memories (memristors). The approach has been implemented for $1 \leq k \leq 4$. Memristors are used as the weight components instead of using transistors as weights. The resistance value of a memristor is called its memristance, and the range of memristance is used to define different weight values. We call such gates k^{th}-order memristive TG (k-mTG). This method of weight implementation reduces significantly the transistor count of the input networks. The presented work is an extended version of the abstract in [87].

Many memristive architectures have been proposed in the literature to implement 1-mTGs [77]. The approaches in [48], [76], [88-91], among others, use different memristance values to implement weights. An advantage of these approaches as well as the proposed design in this paper is that the memristor can be programmed to implement alternative functions and, can be used for rapid prototyping [79, 92, 93]. When considering an upper bound on transistor count, the proposed approach, which benefits from higher order weight components, implements more functions as k-mTG than the existing 1-mTG

approaches in [48], [76], [88-91]. The difference in the transistor count is due to architectural considerations that are orthogonal to the goal of this chapter. For example, the approaches in [50], [79] use memristors in order to increase 1-TG robustness but the weights were implemented in CMOS technology. The proposed method attempts to reduce the transistor count on the input networks using mTG to implement weights, and this is an orthogonal design object. The proposed method is applicable to any 1-mTG method with a specific sensor component.

In order to demonstrate the impact of memristive k-weights in terms of area, power dissipation, and delay, this chapter considers without loss of generality on the current-mode TG (CTG) implementation in [52].

This chapter is organized as follows. Section 4.2 provides preliminaries and describes briefly the existing TG implementations. The memristive implementation of k-TFs is proposed in Section 4.3. Section 4.4 provides experimental results, and Section 4.5 concludes the paper.

4.2 Preliminaries

This section provides with preliminaries on bipolar metal-oxide memristors and overviews on TFs and their traditional CMOS-based and memristive TG implementations.

4.2.1 Bipolar metal-oxide memristors

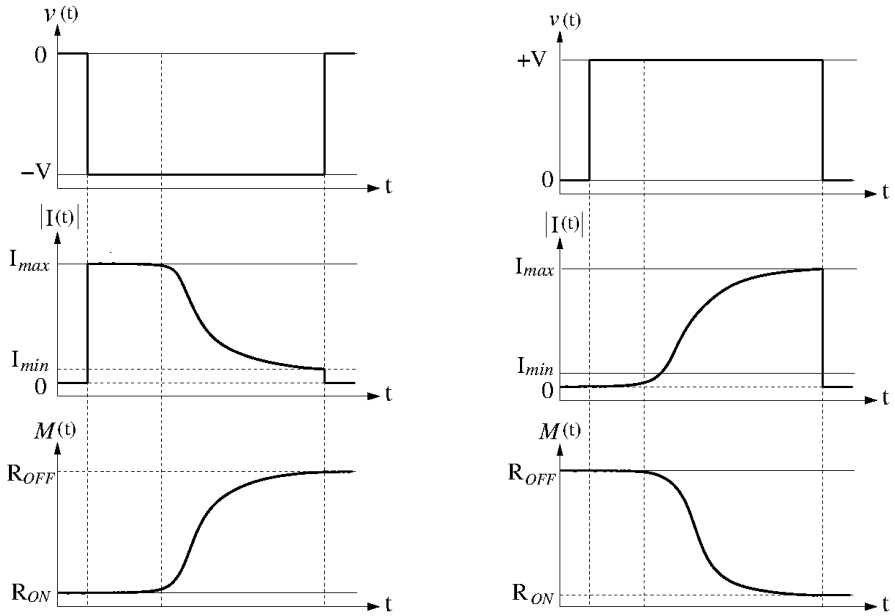
This chapter assumes bipolar metal-oxide memristors. Accurate models for this memristor behavior have been developed in [15, 94-96]. This memristor is a two-terminal device that is formed by a metal-oxide-metal thin film sandwiched between two electrodes [1, 3]. Bipolar metal-oxide memristors are variable resistors. Resistance switching in such

memristors relates to the formation or partial dissolution of the nanoscale conducting filament. This filament changes due to the drift and diffusion direction of the mobile oxygen anions and oxygen vacancies created under Joule heating and electric fields [29].

The memristor is written (or programmed) by biasing positive and negative voltages across the electrodes. By applying a positive (negative) voltage across the device, the total resistance of the device decreases (increases) [21, 30]. Let R_{ON} and I_{max} denote the minimum possible resistance value and its corresponding current, respectively. Similarly, let R_{OFF} and I_{min} denote the maximum possible resistance value and its corresponding current, respectively. Figure 4.1 shows the conductivity and resistivity transition behavior of a memristor during writes (write operations) with positive and negative pulses, $\pm V(t)$. This behavior is accurately reflected by several models such as [15, 94], among others.

As shown in Figure 4.1, when applying a pulse (either positive or negative) to the memristor, the device does not switch immediately, and waits for a random time. This is due to the stochastic nature of the filament formation. The difference between the application time of the write pulse and the time that the device starts to switch is called the wait (switching) time [16, 35, 97]. During the wait time, which varies from cycle-to-cycle, the current and resistivity do not practically change and is considered to be fixed [35].

When applying a read voltage, the current and resistivity change after the wait time. The resistivity change is called memristor leakage and may affect the functionality of current-based mTGs. Based on the stochastic behavior of the filament, it may occur (worst-case scenario) that there is leakage each time the gate operates. However, the



(a) With negative pulse. (b) With positive pulse.

Figure 4.1. Resistivity and current behavior for positive and negative writes for a bipolar metal-oxide memristor with R_{ON}=5KΩ, R_{OFF}=5MΩ, $\pm V = 1V$, using [15, 94].

resistivity switching time is in milliseconds whereas the gate delay is in picoseconds. Therefore, reprogramming (weight tuning) to combat the memristor leakage is only required after millions of mTG operation cycles [98-101]. In fact, programming is done periodically in order to ensure that there is no discrepancy in the memristive value due to leakage from frequent read operations.

Filament variability may result into imprecise memristance value. This chapter assumes robust programming methods, such as in [95] and [101], that program the memristor precisely to the targeted resistance with negligible error (less than 1%), and thus cope with filament variability.

4.2.2 CMOS-based and memristive threshold logic gates

The concept of k-TF and its operation is explained with an example.

***Example 1*:** Consider a five input 3-TF F_1 with non-zero weight configuration $\{w_1, w_2, w_3, w_{3,4,5}; w_T\} = \{2, 2, 4, -2; 3\}$. 1-weight w_i, $1 \leq i \leq 3$, is active when $x_i = 1$, and the 3-weight $w_{3,4,5}$ is active when $x_3 \cdot x_4 \cdot x_5 = 1$. For a given input pattern $\{x_1, x_2, x_3, x_4, x_5\} = \{0, 1, 1, 0, 0\}$, we get $(2 \cdot 0) + (2 \cdot 1) + (4 \cdot 1) + (-2 \cdot 1 \cdot 0 \cdot 0) = 6 > 3$. Hence, for the above input pattern, F_1 evaluates to logic one. Similarly, for input pattern $\{0, 0, 1, 1, 1\}$, we get $(2 \cdot 0) + (2 \cdot 0) + (4 \cdot 1) + (-2 \cdot 1 \cdot 1 \cdot 1) = 2 < 3$, and F_1 is logic zero. □

In CMOS-based TG designs, parallel weight components are implemented with NMOS (or PMOS) transistors [50-55] and [57-59]. The area of a transistor with width $w \cdot X$ is practically the same as w minimum size transistor. Let X denote the width of a minimum size transistor, and I be the active current through it. Each transistor implements a 1-weight with value w when its width and current are $w \cdot X$ and $w \cdot I$, respectively. The gate of the transistor is connected to an input. The gate of the NMOS transistor for the threshold is connected to the power supply (it is active for all input patterns). All transistors have the same length which is determined by the used technology. Therefore the transistor count for such gates is $w_T + \sum_{i=1}^{n} w_i$, which is the total sum of threshold and input weights.

For higher order components, a CMOS-based k-weight with value w is implemented with k transistors of the same size which are connected in series [83]. The size of each transistor is $k \cdot w \cdot X$, and the current through it is $w \cdot I$. The transistor gates are connected to k TG inputs [83], and the transistor count of such component is k^2.

The transistor count reduces to $n + 1$, independent of weight values, when weights are implemented with memristors [48, 76]. Each weight component consists of a memristor and a minimum size NMOS (or PMOS) transistor connected in series. Let I'

be the active current through a memristive weight component when memristor is assigned to a memristance value R_{max}. Any weight with value w can be implemented by programming the memristor to memristance R_{max}/w so that the current through this component becomes $w \cdot I'$. The transistor count is always one and does not relate to the weight values.

The significant transistor count reduction in mTGs results into lower power dissipation [48]. Moreover, the sensor size can be decreased because of the less effective capacitance of the input networks on the output nodes [82]. This reduces further the power dissipation without losing performance. Consider the following example.

***Example 2*:** Consider the 4-variable function $F_2 = x_4x_3 + x_3x_2 + x_3x_1 + x_2x_1$. This function is a 1-TF with weight configuration $w = [w_1, w_2, w_3, w_4;\ w_T] = [4,4,6,2;7]$. Note that for implementation purposes, weights are considered to be integers, and the equality is removed from equation (3.2). Figure 4.2 shows the CMOS-based CTG [52] and memristive CTG [76] implementations of F_2. Let X denote the size of a minimum width transistor to implement a unit integer weight. The total transistor count reduces from 23 to 5 when implementing F_2 as a memristive CTG. □

A small fraction of TFs are 1-TFs [58]. This limits the impact of TGs in digital circuit synthesis. In order to identify more threshold functions, we consider the generalized higher order TF definition described in equation (3.1), and we modify the input network of the existing mTGs to implement efficiently such TFs.

4.3 Generalized memristive threshold logic gates

In the proposed mTG implementation, any k-weight is implemented by k minimum

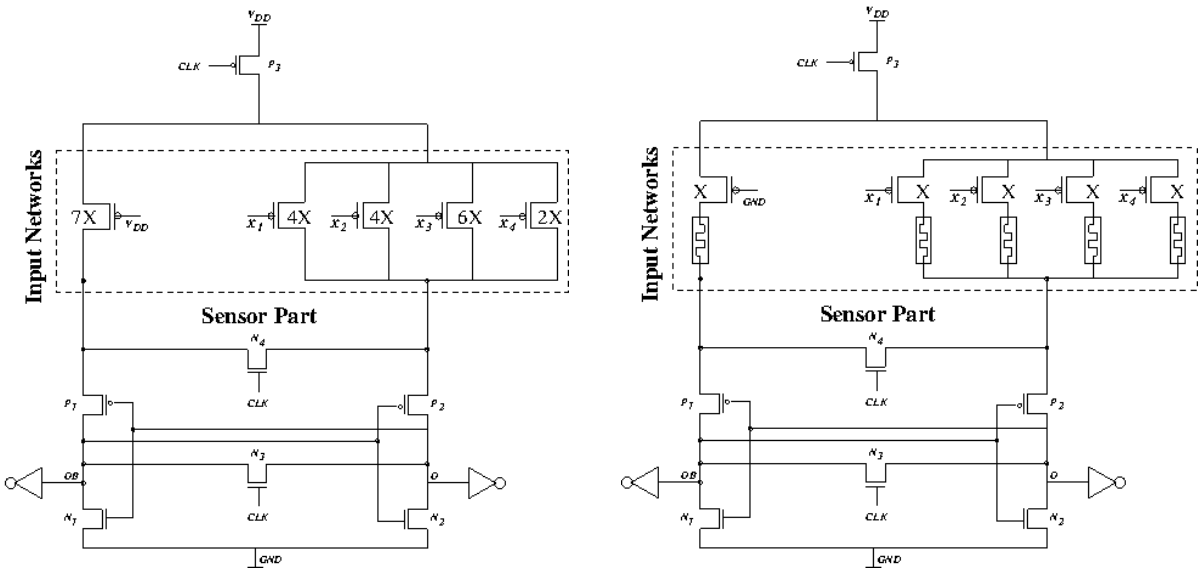


Figure 4.2. The Current mode TG (CTG) implementations for function F_2 in example 2 (a) CTG as in [52] (b) memristive CTG as in [76].

size transistors and a memristor, all connected in series. Therefore, the transistor count of any k-weight with value w is k, and the active current through it is $w \cdot I$. This current value is enforced with appropriate resistivity. This way, the transistor count of the gate reduces significantly which, in turn, reduces the power of the gate. A reduced transistor count subsequently reduces the sensor size which, in turn, decreases further the power dissipation [52]. Figure 4.3 shows the m-weight components, and their transistor count for $1 \leq m \leq k$.

In order to determine if a function is a k-TF we form an Integer Linear Programming (ILP) constraint per input pattern using the right-hand side of equation (3.2) according to the binary evaluation of the function for the pattern [52, 83]. The transistor count ratio of a memristive k-weight with value w over a memristive 1-weight with the same value is called the penalty factor. In order to obtain minimum possible transistor count, the ILP must minimize quantity

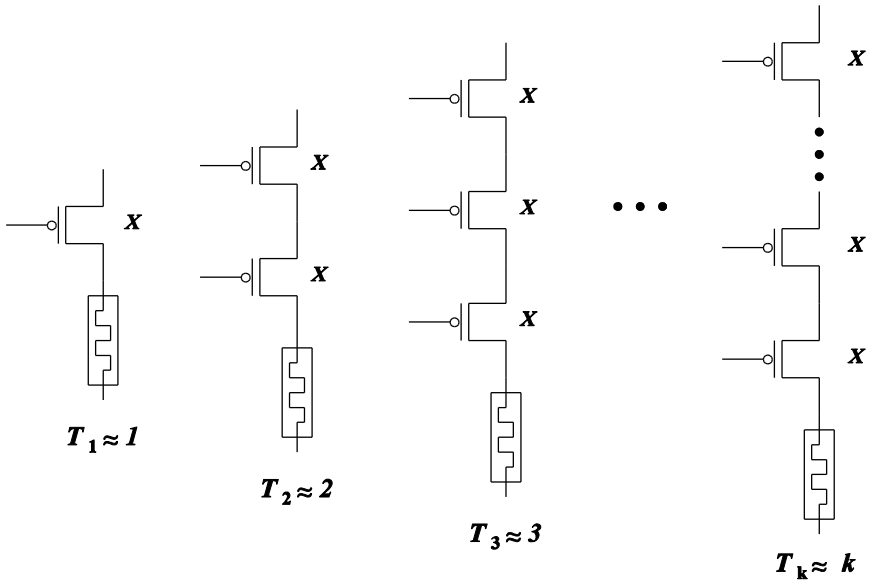


Figure 4.3. Memristive m-weight components for $1 \leq m \leq k$.

$$1 \cdot \left\lceil \frac{|w_T|}{U} \right\rceil + 1 \cdot \sum_{i_1=1}^{n} \left\lceil \frac{|w_{i_1}|}{U} \right\rceil + 2 \cdot \sum_{i_1=1}^{n-1} \sum_{i_2=2}^{n} \left\lceil \frac{|w_{i_1,i_2}|}{U} \right\rceil + \cdots + k \cdot \sum_{i_1=1}^{n-k+1} \sum_{i_2=i_1+1}^{n-k} \cdots \sum_{i_k=i_{k-1}+1}^{n} \left\lceil \frac{|w_{i_1,i_2,\ldots,i_k}|}{U} \right\rceil \tag{4.1}$$

where U is the upper bound on each weight component, $\lceil x \rceil$ denotes the ceiling function which results into the least integer that is greater than or equal to x, and $|w_x|$ denotes the absolute value for each weight w_x, and $w_x \in \{w_T, w_{i_1}, w_{i_1,i_2}, \ldots, w_{i_1,i_2,\ldots,i_k}\}$.

The performance of each CMOS transistor is impacted by process variations. Also the memristance changes over time due to memristor leakage. These factors may modify the designed weight and change the functionality of the mTG. Furthermore, weight values should be assigned to appropriate memristance values so that the leakage current of any inactive weight is negligible when compared to the active current of a unit weight. The following show how to assign resistive weights that tolerate weight variations.

Let constant value C denote the maximum weight deviation and let $|w|$ denote the absolute value of weight w. Let $w_{i_1,i_2,\ldots,i_m}$ denote a m-weight, $1 \leq m \leq k$. The pattern dependent inequalities of the ILP are rewritten as

$$\sum_{\forall m}\left(w_{i_1,i_2,\ldots,i_m} - C \cdot \left|w_{i_1,i_2,\ldots,i_m}\right|\right) x_{i_1} \cdot x_{i_2} \ldots \cdot x_{i_m} > w_T + C \cdot |w_T| \tag{4.2}$$

when function evaluates to 1, and for the remaining input patterns as

$$\sum_{\forall m}\left(w_{i_1,i_2,\ldots,i_m} + C \cdot \left|w_{i_1,i_2,\ldots,i_m}\right|\right) x_{i_1} \cdot x_{i_2} \ldots \cdot x_{i_m} < w_T - C \cdot |w_T| \tag{4.3}$$

The above may only change the total sum of weights. For example, the weight configuration of F_1 in Example 1 considering $C = 5\%$ becomes $w = [w_1, w_2, w_3, w_4, w_5; w_T] = [2, 3, 5, 5, 5; 6]$. Extensive experimental evaluation (see Section 4.4) shows that the value of C does not exceed 5%. The following two examples illustrate the concept of k-TF and the ILP-based method to identify a k-TF considering $C = 8\%$.

***Example 3*:** Consider a 4-input function $F_3 = x_1x_2 + x_3x_4$ with a set of all unknown weights $w = [w_1,\ w_2,\ w_3,\ w_4,\ w_{1,2},\ w_{1,3},\ w_{1,4},\ w_{2,3},\ w_{2,4},\ w_{3,4}; w_T]$. Table 4.1 lists the inequalities of F_3 based on the k-TF definition when $k = 2$. For an input pattern, weight w (either 1-weight or 2-weight) appears in the inequality when all its input signals evaluate to 1. The objective function for a 2-TF is to minimize quantity $\left\lceil\frac{|w_T|}{10}\right\rceil + 1 \cdot \sum_{i=1}^{4}\left\lceil\frac{|w_i|}{10}\right\rceil + 2 \cdot \sum_{i=1}^{3}\sum_{j=\mathrm{i}+1}^{4}\left\lceil\frac{|w_{i,j}|}{10}\right\rceil$ considering weights are in the range [-10, +10]. For the set of constraints listed in Table 4.1, an optimum solution is $w = [0,0,0,0,2,0,0,0,0,2; 1]$. □

***Example 4*:** Function $F_4 = x_1x_2 + (x_1 + x_2)x_3x_4x_5$ is neither an 1-TF nor 2-TF. It is a 3-TF and the weight configuration to implement as a memristive current mode 3-TG is $w = [w_1, w_2, w_{3,4,5}; w_T] = [3, 3, 2; 4]$.

Table 4.1. The Truth Table and ILP Constraints for F_3 Considering 1-weights and 2-weight and $C = 8\%$.

Truth Table			Inequalities
Input Pattern $(x_1x_2x_3x_4)$		F_3	
P_0	0000	0	$0 < 0.92 \cdot w_T$
P_1	0001	0	$1.08 \cdot w_4 < 0.92 \cdot w_T$
P_2	0010	0	$1.08 \cdot w_3 < 0.92 \cdot w_T$
P_3	0011	1	$0.92 \cdot (w_3 + w_4 + w_{3,4}) \geq 1.08 \cdot w_T$
P_4	0100	0	$1.08 \cdot w_2 < 0.92 \cdot w_T$
P_5	0101	0	$1.08 \cdot (w_2 + w_4 + w_{2,4}) < 0.92 \cdot w_T$
P_6	0110	0	$1.08 \cdot (w_2 + w_3 + w_{2,3}) < 0.92 \cdot w_T$
P_7	0111	1	$0.92 \cdot (w_2 + w_3 + w_4 + w_{2,3} + w_{2,4} + w_{3,4}) \geq 1.08 \cdot w_T$
P_8	1000	0	$1.08 \cdot w_1 < 0.92 \cdot w_T$
P_9	1001	0	$1.08 \cdot (w_1 + w_4 + w_{1,4}) < 0.92 \cdot w_T$
P_{10}	1010	0	$1.08 \cdot (w_1 + w_3 + w_{1,3}) < 0.92 \cdot w_T$
P_{11}	1011	1	$0.92 \cdot (w_1 + w_3 + w_4 + w_{1,3} + w_{1,4} + w_{3,4}) \geq 1.08 \cdot w_T$
P_{12}	1100	1	$0.92 \cdot (w_1 + w_2 + w_{1,2}) \geq 1.08 \cdot w_T$
P_{13}	1101	1	$0.92 \cdot (w_1 + w_2 + w_4 + w_{1,2} + w_{1,4} + w_{2,4}) \geq 1.08 \cdot w_T$
P_{14}	1110	1	$0.92 \cdot (w_1 + w_2 + w_3 + w_{1,2} + w_{1,3} + w_{2,3}) \geq 1.08 \cdot w_T$
P_{15}	1111	1	$0.92 \cdot (w_1 + w_2 + w_3 + w_4 + w_{1,2} + w_{1,3} + w_{1,4} + w_{2,3} + w_{2,4} + w_{3,4}) \geq 1.08 \cdot w_T$
minimize:			$\left\lceil \frac{\lvert w_T \rvert}{10} \right\rceil + 1 \cdot \sum_{i=1}^{4} \left\lceil \frac{\lvert w_i \rvert}{10} \right\rceil + 2 \cdot \sum_{i=1}^{3} \sum_{j=i+1}^{4} \left\lceil \frac{\lvert w_{i,j} \rvert}{10} \right\rceil$

Figure 4.4 shows the CTG implementation of F_4 using memristors when $k = 3$. In Figure 4.4, all transistors of input networks are minimum size transistors. □

4.4 Experimental results

In order to evaluate the proposed memristive approach in terms of area, power dissipation, and delay, we implemented k-weights with value w, $1 \leq k \leq 4$ and $1 \leq w \leq 10$, using the Berkeley Predictive Technology Models (PTM) for 45nm CMOS transistors [85]. For each weight value, the memristor was assigned to a memristance value so that

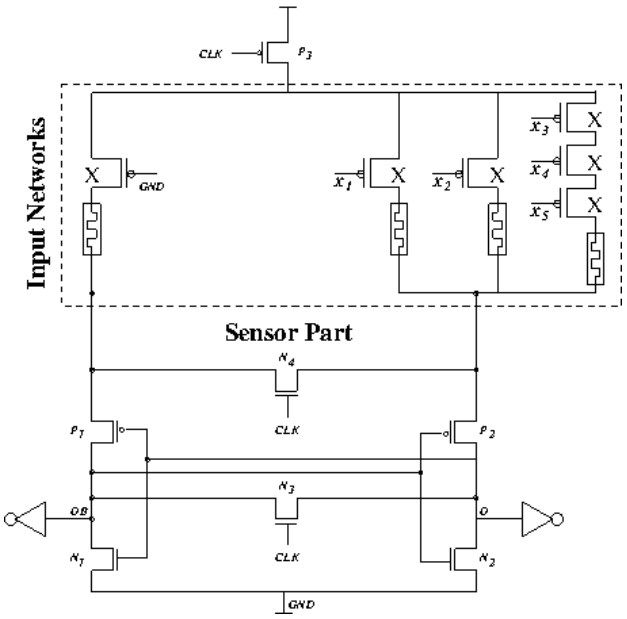


Figure 4.4. The memristive Current mode TG (CTG) implementations for function F_3 in example 3.

the current through that component was proportional to its weight value. We experienced with the popular $\mathrm{TiO_2}$ bipolar metal-oxide memristors for which the VTEAM model in [15] allows for accurate simulations. The length of a memristor and its memristance boundaries R_{ON} and R_{OFF} were set to 5nm, 5KΩ, and 5MΩ, respectively. The remaining memristor parameters were set as in [35]. $\mathrm{V_{DD}}$ was set to $1V$.

Figure 4.5 provides details on resistivity values to implement k-weights. For example, a resistivity of 450KΩ implements weight 4 whereas 300KΩ implements weight 6.

All function implementations listed in this section can tolerate the maximum weight deviation C due to process variations in transistor parameters and any memristor leakage. Table 4.2 shows the current (weight) variation for memristive k-weight components considering 3% variation in width and length of transistors in 45nm technology. We also considered that there might be an additional 3% weight variation due to memristor

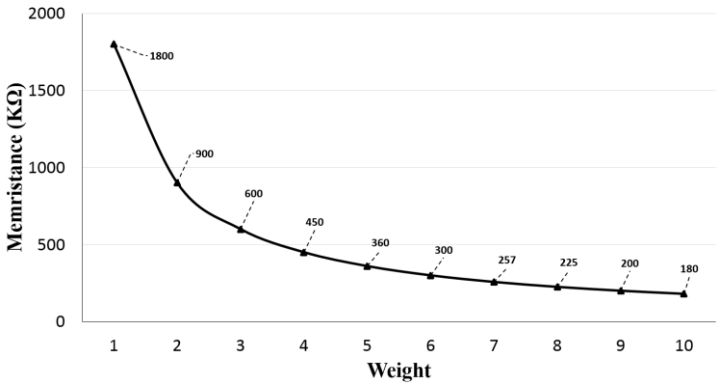


Figure 4.5. Different resistivity values to implement memristive k-weights with value w, for $1 \leq w \leq 10$ and $1 \leq k \leq 4$.

leakage over time as well as any imprecise weight programming using [95, 101]. SPICE simulations were conducted for each k-weight component using the Berkeley Predictive Technology Models (PTM) for 45nm CMOS transistors [76] and VTEAM model in [15] for memristors with R_{ON}=5KΩ, R_{OFF}=5MΩ. The voltage difference over components was set to 1V. The simulations showed that the current (weight) variation of memristive k-weights, $1 \leq k \leq 4$, was lower that 8% when considering 3% variation in all transistor parameters combined with an additional 3% variation in weight due to leakage or imprecise programming. Therefore, C was set to 8% in all following experiments. Note that the length and area of the memristors do not affect the functionality of the mTGs because R_{ON} and R_{OFF} are not used as weights.

The following compare the transistor count, sensor size, power dissipation, and delay of the proposed mTG implementation of randomly selected k-TFs with the CMOS-based implementation in [83]. All functions were described by the weight configuration set and implemented using the current mode TG in [52] considering $k \leq 4$. The V_{DD} was set to 1V. The applied voltages for the clock clk were 1V and 0V for high voltage and low voltage, respectively. The applied load was a minimum size CMOS inverter which had a

Table 4.2. Weight Variation for Memristive k-weight components, $1 \leq k \leq 4$, considering 3% Variation in Width and Length of Transistors, and 3% Variation in Memristor Leakage and Imprecise programming.

Variation Source	Current variation for k-weights			
	1-weight	2-weight	3-weight	4-weight
Transistor Width	2.40%	2.40%	2.28%	2.16%
Transistor Length	2.16%	2.10%	2.04%	2.04%
Memristor Leakage or Imprecise Programming	3.0%	3.0%	3.0%	3.0%
Total	7.56%	7.50%	7.32%	7.20%

PMOS transistor with width 240nm, and a NMOS transistor with width 120nm. The length of all the PMOS and NMOS transistors were set to 45nm. The optimum sensor size was obtained using the approach in [52].

The first Column in Table 4.3 lists the functions that are represented by the optimum (minimum transistor count) integer weight assignment. Columns two to five list the transistor count, sensor size, power dissipation, and the delay of the TG implementation in [83] for each function. Note that the gate delay corresponds to the critical input configurations where the difference between the sum of active weights between two input networks is minimal, and at the same time the total number of active components is minimum. Also, power dissipation corresponds to the average power (including leakage and dynamic) for all possible input patterns [12,102,103]. These values were obtained with SPICE simulations while considering corner cases by simultaneously varying the width and length of all transistors in input networks as well as the sensor part. Note that X in column three denotes the size (area) of a minimum size transistor. Columns six to nine show similar results when considering each function is implemented by the proposed current mode higher order mTG implementation. The weights were

Table 4.3. Transistor Count, Sensor Size, Power Dissipation, and Delay of Randomly Selected k-TFs, $1 \leq k \leq 4$, in $45nm$ Technology Using the CMOS Approach in [83] and the Proposed Memristive Approach. (X Denotes the Minimum Transistor Size.)

Function	[83]				Proposed				% Reduction			
	TC	Sensor size	Power (μW)	Delay (ps)	TC	Sensor size	Power (μW)	Delay (ps)	TC	Sensor size	Power	Delay
$[w_1, w_2, w_3; w_T] = [4,2,2; 5]$	13	$50 \cdot X$	0.76	170	4	$6 \cdot X$	0.27	195	70%	88%	64%	-14%
$[w_1, w_2, w_3, w_4; w_T] = [4,4,2,2; 9]$	21	$50 \cdot X$	0.76	178	5	$6 \cdot X$	0.27	208	76%	88%	64%	-16%
$[w_1, w_2, w_3, w_4, w_5, w_6; w_T] = [4,4,10, -4, -8, -8; 5]$	43	$72 \cdot X$	0.97	205	7	$6 \cdot X$	0.27	215	83%	90%	72%	-5%
$[w_1, w_3, w_{2,3}, w_{3,4}; w_T] = [-6,4, -2, -2; -5]$	31	$66 \cdot X$	0.79	182	7	$6 \cdot X$	0.28	193	77%	90%	65%	-6%
$[w_1, w_{1,3}, w_{2,3}; w_T] = [2, -2,2; 1]$	19	$50 \cdot X$	0.81	187	6	$8 \cdot X$	0.28	220	68%	84%	65%	-17%
$[w_1, w_2, w_3, w_{1,4}; w_T] = [2,2,4,2; 3]$	19	$50 \cdot X$	0.82	187	6	$8 \cdot X$	0.28	202	68%	84%	66%	-8%
$[w_1, w_2, w_3, w_4, w_5, w_{4,5}, w_{1,2,3}; w_T] = [2,2, -2, -2,4, -2,2; 1]$	39	$152 \cdot X$	1.03	236	11	$8 \cdot X$	0.29	233	71%	94%	72%	1%
$[w_1, w_2, w_3, w_4, w_{1,2}, w_{1,3}, w_{1,2,3}; w_T] = [4,2,2, -2, -2, -2, -2; 3]$	47	$166 \cdot X$	1.48	259	12	$11 \cdot X$	0.30	245	75%	93%	79%	5%
$[w_3, w_{1,2}, w_{2,4}, w_{1,2,4,5}; w_T] = [-2,2, -4,2; -1]$	59	$470 \cdot X$	3.96	312	10	$11 \cdot X$	0.32	278	83%	97%	92%	11%
$[w_1, w_2, w_3, w_4, w_{1,2,3,4}; w_T] = [2,2,2,2, -4; 5]$	77	$450 \cdot X$	4.55	320	9	$10 \cdot X$	0.32	290	88%	97%	93%	9%

implemented using the memristance values listed in Figure 4.5.

SPICE simulations were conducted to observe the effect of current leakage through inactive parallel components. The sum of those currents could theoretically approach current from active parallel components and thus invalidate the functionality of the TG. It was observed that the current of an active unit weight was 40 times greater than the maximum leakage current of any inactive component. Simulations also showed that the sensor component operates correctly when higher order mTGs have up to 20 parallel components. However, the maximum number of parallel components observed on any designed mTG never exceed 20, and rarely exceeded 15. Therefore all designed

mTGs operated correctly. In fact, it was observed that the higher order CMOS-based implementation of every designed function (as in [83]) also operated correctly.

The last four columns in Table 4.3 list the percentage reduction in transistor count, sensor size, power dissipation, and delay, respectively, when compared to the CMOS-based current-mode implementation in [83]. The listed results show a significant decrease in area and power dissipation and approximately same delay due to the memristive weights.

Table 4.4 also lists detailed results on area, power dissipation (leakage and dynamic) and delay obtained from post-layout simulations for all functions in Table 4.3. We considered 45nm technology. Post-layout simulation has taken in to consideration circuit parasitics which were extracted from the layout of the TG.

Columns two, five, and eight in Table 4.4 provides with details on the layout area savings for all functions in Table 4.3. Observe that the reduction in layout area is similar to the transistor count reduction in the input networks of those functions, as obtained by the proposed ILP-based synthesis method.

Columns three, six, and nine show the power-related results, and columns four, seven, and ten list the delay-related results. Again, observe that the reduction in power and delay by the proposed method, reflect the savings that were shown earlier in Table 4.3. In fact, post-layout simulation showed that the saving in power is even higher than what was shown at the synthesis level. Any difference in reported average power dissipation among Tables 4.3 and 4.4 is due to parasitics.

Furthermore, the value of the constant C that was set to 8% accommodates circuit parasitics due to interconnections, and all functions operated correctly. Observe that the

Table 4.4. Post-Layout Results: Chip Area, Power Dissipation, and Delay of Randomly Selected k-TFs, $1 \leq k \leq 4$, in $45nm$ Technology Using the CMOS Approach in [83] and the Proposed Memristive Approach.

Function	[83]			Proposed			% Reduction		
	Area (μm^2)	Power (μW)	Delay (ps)	Area (μm^2)	Power (μW)	Delay (ps)	Area	Power	Delay
$[w_1, w_2, w_3; w_T] = [4,2,2; 5]$	11.00	1.68	280	1.96	0.67	285	82%	60%	0%
$[w_1, w_2, w_3, w_4; w_T] = [4,4,2,2; 9]$	11.56	1.96	310	1.96	0.76	360	83%	61%	-16%
$[w_1, w_2, w_3, w_4, w_5, w_6; w_T] = [4,4,10,-4,-8,-8; 5]$	14.60	1.90	415	2.16	0.70	456	85%	63%	-10%
$[w_1, w_3, w_{2,3}, w_{3,4}; w_T] = [-6,4,-2,-2; -5]$	13.30	1.75	405	2.16	0.71	446	84%	59%	-10%
$[w_1, w_{1,3}, w_{2,3}; w_T] = [2,-2,2; 1]$	11.56	1.96	250	2.40	0.68	260	79%	65%	-4%
$[w_1, w_2, w_3, w_{1,4}; w_T] = [2,2,4,2; 3]$	11.56	1.98	296	2.40	0.76	308	79%	62%	-4%
$[w_1, w_2, w_3, w_4, w_5, w_{4,5}, w_{1,2,3}; w_T] = [2,2,-2,-2,4,-2,2; 1]$	20.10	2.27	370	2.55	0.82	381	87%	64%	-3%
$[w_1, w_2, w_3, w_4, w_{1,2}, w_{1,3}, w_{1,2,3}; w_T] = [4,2,2,-2,-2,-2,-2; 3]$	20.26	3.40	476	3.36	0.90	478	83%	73%	0%
$[w_3, w_{1,2}, w_{2,4}, w_{1,2,4,5}; w_T] = [-2,2,-4,2; -1]$	62.88	11.84	503	3.36	1.15	468	94%	90%	7%
$[w_1, w_2, w_3, w_4, w_{1,2,3,4}; w_T] = [2,2,2,2,-4; 5]$	62.50	12.01	542	3.36	1.22	525	94%	90%	3%

reported post-layout simulations in Table 4.4 show that the savings in power and delay are similar to those reported for pre-layout simulations in Table 4.3. The results in Tables 4.3 and 4.4 clearly demonstrate the significance of using memristive k-weights.

The following evaluates the flexibility of the proposed order mTG design in terms of the number of implementable functions, and compare with existing CMOS-based approach in [83] and memristive approaches in [48, 76, 89, 90]. The ILP-based approach in [83] has been modified as explained in previous section and implemented in the C++ language on an Intel Xenon 2.4GHz with 8GB memory. To evaluate the impact, we examined non-scalable functions with up to fifteen inputs. (An n-input non-scalable

function requires no less than n non-empty levels of variables in the Binary Decision Diagram (BDD) representation for some ordering of the variables [58].) Weight values were in the range [-10, +10].

Table 4.5 presents non-scalable n-input TFs can be implemented different TG approaches considering that the transistor count for each gate is bounded to $4n$. This was set as a bound to the objective function for any ILP formulation. The first column in Table 4.5 shows the number of inputs (value of n). For $1 \leq n \leq 4$, we considered all possible functions. For functions with $n > 4$, the entries in Table 4.5 were obtained by sampling randomly 10 thousand functions. In particular, for $n > 4$, the 2^n bit output vector of an n-input function was filled with either 0 or 1 at randomly selected positions (determined by randomly selecting an integer mod 2^n), and so that the number of ones in the function obeyed the distribution of functions based on this property. (For example, the number of 5-input functions with 16 ones in the output bit vector is approximately 10 times more than the number of 5-input functions with 10 ones.) In order for the experiment to have more statistical significance, we only considered non-scalable functions, and when a function is generated, we applied the procedure described earlier in this section to determine that it is non-scalable. (It is asserted that the distribution of non-scalable n-input functions based on the number of ones in their output bit vector is the same as the one described earlier for n-input functions.)

The second column in Table 4.5 lists the number of k-TFs implemented by using CMOS-based method in [83] for $k = 4$. The third column shows the number of 1-mTGs by the approaches in [48, 76]. Column four lists the number of 1-mTGs with the fixed resistivity approaches in [50, 79]. Column five lists the number of k-TFs implemented by

Table 4.5. Number of n-input k-TFs Using $4n$ Transistors.

n	[83]	[48], [76]	[50], [79]	Proposed
1	2	2	2	2
2	8	8	8	10
3	72	72	72	218
4	37	1536	30	11514
5*	18	505	12	10970
6*	0	19	0	8031
7*	31	78	2	18961
8*	5	7	0	19922
9*	3449	368	210	19320
10*	1653	19	7	18998
11*	2452	35	11	19936
12*	1204	307	307	18725
TOTAL	**8931**	**2956**	**661**	**146607**

* Out of 20 thousand randomly selected non-scalable functions.

using the proposed mTG approach for $k = 4$. The transistor count of any function in Table 4.5 was limited to $4n$. For all examined functions, the value of C was set to 8% to take into consideration that memristor and transistor sizes may vary due to process variations. The results in Table 4.5 show that more functions can be implemented as single gates when weights are implemented by higher order memristive components.

The remainder of the chapter provides experimental results on 4-TFs without explicitly limiting the transistor count. Each function was implemented with [83] and the proposed memristive approach. We considered functions as in Table 4.5. For each TF we counted the number of transistors required. Table 4.6 shows that the proposed method requires significantly less transistors when compared to [83]. Each row lists results for different number of inputs.

Table 4.6. Number of 4-TFs That Can Be Implemented With Lower Transistor Count Using Proposed Approach.

n	k-TF as in [83]	Δ (percentage reduction in CTG transistor count)			
		50%<Δ<60%	60%≤Δ<75%	75%≤Δ<90%	Δ≥90%
1	2	2	0	0	0
2	10	8	2	0	0
3	218	58	84	76	0
4	65160	558	16551	47085	966
5*	19820	87	6244	13470	19
6*	20037	33	4372	15586	46
7*	19979	72	5941	13951	15
8*	19960	51	5557	14319	33
9*	19364	256	7348	11697	63
10*	20967	1008	6079	13761	119
11*	19942	50	7392	12494	6
12*	19894	70	5278	13720	826
TOTAL	**225353**	**2253**	**64848**	**156159**	**2093**

* Out of 20 thousand randomly selected non-scalable functions.

Column two in Table 4.6 lists the total number of functions examined for a given value of n. Let Δ denote the percentage reduction in TG transistor count of the proposed method over [83]. Columns three shows the number of TFs that were implemented with transistor count savings in the range 50%-60% over [83]. Column four lists the number of functions where the savings are in the range 60%-75%, column five list function with savings in the range 75%-90%, and, finally, last column gives the number of functions when the savings are no less than 90%. These columns were generated based on different ranges of Δ.

The results in Table 4.6 show that many k-TFs were implemented as proposed mTG with lower transistor count, and hence with lower area and power dissipation.

Approximately 98% of selected k-TFs were implemented with approximately 75% lower transistor count.

4.5 Conclusion

A memristive-based approach has been presented to implement many more functions as threshold logic gates with less transistor count when compared to CMOS-based implementations. Experimental results show that more than 95% of threshold functions can be implemented with approximately 75% lower transistor count, 90% lower sensor size, and 70% less power consumption.

CHAPTER 5

RELIABLE MEMRISTIVE NEURAL NETWORK

5.1 Introduction

Artificial neural networks are used in machine learning applications and intelligent systems where human intelligence is required for pattern matching, character and speech recognition, and big data management [106-110]. They consist of an input layer, an output layer and multiple hidden layers [111]. Each layer is made up of single neurons (called perceptrons) which usually perform two operations: convolution and activation. Convolution calculates the sum of inner products (multiplications) of inputs by their corresponding weights, and activation assigns the neuron output by comparing the sum of products with a predetermined threshold value [112, 113]. If the sum of the convolution is greater than the threshold weight, then the output of the neuron is logic one otherwise it is logic zero. The output of neuron j is denoted by y_j [107, 112-114]

$$y_j(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & if \quad \sum_{i=1}^{n} w_{i,j} \cdot x_i > \theta_j \\ 0 & otherwise \end{cases}, \tag{5.1}$$

Where each x_i, $i = 1, \dots, n$, is an analog external input, $w_{i,j}$ is the synaptic weight corresponding to the i^{th} input of j^{th} neuron, and θ_j is the threshold weight (threshold value) of neuron j. A training process achieves the synapse weight set of each neuron [107, 114].

Neuron operations are inherently parallel and are typically performed on graphic processing units (GPUs) [115-118]. However, all neural network GPU-based

implementations methods on GPUs are very power hungry. Many neuromorphic architectures have been proposed recently to reduce the power dissipation, and increase performance.

Approaches in [110-111, 118], among others, present efficient architectures using stochastic computing. The stochastic binary hybrid design in [119] splits the computation between different domains which can be used efficiently in near-sensor neural network applications. An energy/performance efficient technique on general purpose GPU architectures was proposed in [120]. It utilizes content addressable memory blocks in order to store highly frequent patterns and precomputed results.

Memristors have been used recently to implement the synaptic weights [109, 112, 121-122]. The programmable resistance value of a memristor is called its memristance, and the range of memristance is used to define different logic values and intermediate states. Memristors are typically placed very densely and are accessed using a crossbar array architecture. The crossbar architecture consists of two perpendicular sets of wires. There is one memristor at the intersection of each vertical and horizontal lines which are called the column and row, respectively. Each memristor is isolated from the others by a transistor connects in series to it. This prevents parallel formation of unwanted paths (called sneak-paths) in the crossbar architecture which may cause errors during read and write operations [97]. A memristor with its isolating transistor is called a cell. Each cell is accessed individually using row and column decoders.

Figure 5.1 shows the MCA for feedforward NN as proposed in [126]. It consists of n rows and $2m$ columns. Each column pair in MCA consists of a low power interface module that generates the total synaptic current. This eliminates the additional voltage

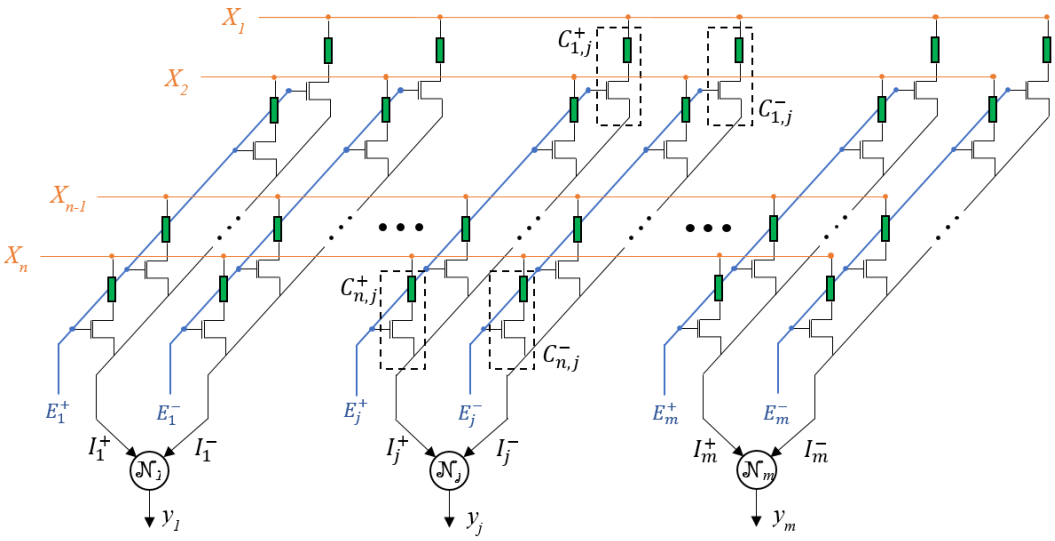


Figure 5.1. Memristive crossbar array for feedforward NN as in [126] and the interface modules.

converter required at each input of each layer as in [109, 124], therefore, it reduces power and hardware overhead.

In Figure 5.1, the input is an array of n real numbers X_i, $1 \leq i \leq n$. All the numbers are in the range $[0, 1]$. The output of the j^{th} neuron y_i, $1 \leq j \leq m$ is a real value, and it is input to the next layer. Each synaptic weight $W_{i,j}$ linking the i^{th} input and the j^{th} neuron consists of two adjacent memristors $M^+_{i,j}$ and $M^-_{i,j}$ in cells $C^+_{i,j}$ and $C^-_{i,j}$, respectively. Only one of these two memristors is in the off-state (highest possible resistivity). For instance, considering the j^{th} column pair, if the weight is positive, $M^+_{i,j}$ is programmed to the specific weight value, and $M^-_{i,j}$ is in the off-state. However, if the weight is negative, $M^-_{i,j}$ is programmed to the weight value, and $M^+_{i,j}$ is in the off-state. Each column in the crossbar array of Figure 5.1 calculates the partial weighted sum of either positive or negative convolutions and in part of a neuron. Let I^+_j and I^-_j denote the synaptic current for the positive and negative convolutions, respectively, in the j^{th} column pair that is part of the

j^{th} neuron. The difference between two I_j^+ and I_j^- is calculated by the interface module (IM). This is the j^{th} total synaptic current.

It is observed that analog-based ANNs may result in erroneous computations due to transistor aging. In fact, the performance of the CMOS transistor in a cell C is impacted by aging. In particular, Bias Temperature Instability (BTI), and to a lesser extent, dielectric breakdown as well as Hot Carrier Injections (HCI) shift the threshold voltage of the CMOS transistor causing the reduction in the drain current [97, 109, 119-122]. Therefore, the synaptic current reduces as CMOS component of the cell ages. This impacts the value of each convolution. We call this side effect as the cell aging effect. It is experimentally shown that cell aging impacts the computational accuracy of analog ANNs.

This chapter presents a solution to this problem. This chapter enhances MCA columns with a calibration circuit to alleviate the cell-aging effect and maintain invariant sum of synaptic currents. The proposed approach uses a built-in current-based calibration circuit (CC) to restore the total synaptic (column) current.

It should be mentioned that memristive leakage may also impact the value of convolutions. Techniques as in [98, 99], as well as the modifications of learning methods in [123, 125] can handle the memristor leakage by periodically updating the weight values, and, therefore memristive leakage is not the focus of this chapter.

This chapter is organized as follows. Section 5.2 describes the current tuning mechanism that mitigate the cell aging effect. Section 5.3 provides with experimental results that show that cell aging may impact the reliability of crossbar-based neuromorphic applications. It also presents the experimental evaluation of the proposed architecture. Section 5.4 concludes the paper.

5.2 Enhanced architecture for improved reliability

This section introduces an enhanced MCA based ANN architecture to improve reliability due to the cell aging effect. The MCA is enhanced by an extra row (the calibration row) and an extra column (the spare column). A built-in current-based calibration circuit (CC) is introduced to restore the total synaptic current. The CC is a current sensor that receives the ideal reference current for non-aged column and restores the reduced sensed current at each column to the ideal value. Figure 5.2 shows the enhanced MCA.

We describe how the current is restored one column at a time. The column has $n+1$ cells: n cells that implement weights and one spare cell (calibration cell) for calibrating the current that is set initially in a high resistive state. Current calibration is done in three cycles. During the first cycle, all cells in the spare column are programmed sequentially to the same weight values as in target column. This is done by using any programming approach as in [98, 99]. During the second cycle, the target column is

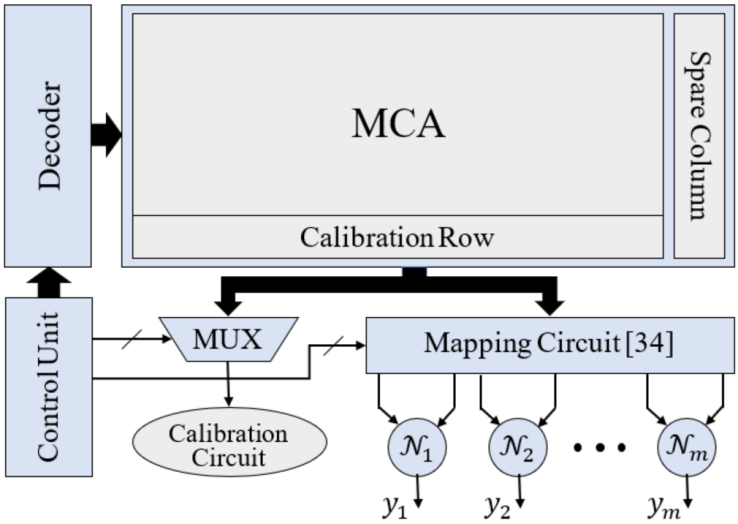


Figure 5.2. Enhanced ANN architecture including control and mapping unit for isolating the target column.

disconnected from the MCA, and the spare column is mapped to its address to ensure correct ANN functionality. The replacement and mapping process uses a built-in hardware, which implements the repair algorithm during the memory testing [127]. The target column is now disconnected from its neuron. During the last cycle, the restore operation applies to calibration cell in the target column. The memristance of this cell automatically assigns to a value so that the total current of the target column reflects the reference current. The restore operation in target column is online, and therefore, it does not interrupt the normal operation of the circuit.

The built-in current-based CC is shown in Figure 5.3. It contains a current sensor and a feedback loop. When restoring the current, the target column C_j^+ is connected to the current sensor, a low read voltage V_r is applied to the n cells that implement weights, and a high write voltage V_w is applied to the calibration cell. The current sensor compares the resulting current I_j^+ with a predetermined reference current $I_{j\_ref}^+$. If I_j^+ is less than $I_{j\_ref}^+$, the output of the sensor is set to logic 0. During this time, I_j^+ increases as the calibration memristance decreases over time. The memristance continues to decrease until I_j^+ equals $I_{j\_ref}^+$. At that time, the output of the sensor changes to logic 1, and the feedback loop deactivates the restore operation. The latch in the feedback loop ensures stable operation.

The current sensor contains a minimum size buffer, a sensor resistivity R_S, and an external sensor voltage V_S. Voltage V_S in current sensor is controlled externally, and the resistor R_S can be implemented either by a parallel network of resistors or by an additional memristor which can be programmed to any resistive value. During the normal mode of operation and the first two calibration cycles, enable signal E_r is low; subsequently, the

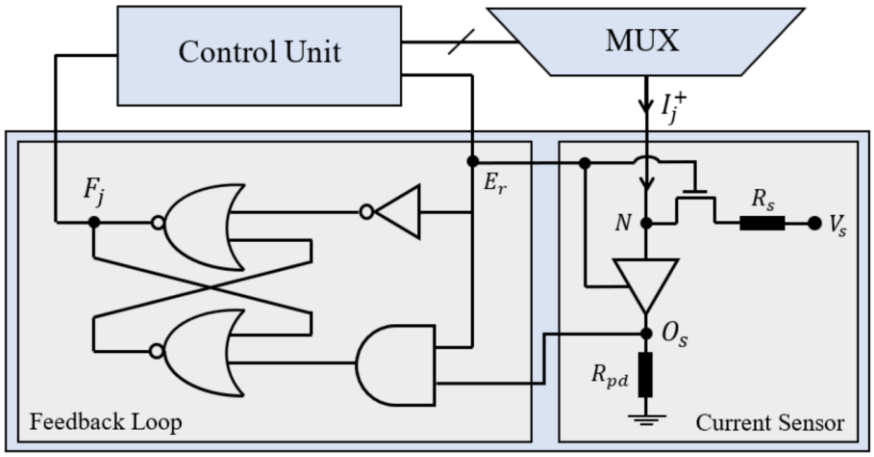


Figure 5.3. The circuitry of the proposed current-based calibration circuit.

current sensor is deactivated, and the output of the sensor (O_s) has low voltage because of the pull-down resistivity R_{pd}. During the restore operation (the third cycle), the control signal unit connects the target column to the current sensor through a multiplexor. In addition, the enable signal E_r is high, and therefore, the sensor is active. When the restore operation starts, the voltage at node N (V_N) is expressed as $V_N = V_S + R_S \cdot I_j^+$, and increases because I_j^+ increases. The calibration memristance decreases continuously until I_j^+ equals $I_{j\_ref}^+$. At this point, V_N will be larger than half of bias voltage of the buffer, and therefore the output of the buffer changes to high voltage.

When designing the sensor, the sensor resistivity R_S and the voltages V_S and V_r are determined so that $V_N = \frac{V_{DD}}{2}$ when $I_j(t) = I_{j\_ref}^+$, and by considering V_{DD} and ground as bias voltages for the buffer. The value of $I_{j\_ref}^+$ is determined by SPICE simulation. Different R_S and V_S values can be used to sense various currents. (Note that $V_S < 0.5 < V_r < V_w$.) The precision in restoring the total current is directly proportional to the value of R_s. Delay and the noise sensor margins are determined by the delay and noise margins

of the buffer. Its noise margin also specifies the maximum number of cycles between every restore operation.

The feedback loop in Figure 5.3 contains a two input AND gate, an inverter, and an edge-triggered latch. During the normal mode of operation, $E_r = 0$, and the feedback signal F_j as well as the output of the sensor block O_S are initialized to 0. This operation is kept to be active until the output of sensor block changes to 1 (which means that the current I_j^+ of the cell C_j^+ is tuned). Then after, the signal F_j changes to 1 immediately to deactivate the restore operation through the control unit. The current I_j^+ subsequently reduces to zero and forces O_S to return immediately to 0. However, the Signal F_j will be 1 until E_r being disable.

Figure 5.4 shows the resistivity $M_{n+1,j}$, current I_j^+ transitions, the pulse generated in the output of the current sensor block O_S, and the output of the feedback loop F_j during restore operation applied to column C_j^+. As shown in Fig 5.4, O_S changes to '1', when the current I_j^+ is restored to the predetermined reference current. It changes again to '0' when the feedback loop disconnects V_r and V_w from the target calibration cell. The time difference between the time that $V_N = 0.5$ and the time that F_j changes to '1' is denoted by d_S. The time d_S depends on the total delay of the sensor block and its feedback loop. During d_S, the current increases over its reference value. This increment determines the precision of the proposed current-based calibration circuit. The time d_S depends strongly on the size of the implemented approach, which is fixed after post-silicon fabrication. However, the precision can be justified by I_j^+ transition rate during the restore operation, which in turn, relates to voltages V_w, and V_S. The accuracy increases with decrease in $V_w - V_S$.

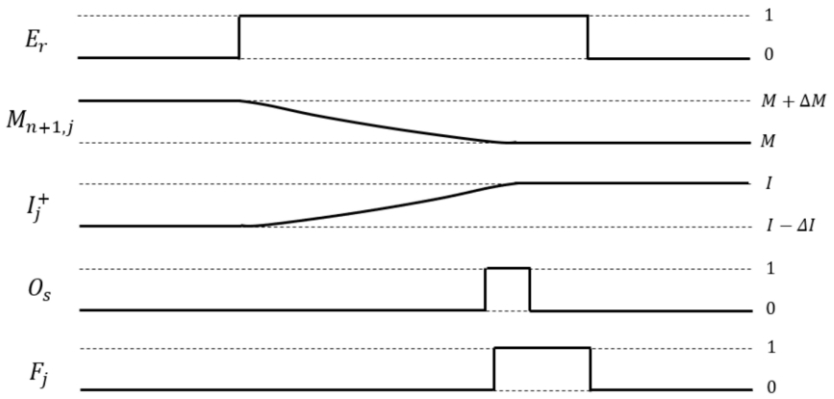


Figure 5.4. Timing diagram for signals E_r, O_s, and F_j during the restore operation for target column C_j^+.

The presented MCA enhancement may also reduce the time required to train the aged MCA for a new application by slightly modifying existing hardware-based training methods [123, 125]. Essentially, it maps any assigned memristor to appropriate value for aged product. Experimental evidence for the training speedup will be provided in future work. Finally, it is noted that aged transistors in the neurons *N*, the control unit, the decoder, and the column isolating circuit only impact the temporal characteristic of the components, and therefore, do not result in misclassification. In particular, it impacts d_s and hence, reduces tuning precision. However, this can be handled by assigning appropriate values to R_s and V_s.

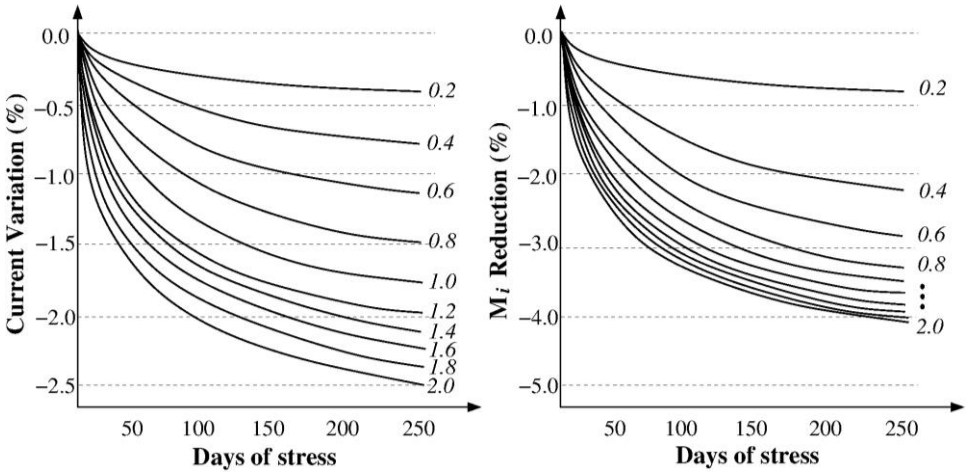
5.3 Experimental results

This section shows how CMOS aging effects the reliability of a memrisive-based ANN, and therefore, the proposed approach helps to mitigate the aging effect. We estimated the cell aging effect by assuming that the transistors were continuously under stress. We used the static aging model in [84] to implement aged transistors by assigning

different threshold voltages. The VTEAM model [15] for TiO_2 bipolar metal-oxide memristors was used during simulation. The length and memristance boundaries (R_{ON} and R_{OFF}) were set to 5nm, 5KΩ, and 5MΩ, respectively. Other memristor parameters were set as in [35]. Switching time for such memristor is approximately 10μs using applied voltage $\pm 1V$.

Memristance $M_{i,j}$ at cell $C_{i,j}$ was implemented with a 5-bit memristive multi-level cell [99, 128-130], where more than one bit of information can be implemented in a single cell with various levels of memristance. Since the current-voltage relation of a memristor is nonlinear, each level corresponding to a weight value was assigned using the approach in [129]. Any level or weight value can be realized by changing the memristance of the memristor gradually with a precise write control [99]. We used five different levels to implement 32 weight values.

Figure 5.5 (a) shows current change over time for a single cell with different weight values, while considering only the threshold voltage increase over time. Figure 5.5 (b) shows how the proposed built-in current-based calibration circuit (CC) restore the current by using the calibration cell. This experiment assumes that a column has only two rows (one is for weight implementation, and the other is for calibration). The calibration resistivity $M_{i,j}$ decreases systematically over time to compensate the current reduction of the aged cell and alleviate the aging effect. The values in Figure 5.5 were obtained considering 3% width and length variations for transistors [12, 105, 131-132]. The delay of the approach, d_S, is less than 130ps, and each current listed was restored at most within 98.8% of its initial value when V_w and V_S were set to $1V$ and $0V$, respectively. However, the precision increases as $V_w - V_S$ decreases. Additional experiments showed



(a) (b)

Figure 5.5. (a) The current degradation over time due to aging, and (b) the calibration resistivity M_j changes over time to compensate the current degradation. Ten different weight values are considered.

that the currents (weights) of Figure 5.5 were restored within 99.2% of their initial values when $V_w = 0.8V$ and $V_S = 0.2V$.

Figure 5.6 shows how weights change due to transistor aging for various input voltage x_i without considering memristive leakage. Only four levels of weights were considered during this analysis. Aging factor depends on the total time that transistors of the cells are active which in turn, depends on the total number of testing samples. However, in neural network applications, the impact of aging on current variation for each cell increases exponentially when the input voltage decreases. This is shown in Figure 5.6. As an example, for a high weight value $W_{i,j}$ with $25mV$ threshold voltage increment, the current variation was more than 40% when input $x_i = 0.5V$ while it reduced to less than 2% when $x_i = 1V$.

The following presents experimental evidence on the impact of the proposed

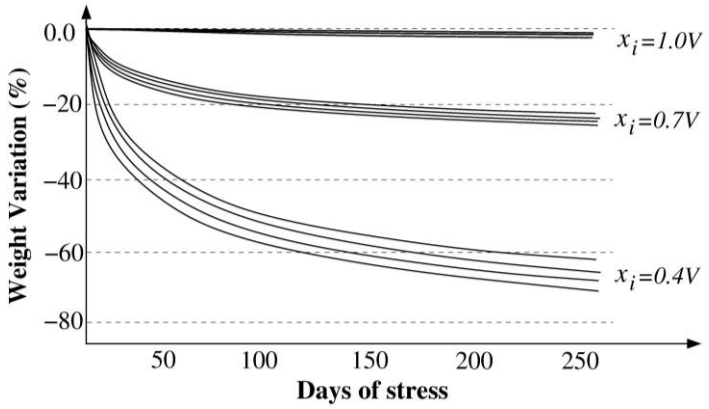
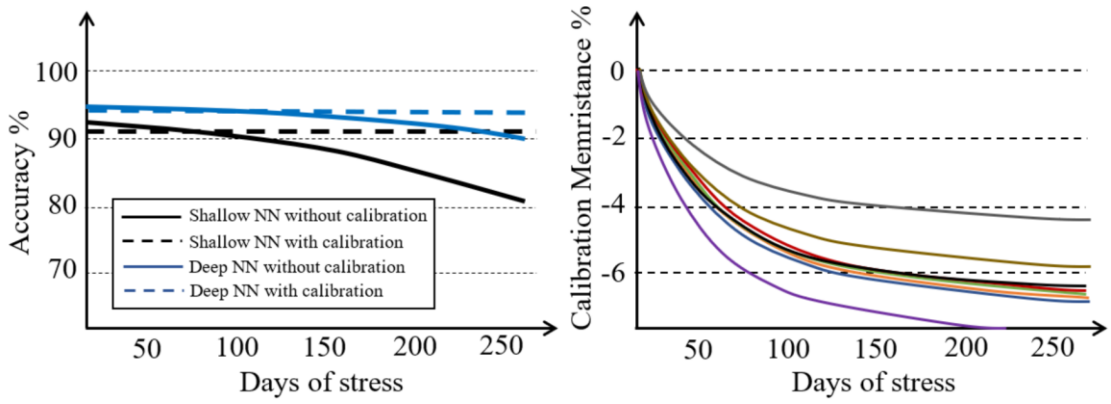


Figure 5.6. The weights change due to transistor aging for various input voltage x_i.

approach in pattern recognition. We evaluated the impact of CMOS-component aging on synaptic weight precisions. The accuracy and the hardware overhead of the proposed current-based CC were investigated on a handwritten digit recognition task using the neural network based MNIST database [111, 120].

We implemented two ANN; one was a shallow network with a single hidden layer, and the other one was a deep network with four hidden layers. We considered MNIST with 784 input neurons, 10 output neurons, 6000 training samples, and 1000 testing samples. We used 300 epochs (maximum number of iterations) for training. Learning rate, epsilon, and momentum were set to 0.001, 0.001, and 0.9, respectively. We used sigmoid function $f(x) = 1/(1 + \exp(-x))$ as an activation function, and we introduced a threshold value so that if the output of the winner neuron in output layer is within the 10% of the expected value, the output is correct and input sample is classified. Otherwise, the sample cannot be classified. Furthermore, the size of testing samples were 28×28, and each pixel was presented by either 0 or 1. We made the analog input for each pixel by getting the average of the pixel binary value and its eight neighbors. Therefore, the input voltages



(a) **(b)**

Figure 5.7 (a) The accuracy over days of stress of the implemented shallow and deep neural networks with and without using the built-in calibration circuit, and (b) calibration memristance for each column in output layer over days of stress to compensate the aging effect.

were set in 11 levels in the range from 0V to 1V.

The accuracy over days of stress on fully connected ANN networks with and without using the calibration circuit is shown in Figure 5.7 (a). The accuracy was determined by dividing the number of correct pattern recognized by the total number of testing samples. Results show a significant loss of accuracy due to aging, while it has no effect when using the proposed current calibration circuit (CC) along with crossbar array. This is due to the periodic adjustment of the memristance in the target column over days of stress. Figure 5.7 (b) shows that the calibration memristance of the output layer (10 columns) reduces over time in order to compensate cell-aging effect.

Figure 5.8 shows three testing samples from MNIST data set that were predicted correctly with less than 0.01% mean square error (MSE) at time $t = 0$. However, considering continuous stress on transistors, the neural network predicts an incorrect value for each sample after 28 minutes. Therefore, the samples were misclassified and the MSE was more than 30%.

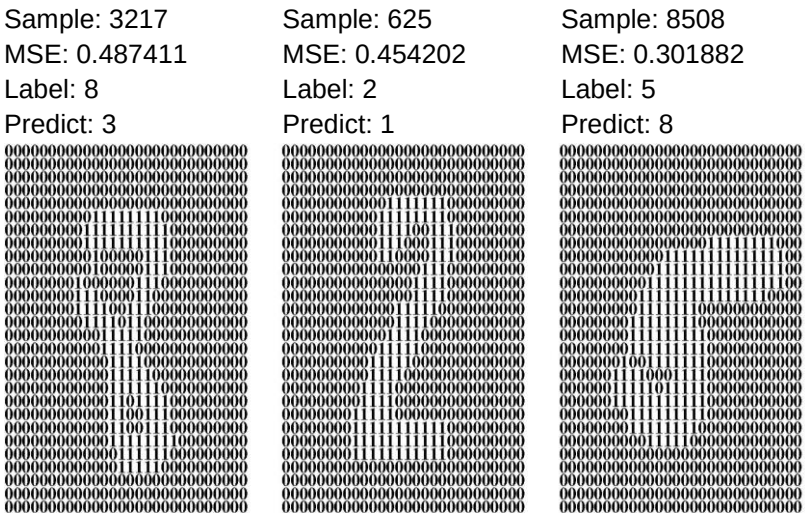


Figure 5.8. Aging effect in correctness of pattern recognition for three different samples.

Using the proposed current tuning mechanism of Section 5.2, the current can be tuned with a high precision, which relates to restore voltage V_w and external voltage V_S. SPICE simulation of an input neuron in presence of 3% variation in transistor, shows that any synaptic current in both implemented ANNs can be tuned with less than 1% precision (within 99% of the initial value) when $V_w = 1V$ and $V_s = 0V$. This precision should be considered during the weight training to assign weights so that the functionality of the implemented network holds.

The proposed current-based calibration circuit was implemented in 45nm technology with 19 transistors and 2 resistors. The area of the designed sensor was found to be only 4.36 μm^2. To find the winner neuron at the output layer, we implemented the 5-bit winner-take-all (WTA) circuit with 10 inputs proposed in [109] within 0.17 mm^2. The area overhead of the designed approach was around 0.043%. In addition, the total power dissipation during the calibration operation (3 cycles) to tune the current of the target column 97μW. The Overall increase in total power is 0.088%.

5.4 Conclusion

A reliable and low power built-in current-based calibration circuit is proposed to periodically restore each neuron's current when it reduces due to aging. Experimental results show a significant saving in both shallow and deep neural networks while the area overhead to ensure reliable operation is negligible.

CHAPTER 6

CONCLUDING REMARKS

Nano scale devices such as memristors are prone to defects. Some defects impact the logical behavior of the device. Others impact the temporal behavior. A methodology for testing the hybrid crossbar architecture has been presented in Chapter 2. The new approach used a new fast write operation in order to reduce the test application time. The proposed fast write operation benefits from the behavior of memristor device which is nonlinear and asymmetric. The random memristive behavior and sneak-paths in crossbar memory were also taken into consideration.

A new Design for Testability (DfT) mechanism was proposed in Chapter 2 to implement the proposed fast write operation. The programmable DfT was able to assign different access times for the proposed fast operations. The total area overhead of the proposed DfT depends only on the number of columns and does not depend on the number of rows and the number of tiles. The experimental results in 45nm technology on DfT implementation showed that the area overhead of the designed DfT was found to be only 8.875μm^2 per column. Moreover, experimental results showed that the approach reduced the test application time by 70% and the test energy by 40%. Also, the method had similar test application time when sneak paths were considered during test in order to increase reliability.

The proposed methods have been presented assuming that each memory cell is a bipolar metal-oxide memristor which is a popular technology. The methods can be generalized to other types of memristors as long as they have nonlinear and asymmetric

characteristics in the switching parameters. Future work of Chapter 1 will investigate on online testing of memristor-based memories.

Threshold logic functions (TFs) and their implementations have been investigated in Chapter 3. A small fraction of Boolean functions are TFs and can be implemented as a single gate. This limits the impact of threshold logic gates (TGs) in digital circuit synthesis. Chapter 3 proposed a new method to implement efficiently more functions as single TG. This has been done by introducing higher order non-integer weights. The method benefits from the higher order definition of TF and tries to control the number of non-zero weight components in order to reduce the TG transistor count.

It has been demonstrated that the presented approach can implement many more functions as current mode TGs (CTGs) with similar or less transistor count when compared to existing method. In particular, for 100 thousand randomly selected functions, when considering up to 4^{th}-order weights, about 24.9 times more functions can be implemented as CTGs with similar or less transistor count. Also around 90% of existing TFs can be implemented as CTGs with approximately 60% less power dissipation, and 20% less delay when considering higher order non-integer weights in the presence of circuit parasitics. Future work of Chapter 3 will investigate heuristic approaches to implement higher order TFs with rational weights.

In addition, Chapter 4 investigated the impact of emerging technology on resistive devices such as memristors. A memristive-based approach has been presented to implement many more functions as threshold logic gates with less transistor count when compared to CMOS-based implementations in Chapter 3. The resistivity range of memristor was used to define different weight values.

Experimental results showed that the transistor count was reduced further when implementing the weights (including 1st–order and higher order weights) with resistive devices. This method of weight implementation reduced significantly the transistor count of the TG. Experimental results showed that more than 95% of threshold functions can be implemented with approximately 75% lower transistor count, 90% lower sensor size, and 70% less power consumption.

Chapter 5 focused on reliability of analog artificial neural network (ANN) where synaptic weights were implemented by memristor. It was observed that analog ANNs may result in erroneous computations due to transistor aging. In particular, it was shown that aging impacts the value of each multiplication. A new method to improve reliability was proposed in Chapter 5. The approach benefits from the enhanced memristive crossbar array (MCA) which contains an extra row (the calibration row) and an extra column (the spare column). A built-in current-based calibration circuit was designed to restore the total synaptic weight.

Experimental results on the proposed calibration circuit in 45nm technology showed that the currents of aged synapses (weights) were restored within approximately 90% of their initial values. Furthermore, the results showed that the area overhead of the designed circuit was around 0.043% and the total increase in power dissipation due to the calibration operation was 0.088%. Additional experiments on MNIST dataset showed a significant saving in both shallow and deep neural networks. Future work of Chapter 5 will investigate on tuning of activation functions rather than multiplications.

REFERENCES

[1] L. O. Chua, “Memristor—The Missing Circuit Element,” *IEEE Transactions on Circuit Theory*, vol.18, no. 5, pp. 507-519, 1971.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, R. S. Williams, “The missing Memristor found,” *Nature*, 453, pp. 80-83, 2008.

[3] Y. Ho, G. M. Huang, P. Li, “Nonvolatile memristor memory: device characteristics and design implications,” *IEEE/ACM International Conference on Computer-Aided Design*, pp. 485-490, 2009.

[4] M. Imani, A. Rahimi, T. Rosing, “Resistive Configurable Associative Memory for Approximate Computing,” *Proceeding of IEEE Design, Automation and Test in Europe (DATE),* pp. 1327-1332, 2016.

[5] S. Balatti, S. Ambrogio, Z. Wang, D. Ielmini, “True Random Number Generation by Variability of Resistive Switching in Oxide-Based Devices,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 2, pp. 214-221, 2015.

[6] P. Knag, Wei Lu; Zhengya Zhang, “A Native Stochastic Computing Architecture Enabled by Memristors,” *IEEE Transactions on Nanotechnology*, vol. 13, no. 2, pp. 283-293, 2014.

[7] J. Hutchby, M. Garner, “Assessment of the potential and maturity of selected emerging research memory technologies”, *Workshop and ERD/ERM Working Group Meeting, Japan*, 2010.

[8] M. Imani, P. Mercati, T. Rosing, “ReMAM: Low energy Resistive Multi-stage Associative Memory for energy efficient computing,” *International Symposium on Quality Electronic Design (ISQED)*, pp. 101-106, 2016.

[9] M. Imani, Shruti Patil, T. Rosing, “Approximate Computing using Multiple-Access Single-Charge Associative Memory,” *IEEE Transaction on Emerging Topics in Computing (TETC)*,

2016.

[10] M. Imani, Y. Kim, A. Rahimi, T. Rosing, “ACAM: Approximate Computing Based on Adaptive Associative Memory with Online Learning.” *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 162-167, 2016.

[11] N. Z. Haron, S. Hamdioui, “On Defect Oriented Testing for Hybrid CMOS/Memristor Memory,” *IEEE Asian Test Symposium*, pp. 353-358, 2011.

[12] S. N. Mozaffari, A. Afzali-Kusha, “Statistical Model for Subthreshold Current Considering Process Variations,” *2nd Asia Symposium on Quality Electronic Design (ASQED)*, pp. 356-360, 2010.

[13] M. Bushnell, V. Agrawal, “Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits,” *Springer* 2000.

[14] S. Kvatinsky, E. G. Friedman, A. Kolodny, U. C. Weiser, “TEAM: ThrEshold Adaptive Memristor Model,” *IEEE Transactions on Circuits and Systems—I: Regular Papers*, vol. 60, no. 1, pp. 211-221, 2013.

[15] S. Kvatinsky, M. Ramadan, E. G. Friedman, A. Kolodny, “VTEAM-A General Model for Voltage Controlled Memristors,” *IEEE Transactions on Circuits and Systems—II: Express Briefs*, vol. 62, no. 8, pp. 786-790, 2015.

[16] S. Gaba, P. Sheridan, J. Zhou, S. Choi, W. Lu, “Stochastic memristive devices for computing and neuromorphic applications,” *Nanoscale*, vol. 5, no. 13, pp. 5872-5878, 2013.

[17] Gaba, S., Knag, P., Zhang, Z., Wei Lu, “Memristive devices for stochastic computing,” *IEEE International Symposium on Circuits and Systems (ISCAS),* pp. 2592-2595, 2014.

[18] S. Kannan, J. Rajendran, R. Karri, O. Sinanoglu, “Sneak-path Testing of Memristor-based Memories,” *International Conference on VLSI Design*, pp. 386-391, 2013.

[19] S. Kannan, J. Rajendran, R.Karri, O. Sinanoglu, “Sneak-Path Testing of Crossbar-Based Nonvolatile Random Access Memories,” *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 413-426, 2013.

[20] S. Kannan, N. Karimi, R. Karri, O. Sinanoglu, “Detection, Diagnosis, and Repair of Faults in Memristor-based Memories,” *IEEE VLSI Test* Symposium (VTS), pp. 1-6, 2014.

[21] S. Kannan, R. Karri, O. Sinanoglu, “Sneak path testing and Fault Modeling for Multi-level Memristor-based Memories,” *IEEE International Conference on Computer Design (ICCD)*, pp. 215-220, 2013.

[22] S. Kannan, N. Karimi, R. Karri, O. Sinanoglu, “Modeling, Detection, and Diagnosis of Faults in Multi-Level Memristor Memories,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 822-824, 2015.

[23] L. Dillilo, P. Girard, S. Pravossoudovitch, A. Virazel, “Resistive-Open Defects in Embedded-SRAM core cell: Analysis and March Test Solution,” *IEEE Test Symposium*, pp. 266-271, 2004.

[24] S. Hamdioui, M. Taouil, N. Z. Haron, “Testing Open Defects in Memristor-Based Memories,” *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 247-259, 2015.

[25] N. Z. Haron and S. Hamdioui, “DfT Schemes for Resistive Open Defects in RRAMs,” *IEEE Design, Automation and Test in Europe (DATE)*, pp. 799-804, 2012.

[26] S. Hamdioui, H. Aziza, G. Sirakoulis, “Memristor Based Memories: Technology, Design and Test,” *IEEE Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pp. 1-7, 2014.

[27] S. N. Mozaffari, S. Tragoudas and T. Haniotakis, “Fast march tests for defects in resistive memory,” *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 88-93, 2015.

[28] C. Yakopcic, R. Hasan, T. M. Taha, “Hybrid crossbar architecture for a memristor based cache,” *Microelectronics Journal*, vol. 46, no. 11, pp. 1020-1032, 2015.

[29] J. J. Yang et al., “The mechanism of electroforming of metal oxide memristive switches,” *Nanotechnology*, vol. 20, no. 21, 2009.

[30] A. H. Edwards, H. J. Barnaby, K. A. Campbell, M. N. Kozicki, W. Liu, M. J. Marinella,

"Reconfigurable Memristive Device Technologies," *Proceedings of the IEEE*, vol. 103, no. 7, pp. 1004-1033, 2015.

[31] M. Saremi, "A physical-based simulation for the dynamic behavior of photodoping mechanism in the chalcogenide materials used in the lateral programmable metallization cells" *Solid State Ionics*, vol. 290, p. 1, 2016.

[32] M. Saremi, H. J. Barnaby, A. Edwards, and M. N. Kozicki, "Analytical relationship between anion formation and carrier-trap statistics in chalcogenide glass films," *Electrochemistry Letters*, vol. 4, no. 7, pp. H29-H31, 2015.

[33] S. Rajabi, M. Saremi et al., "Static impedance behavior of programmable metallization cells" *Solid State Electronics*, vol. 106, p. 27, 2015.

[34] M. Hu, Y. Wang, Q. Qiu, Y. Chen, H. Li, "The stochastic modeling of TiO_2 memristor and its usage in neuromorphic system design," *IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 831-836, 2014.

[35] S. Kvatinsky, K. Talisveyberg, D. Fliter, A. Kolodny, U. C. Weiser and E. G. Friedman, "Models of memristors for SPICE simulations," *IEEE Convention of Electrical & Electronics Engineers in Israel (IEEEI),* pp. 1-5, 2012.

[36] P. Sheridan, K.-H. Kim, S. Gaba, T. Chang, L. Chen, W. Lu, "Device and SPICE modeling of RRAM devices," *Nanoscale*, vol. 3, no. 9, pp. 3833-3840, 2011.

[37] S. H. Jo, K. H. Kim, W. Lu, "Programmable resistance switching in nanoscale two-terminal devices," *Nano Letters*, vol. 9, no. 1, pp. 496-500, 2009.

[38] A. Ghofrani et al., "A Low-Power Variation-Aware Adaptive Write Scheme for Access-Transistor-Free Memristive Memory." *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 1, Article 3., 18 pages, 2015.

[39] J. J. Yang et al., "High switching endurance in TaOx memristive devices," *Applied Physics Letters*, vol. 97, no. 23, 3 pages, 2010.

[40] G. Medeiros-Ribeiro, F. Perner, R. Carter, H. Abdalla, M. D. Pickett, R. S. Williams,

"Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution," *Nanotechnology*, vol. 22, no. 9, 2011.

[41] C. Yang, B. Liu, Y. Wang, Y. Chen, H. Li, X. Zhang, G. Sun, "The Applications of NVM Technology in Hardware Security," *Proceedings of Great Lakes Symposium on VLSI (GLVLSI)*, pp. 311-316, 2016.

[42] H. Aziza, M. Bocquet, J-M. Portal, C. Muller, "Bipolar OxRRAM Memory Array Reliability Evaluation based on Fault Injection," *Proceedings of Design and Test Workshop (IDT)*, pp. 78-81, 2011.

[43] J. Wu, M. Choi, "Memristor lookup table (MLUT)-based asynchronous nanowire crossbar architecture," *IEEE Conference on Nanotechnology*, pp. 1100-1103, 2010.

[44] Y. Cassuto, S. Kvatinsky, E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays." *IEEE International Symposium on Information Theory (ISIT)*, pp. 156-160, 2013.

[45] P. P. Sotiriadis, "Information capacity of nanowire crossbar switching networks," *IEEE Transactions on Information Theory*, vol. 52, no. 7, pp. 3019-3032, 2006.

[46] S. Shin, K. Kim, S.-M. Kang, "Analysis of passive memristive devices array: Data-dependent statistical model and self-adaptable sense resistance for RRAMs," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 2021-2032, 2012.

[47] S. Gupta, A. Crouch, J. Dworak and D. Engels, "Increasing IJTAG bandwidth and managing security through parallel locking-SIBs," *IEEE International Test Conference (ITC)*, pp. 1-10, 2017.

[48] C. B. Dara, T. Haniotakis and S. Tragoudas, "Low power and high speed current-mode memristor-based TLGs," *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp.89-94, 2013.

[49] M. J. Avedillo, J. M. Quintana, A. Rueda and E. Jimenez, "Low-power CMOS threshold-logic gate," *Electronics Letters*, vol. 31, no. 25, pp. 2157-2159, 1995.

[50] J. Yang, N. Kulkarni, S. Yu and S. Vrudhula, “Integration of threshold logic gates with RRAM devices for energy efficient and robust operation,” *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 39-44, 2014.

[51] N. Kulkarni, J. Yang, J. S. Seo and S. Vrudhula, “Reducing Power, Leakage, and Area of Standard-Cell ASICs Using Threshold Logic Flip-Flops,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 9, pp. 2873-2886, 2016.

[52] C. B. Dara, T. Haniotakis and S. Tragoudas, “Delay Analysis for Current Mode Threshold Logic Gate Designs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1063-1071, 2017.

[53] S. Bobba and I. N. Hajj, “Current-mode threshold logic gates,” *Proceedings of the International Conference on Computer Design*, pp. 235-240, 2000.

[54] T. Gowda, S. Vrudhula, N. Kulkarni and K. Berezowski, “Identification of Threshold Functions and Synthesis of Threshold Networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 665-677, 2011.

[55] V. Beiu, J. M. Quintana, M. J. Avedilo and R. Andonie, “Differential implementations of threshold logic gates,” *Proceedings of the International Symposium on Signals, Circuits and Systems 2 (SCS)*, vol. 2, pp. 489-492, 2003.

[56] M. Nikodem, M. A. Bawiec and J. Biernat, “Synthesis of generalised threshold gates and multi threshold threshold gates,” *INTL Journal of Electrinics and Telecommunications*, vol. 58, no. 1, pp. 49-54, 2012.

[57] R. O. Winder, “Threshold logic,” Ph.D. Dissertation, Princeton University, Princeton, NJ, 1962.

[58] S. Muroga, “Threshold Logic and its Applications,” *John Wiley, New York*, 1971.

[59] M. L. Dertouzos, “Threshold Logic: A Synthesis Approach,” *MIT Press*, Cambridge, MA, 1965.

[60] C. Wang and A.C. Williams, “The threshold order of a Boolean function,” *Discrete Applied Mathematics*, vol. 31, no. 1, pp. 51-69, 1991.

[61] A. K. Palaniswamy, M. K. Goparaju and S. Tragoudas, “An Efficient Heuristic to Identify Threshold Logic Functions,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 8, no. 3, pp. 19:1–19:17, 2012.

[62] A. Neutzling, M. G. A. Martins, R. P. Ribas and A. I. Reis, “Synthesis of threshold logic gates to nanoelectronics,” *proceedings of the 26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 1-6, 2013.

[63] A. Neutzling, J. Maick Matos, A. I. Reis, R. P. Ribas and A. Mishchenko, “Threshold logic synthesis based on cut pruning,” *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 494-499, 2015.

[64] A. K. Palaniswamy and S. Tragoudas, “Improved Threshold Logic Synthesis Using Implicant-Implicit Algorithms,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 3, 2014.

[65] T. Gowda and S. Vrudhula, “Decomposition based approach for synthesis of multi-level threshold logic circuits,” *proceedings of the Asia and South Pacific Design Automation Conference*, pp. 125-130, 2008.

[66] A. Neutzling, M. G. A. Martins, R. P. Ribas and A. I. Reis, “A constructive approach for threshold logic circuit synthesis,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 385-388, 2014.

[67] R. Zhang, P. Gupta, L. Zhong and N. K. Jha, “Threshold network synthesis and optimization and its application to nanotechnologies,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 107-118, 2005.

[68] T. Ogawa, T. Hirose, T. Asai and Y. Amemiya “Threshold Logic Devices Consisting of Subthreshold CMOS Circuits,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92.A, no. 2, pp. 436-442, 2009.

[69] T. Gowda, S. Vrudhula, N. Kulkarni and K. Berezowski, “Identification of Threshold Functions and Synthesis of Threshold Networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 665-677, 2011.

[70] R. Zhang, P. Gupta, L. Zhong and N. K. Jha, “Synthesis and Optimization of Threshold Logic Networks with Application to Nanotechnologies,” *Proceedings of the conference on Design, Automation & Test in Europe (DATE)*, pp. 904-909, 2004.

[71] A. Mishchenko, “Enumeration of irredundant circuit structures,” *Proceedings of International Workshop on Logic and Synthesis*, 2014.

[72] J. Keane, X. Wang, D. Persaud, and C. H. Kim, “An All-In-One Silicon Odometer for Separately Monitoring HCI, BTI, and TDDB,” *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 817-829, 2010.

[73] B. Eghbalkhah, M. Kamal, A. Afzali-Kusha, M. B. Ghaznavi-Goushchi, and M. Pedram, “CSAM: A Clock Skew-aware Aging Mitigation Technique,” *Microelectronics Reliability*, vol. 55, no. 1, pp. 282-290, 2015.

[74] S. Mahapatra et al., “Characterization and modeling of NBTI stress, recovery, material dependence and AC degradation using R-D framework,” *IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, pp. 1-7, 2011.

[75] L. Gao, F. Alibart and D. B. Strukov, “Programmable CMOS/ Memristor Threshold Logic,” *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 115-119, 2013.

[76] J. Rajendran, H. Manem, R. Karri and G. S. Rose, “An Energy-Efficient Memristive Threshold Logic Circuit,” *IEEE Transactions on Computers*, vol. 61, no. 4, pp. 474-487, 2012.

[77] K. Maan, D. A. Jayadevi and A. P. James, “A Survey of Memristive Threshold Logic Circuits,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 8, pp. 1734-1746, 2017.

[78] J. M. Quintana, J. M. Avedillo, J. Nunez and H. P. Roldan, “Operation Limits for RTD-Based MOBILE Circuits,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 2, pp. 350-363, 2009.

[79] S. Vrudhula, N. Kulkami and J. Yang, “Design of threshold logic gates using emerging devices,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 373-376, 2015.

[80] N. Kimizuka et al., “The impact of bias temperature instability for direct-tunneling ultra-thin gate oxide on MOSFET scaling,” *Symposium on VLSI Technology. Digest of Technical Papers (IEEE Cat. No.99CH36325)*, pp. 73-74, 1999.

[81] S. Khan et al., “BTI impact on logical gates in nano-scale CMOS technology,” *IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 348-353, 2012.

[82] S. N. Mozaffari, S. Tragoudas, and T. Haniotakis, “Reducing power, area, and delay of threshold logic gates considering non-integer weights” *IEEE International Symposium on Circuits & Systems (ISCAS)*, pp. 1-4, 2017.

[83] S. N. Mozaffari, S. Tragoudas and T. Haniotakis, “A new method to identify threshold logic functions,” *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 934-937, 2017.

[84] W. Wang et al., “The Impact of NBTI Effect on Combinational Circuit: Modeling, Simulation, and Analysis,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 2, pp. 173-183, 2010.

[85] Berkeley Predictive Technology Models (PTM), http://ptm.asu.edu/modelcard/LP/45nm_LP.pm, 2013.

[86] S. Gupta, J. Dworak, D. Engels and A. Crouch, “Mitigating simple power analysis attacks on LSIB key logic,” *IEEE North Atlantic Test Workshop (NATW)*, pp. 1-6, 2017.

[87] S. N. Mozaffari, and S. Tragoudas, “Memristive Current Mode Threshold Logic Gates,” *International Conference on Memristive Materials, Devices & Systems (MEMRISYS 2017)*, April 2017.

[88] J. Rajendran, H. Manem, and G. S. Rose, “NDR based threshold logic fabric with memristive synapses,” *IEEE Conference on Nanotechnology (IEEE-NANO)*, pp. 725–728, 2009.

[89] J. Rajendran, H. Manem, R. Karri, and G. S. Rose, “Memristor based programmable threshold logic array,” *IEEE/ACM International Symposym Nanoscale Architecture (NANOARCH)*, pp. 5-10, 2010.

[90] M. Sharad, D. Fan, and K. Roy, “Ultra-low energy, high-performance dynamic resistive threshold logic,” 2013, arXiv:1308.4672v1

[91] A. P. James, D. S. Kumar, and A. Ajayan, “Threshold logic computing: Memristive-CMOS circuits for fast fourier transform and vedic multiplication,” *IEEE Transactions on Very Large Scale Integration Sestems (TVLSI)*, vol. 23, no. 11, pp. 2690-2694, 2015.

[92] A. Rothenbuhler, T. Tran, E. H. B. Smith, V. Saxena, and K. A. Campbell, “Reconfigurable Threshold Logic Gates using Memristive Devices” *Journal of Low Power Electronics and Applications*, vol. 3, no. 2, pp. 174-193, 2013.

[93] L. Gao, F. Alibart, D. B. Strukov, “Programmable CMOS/Memristor Threshold Logic,” *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 115-119, 2013.

[94] I. Messaris, A. Serb, A. Khiat, S. Nikolaidis, and T. Prodromakis, “A compact Verilog-A ReRAM switching model,” arXiv:1703.01167.

[95] I. Messaris et al., “A TiO_2 ReRAM parameter extraction method,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-4, 2017.

[96] I. Messaris et al., “Live demonstration: A TiO2 ReRAM parameter extraction method,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-1, 2017.

[97] S. N. Mozaffari, S. Tragoudas, and T. Haniotakis, “More Efficient Testing of Metal-oxide Memristor-based Memory”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 6, pp. 1018-1029, 2017.

[98] H. Manem, J. Rajendran, and G. S. Rose, “Design considerations for multi-level CMOS/nano memristive memory,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 8, no. 1, p.1-22, 2012.

[99] F. Alibart, L. Gao1, B. D. Hoskins, and D. B. Strukov, “High precision tuning of state for memristive devicesby adaptable variation-tolerant algorithm,” *Nanotechnology*, vol. 23, no. 7, pp. 1-7, 2012.

[100] H. Manem et al., “Stochastic Gradient Descent Inspired Training Technique for a CMOS/Nano Memristive Trainable Threshold Gate Array,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 5, pp. 1051-1060, 2012.

[101] R. Berdan, T. Prodromakis, and C. Toumazou, “High precision analogue memristor state tuning,” *Electronics Letters*, vol. 48, no. 18, pp. 1105-1107, 2012.

[102] S. D. Gupta and M. A. Thornton, “A Fixed-Point Squaring Algorithm Using an Implicit Arbitrary Radix Number System,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 1, pp. 34-43, March 2016.

[103] Q. Xie, A. Yekkehkhany, and Y. Lu, “Scheduling with multi-level data locality: Throughput and heavy-traffic optimality,” *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1-9, 2016.

[104] S. N. Mozaffari, S. Tragoudas, and T. Haniotakis, “A Generalized Approach to Implement Efficient CMOS-Based Threshold Logic Functions,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 3, pp. 946-959, 2017.

[105] M. Hosseini, Y. Jiang, A. Yekkehkhany, R. R. Berlin, L. Sha, “A Mobile Geo-Communication Dataset for Physiology-Aware DASH in Rural Ambulance Transport,” *Proceedings of the 8th ACM on Multimedia Systems Conference*, pp. 158-163, 2017.

[106] J. Grollier, D. Querlioz, and M. D. Stiles, “Spintronic Nanodevices for Bioinspired Computing,” *Proceedings of the IEEE*, vol. 104, no. 10, pp. 2024-2039, 2016.

[107] A. Sengupata et al., “Proposal for an All-Spin Artificial Neural Network: Emulating Neural and Synaptic Functionalities Through Domain Wall Motion in Ferromagnets,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, no. 6, pp. 1152-1160, 2016.

[108] Y. V. Pershin, and M. D. Ventra, “Experimental Demonstration of Associative Memory With memristive Neural Network,” *Journal of Neural Networks*, vol. 23, no. 7, pp. 881-886, 2010.

[109] M. Sharad, D. Fan, K. Aitken, and K. Roy, “Energy-Efficient Non-Boolean Computing With Spin Neurons and Resistive Memory,” *IEEE Transactions on Nanotechnology*, vol. 13, no. 1, pp. 23-34, 2014.

[110] K. Kyounghoon, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, “Dynamic Energy- Accuracy Trade-off Using Stochastic Computing in Deep Neural Networks,” *IEEE Design Automation Conference (DAC)*, pp. 1-6, 2016.

[111] Z. Li, A. Ren, J. LI, Q. Qiu, B. Yuan, J. Draper, and Y. Wang, “Structural design optimization for deep convolutional Neural Network using stochastic computing,” *Design, Automation and Test in Europe (DATE)*, to appear in March 2017.

[112] K. Yogendra, D. Fan, and K. Roy, “Coupled Spin Torque Nano Oscillators for Low Power Neural Computation,” *IEEE Transactions on Magnets*, vol. 51, no. 10, pp. 1-9, 2015.

[113] W. Qinruo, Y. Bo, X. Yun, and L. Bingru, “The Hardware Structure of Perceptron with FPGA Implementation,” *IEEE International Conference on Systems, Man and Cybernetics*, pp. 762-767, 2003.

[114] M. Sharad, C. Augustine, G. Panagopoulos, and K. Roy, “Spin-Based Neuron Model With Domain-Wall Magnets as Synapse,” *IEEE Transactions on Nanotechnology*, vol. 11, no. 4, pp. 843-853, 2012.

[115] T. Harada, “Real-time rigid body simulation on GPU (chapter 29) in GPU Gems 3”, Addison Wesley, 2008.

[116] A. Lefohn, J. Kniss and J. Owens, “Implementing efficient parallel data structures on GPUs (chapter 32) in GPU Gems 2”, Addison Wesley, 2006.

[117] X. Sierra-Canto, F. Madera-Ramirez and V. Uc-Cetina, “Parallel Training of a Back-Propagation Neural Network Using CUDA,” *International Conference on Machine Learning and Applications*, pp. 307-312, 2010.

[118] R. Uetz, and S. Behnke, “Large-scale object recognition with CUDA-accelerated hierarchical neural networks,” *IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS)*, pp. 536-541, 2009.

[119] V. Lee, A. Alaghi, J. Hayes, V. Sathe, and L. Ceze, “Energy-Efficient Hybrid Stochastic-Binary Neural Networks for Near-Sensor Computing,” *Design, Automation and Test in Europe (DATE)*, pp. 13-18, 2017.

[120] M. Imani, D. Peroni, Y. Kim, A. Rahimi, and T. Rosing, “Efficient Neural Network Acceleration on GPGPU using Content Addressable Memory” *Design, Automation and Test in Europe (DATE)*, pp. 1026-1031, 2017.

[121] L. Gao, F. Alibart, and D. B. Strukov, “Analog-input analog-weight dot-product operation with Ag/a-Si/Pt memristive devices,” *IEEE/IFIP International Conference on VLSI and System-on-Chip (VLSI-SoC)*, pp. 88-93, 2012.

[122] A. Sengupta, M. Parsa, B. Han, and K. Roy, “Probabilistic Deep Spiking Neural Systems Enabled by Magnetic Tunnel Junction,” *IEEE Transactions on Electron Devices*, vol. 63, no. 7, pp. 2963-2970, 2016.

[123] S. P. Adhikari, H. Kim, R. K. Budhathoki, C. Yang and L. O. Chua, “A Circuit-Based Learning Architecture for Multilayer Neural Networks With Memristor Bridge Synapses,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 215-223, 2015.

[124] A. Sengupta and K. Roy, “A Vision for All-Spin Neural Networks: A Device to System Perspective,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2267-2277, 2016.

[125] M. V. Nair and P. Dudek, “Gradient-descent-based learning in memristive crossbar arrays,” *International Joint Conference on Neural Networks (IJCNN)*, pp. 1-7, 2015.

[126] O. Bichler, M. Suri, D. Querlioz, D. Vuillaume, B. DeSalvo and C. Gamrat, “Visual Pattern Extraction Using Energy-Efficient 2-PCM Synapse Neuromorphic Architecture,” *IEEE Transactions on Electron Devices*, vol. 59, no. 8, pp. 2206-2214, 2012.

[127] M. T. Rab, A. A. Bawa, N. A. Touba, “Improving Memory Repair by Selective Row Partitioning” *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 211-219, 2009.

[128] S. Yu et al., “Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory,” *Applied Physics Letters*, vol. 98, no. 10, p. 103514, 2011.

[129] L. Zhang, D. Strukov, H. Saadeldeen, D. Fan, M. Zhang and D. Franklin, “SpongeDirectory: Flexible sparse directories utilizing multi-level memristors,” *International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pp. 61-73, 2014.

[130] M.-C. Wu et al., “A study on low-power, nanosecond operation and multilevel bipolar resistance switching in ti/zro$_2$/pt nonvolatile memory with 1T1R architecture,” *Semiconductor Science and Technology*, vol. 27, p. 065010, 2012.

[131] A. Yekkehkhany, A. Hojjati, and M. H. Hajiesmaili “GB-PANDAS:: Throughput and heavy-traffic optimality analysis for affinity scheduling,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 2-14, 2018.

[132] A. Daghighi and M. Kavousi, “Scheduling for data centers with multi-level data locality,” *2017 Iranian Conference on Electrical Engineering (ICEE)*, pp. 927-936, 2017.

VITA

Graduate School
Southern Illinois University

Seyed Nima Mozaffari Mojaveri

nima.mozaffari@siu.edu

University of Mazandaran, Mazandaran, Iran
Bachelor of Science in Electrical and Computer Engineering, September 2007

University of Tehran, Tehran, Iran
Master of Science in Electrical and Computer Engineering, September 2010

Special Honors and Awards:
Dissertation Research Award (DRA), 2017

Dissertation Title:
DESIGN AND TEST OF DIGITAL CIRCUITS AND SYSTEMS USING CMOS AND EMERGING RESISTIVE DEVICES

Major Professor: Dr. Spyros Tragoudas

Publications:

1. S. N. Mozaffari and S. Tragoudas, “Maximizing the Number of Threshold Logic Functions Using Resistive Memory,” *IEEE Transactions on Nanotechnology*, vol. PP, no. 99, pp. 1-1, 2018.

2. S. N. Mozaffari, S. Tragoudas and T. Haniotakis, “A Generalized Approach to Implement Efficient CMOS-Based Threshold Logic Functions,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 3, pp. 946-959, 2018.

3. S. N. Mozaffari, S. Tragoudas and T. Haniotakis, “More Efficient Testing of Metal-Oxide Memristor–Based Memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 6, pp. 1018-1029, 2017.

4. S. N. Mozaffari, S. Tragoudas and T. Haniotakis, “Reducing power, area, and delay of threshold logic gates considering non-integer weights,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-4, 2017.

5. S. N. Mozaffari, S. Tragoudas and T. Haniotakis, “Fast march tests for defects in resistive memory,” *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH15)*, pp. 88-93, 2015.

6. S. N. Mozaffari, S. Tragoudas and T. Haniotakis, “A new method to identify threshold logic functions,” *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 934-937, 2017.

7. S. N. Mozaffari, and Spyros Tragoudas, “Memristive Current Mode Threshold Logic Gates,” *International Workshop on Memristive Materials, Devices & Systems (MEMRISYS)*, 2017.