Bio-inspired Hardware Architectures for Memory, Image Processing, and Control Applications

by

Yalcin Yilmaz

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2017

Doctoral Committee:

Professor Pinaki Mazumder, Chair
Assistant Professor Kira Barton
Professor Yogesh Gianchandani
Professor Semyon M. Meerkov

Yalcin Yilmaz

yalciny@umich.edu

ORCID iD: 0000-0003-1524-4633

For my parents Birsen and Zafer Yilmaz

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table

ABSTRACT

Bio-inspired Hardware Architectures for Memory, Image Processing, and Control Applications

by

Yalcin Yilmaz

Chair: Pinaki Mazumder

Emerging technologies are expected to partially replace and enhance CMOS systems as the end of transistor scaling approaches. A particular type of emerging technology of interest is the variable resistance devices due to their scalability, non-volatile nature, and CMOS process compatibility. The goal of this dissertation is to present circuit and system level applications of CMOS and variable resistance devices with bio-inspired computation paradigms as the main focus. The summary of the results offered per chapter is as follows:

In the first chapter of this thesis, an introduction to the work presented in the rest of this thesis and the model for the variable resistance device is provided.

In the second chapter of this thesis, a crossbar memory architecture that utilizes a reduced constraint read-monitored-write scheme is presented. Variable resistance based crossbar memories are prime candidates to succeed the Flash as the mainstream nonvolatile memory due to their density, scalability, and write endurance. The proposed scheme supports multi-bit storage per cell and utilizes reduced hardware, aiming to decrease the feedback complexity and latency while still operating

with CMOS compatible voltages. Additionally, a read technique that can successfully distinguish resistive states under the existence of resistance drift due to read/write disturbances in the array is presented. Derivations of analytical relations are provided to set forth a design methodology in selecting peripheral device parameters.

In the third chapter of this thesis, an analog programmable resistive grid-based architecture mimicking the cellular connections of a biological retina in the most basic level, capable of performing various real time image processing tasks such as edge and line detections, is presented. Real time vision systems require computationally intensive tasks which often benefit greatly from fast and accurate feature extractions. Resistive grid-based analog structures have been shown to perform these tasks with high accuracy and added advantages of compact area, noise immunity, and lower power consumption compared to their digital counterparts. However, these are static structures that can only perform one type of image processing task. The proposed unit cell structure employs 3-D confined resonant tunneling diodes called quantum dots for signal amplification and latching, and these dots are interconnected between neighboring cells through non-volatile continuously variable resistive elements. A method to program connections is introduced and verified through circuit simulations. Various diffusion characteristics, edge detection, and line detection tasks have been demonstrated through simulations using a 2-D array of the proposed cell structure, and analytical models have been provided.

In the fourth chapter of this thesis, a bio-inspired hardware designed to solve the optimal control problem for general systems is presented. Adaptive Dynamic Programming algorithms provide means to approximate optimal control actions for linear and non-linear systems. Action-Critic Networks based approach is an efficient way to approximately evaluate the cost function and the optimal control actions. However, due to its computation intensiveness, this approach is usually implemented in high level programming languages run using general purpose processors. The

presented hardware design is aimed at approximating the solution to the Bellman equation to find the optimal control action and to reduce the computation time and the hardware overhead by using the Heuristic Dynamic Programming algorithm which is a form of Adaptive Dynamic Programming. The proposed hardware operating at mere speed of 10 MHz yields 237 times faster learning rate in comparison to conventional software implementations running on fast processors such as the 1.2 GHz Intel Xeon processor. The proposed system-on-chip (SoC) integrated circuit is designed using 65 nm CMOS LP technology, and has a dimension of 1950 μm $\times$ 1550 μm while consuming 2.1 mW at 10 MHz operation frequency and 1.2 V supply voltage.

CHAPTER I

Introduction

CMOS scaling has been consistently providing increased density for the modern VLSI chips as predicted by the Moore's law, which dictates that the number of transistors in a semiconductor chip doubles approximately every two years. CMOS technology has facilitated Van Neumann architecture based computation paradigms to flourish and dominate the digital world for decades. However, as the transistor scaling is reaching its physical limits, and with the emergence of new technologies that provide interesting physical properties, alternative computation paradigms might need to be adopted in wide range of applications.

The nervous systems of living organisms perform many complex tasks with much more energy and computational efficiency than the current VLSI chips. For example, human brain can perform 10^{17} FLOPS while dissipating around merely 15W. Therefore, bio-inspired neuromorphic computation paradigms have been attracting significant attention. Especially, mimicking neuron and synapse functionalities with CMOS circuitry has been the goal of many researchers in order to investigate if the computational efficiency of the biological systems can be attained on semiconductor chips.

The emerging technologies provide inherent advantages over the CMOS technology in some applications due to the fact that their operation principles are based

on different physical properties. For example, such devices facilitate the mapping of certain computations directly to their physical properties which mean that a single device can realize a functionality that would otherwise require tens of CMOS transistors.

Among the emerging technologies, programmable variable resistance devices have recently attracted significant attention in various applications (bio-inspired and otherwise) after Hewlett-Packard research labs revealed that "memristance" can be observed in nano-scale thin film devices [1]. The significance of these devices arises from the fact that they can retain their resistive state even when power is turned off, displaying non-volatility and they might enable scaling beyond CMOS technology limits. The variable resistance characteristics of these devices are proposed to be utilized in ultra-dense crossbar memories [2], configurable logic applications and as synaptic connections in neuromorphic architectures [3]. They have also been used for carrying out image processing tasks which benefit from their non-linearity and adaptive characteristics [4]. Most of these applications could benefit from the use of these devices more if the devices show properties of long term stability of resistive states and little or no degradation of these states when the values stored in these devices are read. Fabrication results reported in [5] indicate the observation of diode-like behavior in amorphous Silicon (a-Si) devices which are undisturbed when the voltages across the devices are below a certain threshold and can retain their states more than 4 years at room temperature.

This thesis focuses on the circuit and system level applications of CMOS and variable resistance devices, bio-inspired computation paradigms being the main focus. The first application that will be presented is a non-volatile multi-level crossbar memory, capable of storing two or more bits per cell, facilitating the design of ultra-dense data storage. This application has the most potential for immediate adoption for commercialization as it provides increased memory density, reducing storage cost

per bit. The following applications focus on the bio-inspired computation paradigms, one being the implementation of a programmable resistive grid acting as an artificial retina to realize various computer vision tasks such as edge and line detection. In fact, the structure can be programmed to implement different diffusion characteristics which can be extended to other image processing tasks. The final application that will be presented is the most significant contribution of this thesis, which provides a bio-inspired hardware to solve the optimal control problem for general systems. Living organisms interact with their environments and learn from their experiences to optimize their actions to get the best outcome. For example, they try to find the best food source, find the best path to avoid predators, etc. Inspired by this type of 'Reinforcement Learning', a neuromorphic hardware is proposed, which efficiently performs the objective minimization/maximization via on-chip learning mechanisms that is otherwise implemented in high level programming languages and run on general purpose processors with much less computation and energy efficiency.

Next subsection will provide information on the physical properties and modeling of the variable resistance devices (memristors) which comprise the main components presented in the next two chapters of this work.

1.1 Variable Resistance Device Model

"Memristor" is the fourth fundamental circuit element which relates charge with magnetic flux as described by Leon Chua in [6]. HP research labs revealed in [1] that two terminal thin-film based devices can exhibit variable resistance behavior.

As laid out in [1], these devices can be modeled as a combination of two series variable resistors, with one of the resistors having a high dopant concentration, thus having low resistance and the other having a low dopant concentration, thus having high resistance. This model is visualized in Figure 1.1. Application of a voltage across the terminals of the device triggers dopant drift. Depending on the voltage polarity,

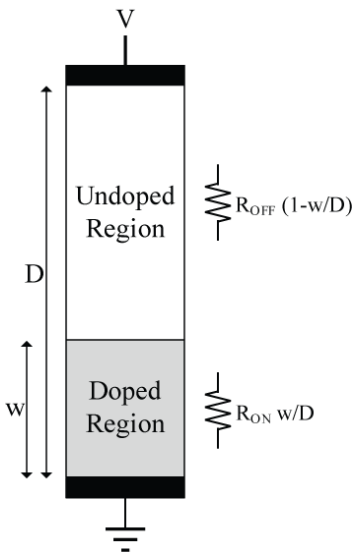

Figure 1.1: Variable Resistance Device Structure

the width of the doped region can increase or decrease. As the width of the doped region increases, conductance increases and as it decreases, conductance decreases.

Therefore the total resistance of the device can be expressed as:

$$R = \frac{w}{D}R_{ON} + (1 - \frac{w}{D})R_OFF \tag{1.1}$$

where w is the width of the doped region, D is the total length of the thin film, R_{ON} is the lowest resistance when $w = D$ and R_{OFF} is the highest resistance when $w = 0$. When the current is passed through the device, the width of the doped region, w changes. The rate of change of w with time is:

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{ON}}{D} i(t) \tag{1.2}$$

where μ_v is the dopant mobility and $i(t)$ is the current passing through the device. The above model presented by HP labs is a linear model and does not account for the nonlinearities that are present in most fabricated devices due to second order boundary effects seen at thin film edges. The movement of the boundary between

doped and undoped regions is greatly hindered when the width of the doped region approaches device limits (i.e., $w = 0$ or $w = D$) [7]. After including the boundary effects, the model expression becomes:

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{ON}}{D} i(t) f(x) \tag{1.3}$$

where $f(x)$ is the window function modeling the nonlinear dopant drift. This function is an estimation of nonlinearity and depends on the specific device behavior. A sample function is provided in [7].

The actual switching characteristics, namely switching delay, of the devices depend on material properties, device dimensions and biasing voltage.

There have been significant efforts in developing memristor SPICE models [8, 9, 10] and even SPICE-like simulators [11] to facilitate the design of hybrid memristor and CMOS circuits through conventional simulation tools that are familiar to circuit designers.

A SPICE model [8] based on this device model is adapted and implemented in Verilog-A to carry out circuit simulations in the following chapters.

CHAPTER II

Multi-Level Memory Architecture

2.1 Introduction

Nonvolatile memory technologies led by NAND Flash have been generating increased market revenues due to the increased usage of these devices, especially in portable consumer electronics and solid state drives (SSDs) [12]. The trend toward the cloud storage and computing is continually demanding enterprises to invest especially in SSD-based storages, as these provide higher performance compared to hard disk drives (HDDs) [13].

Flash memories have been providing solutions to the ever-increasing high- performance storage demands with continued feature scaling. However, flash scaling is reaching its limits due to the increased reliability problems such as aging of the oxide, charge leakage, retention problems, and the increased capacitive coupling between the floating gates of the neighboring cells [14].

The approaching end of the flash scaling has led researchers to look for alternative nonvolatile memory technologies that can sustain the scaling trend [15]. Many promising emerging technologies have been proposed, each with its own advantages and challenges. Magnetoresistive random-access memory (MRAM) [16], spin-transfer torque random-access memory (STT-RAM) [17], phase-change memory (PCRAM) [18] and resistive random-access memory (RRAM) [19], which is also commonly re-

ferred to as memristive crossbar memory, have been the major candidates to supersede the flash technology.

The successor technology has to be dense, has to be scalable, has to have high write endurance, and has to support multi-level cell structure as this has been the trend in flash. Variable resistance devices (memristors) as predicted by Chua in his 1971 paper [6] and realized by Hewlett-Packard Labs in [1] meet all these requirements with their CMOS compatibility, write endurance, data retention, multilevel storage capability, and scalability down to molecular dimensions [20].

Ever since the discovery of the missing variable resistance devices [1], they have attracted great interest not only due to their nonvolatile nature but also due to their hysteretic variable resistance characteristics which allowed for the realization of unconventional circuits and systems. They have found their applications in logic circuits [21, 22], neural computing [23, 24, 25], image processing [26], analog circuits [27, 28], field-programmable gate arrays (FPGAs) [29, 30] and nonvolatile memory [31, 32, 33]. However, among these, the most commercially promising application is the nonvolatile crossbar memory due to existing consumer market.

The crossbar memory has attracted more attention, due to its increased cell density compared to the other architectures that have been proposed, such as the unfolded architecture presented in [34], which requires more metal connections to be routed. In order to further increase the storage density, there has been much research effort in terms of achieving multi-bit storage per cell [33, 35] rather than single-bit. Multi-bit provides increased storage per unit area, reducing fabrication costs.

To achieve single or multi-bit storage per cell, various write schemes have been proposed. These schemes are split into two main categories: Pulse based schemes, where a predetermined duration and amplitude pulse is applied to the cell vs feedback based schemes, where the pulse duration depends on the feedback circuit, indicating if the cell has reached the desired state [36].

Feedback schemes are shown to have advantages over the pulse-based schemes, as they limit the resistive distributions of the programmed cells. In [37], it has been shown that a feedback-based scheme shows narrowing of the resistance distributions compared to a pulse-based scheme. However, the use of DAC, ADC [32, 33], or multi-stage comparisons [36] in feedback circuitry can introduce significant peripheral circuitry overhead, and can introduce latency in response time that can be significant when the memory device is highly non-linear. Thus, a more simplistic approach is required to reduce the circuit overhead, and to reduce latency to avoid over-programming.

Aside from the read/write techniques, another important factor that plays a role in the design of the resistive crossbar memory is the cell structure that is used in the memory array. There are three major types of cell structures that have been proposed: 1T1R, where a selection transistor is integrated in series with a resistive device [38], 1D1R structure, where a series diode is integrated with a resistive device [39] or the device itself shows diode-like behavior [40] and 1R structure [41], where a resistive device does not have any series selection device or diode-like behavior. Other device types are also proposed such as a 3-terminal resistive devices as in [42].

1T1R structure has density problem. With this structure, the memory density is dictated by the scaling of the series transistor, which has bigger feature size than the memristor cell itself. 1R structure has so called "sneak paths" problem which limit the array size due to the deterioration of sensing margins. In fact, relatively smaller-size arrays have been shown to provide enough margin for sensing [43]. In these structures, so called "half-selected cells" still see VDD/2 voltage levels across them [38] and their resistances can drift over time due to the read/write disturbances [44]. Although some methods claim the disturbance is not significant [36], it is more pronounced in 1R architectures as there is no selection device to reduce or eliminate the leakage through these cells. Some proposed methods such as grounding of unselected rows

and columns can even cause the half-selected cells to see a higher voltage across them than the selected cell as shown in [45], significantly disturbing the half-selected cells.

Thus, it is believed that 1D1R structure is the most promising solution as it does not suffer from density issues like 1T1R and it does not suffer from the sneak paths problem as much as the 1R structure. Indeed, Crossbar Inc. recently presented a RRAM structure that utilizes series selector devices that provide programming thresholds [46], strengthening the belief that 1D1R structures will be widely adopted in RRAM designs. In order to realize these structures, there have been various approaches. The introduction of series diodes [39] or metal-insulator-metal (MIM) diodes [47, 48] and the engineering of the devices to integrate diode-like behavior [40, 49] in the device itself is presented in literature. Even with the series diode, cell-to-cell isolation is not perfect. Most works in literature fail to consider what happens to unselected or half- selected cells as the other cells are being programmed. In this work, how the programmed resistance distributions change as all the cells in the array are programmed is observed, and a read method that compensates for the expanded and shifted distributions is presented.

A read/write scheme where the voltage references are derived from the intermediate node via means of a combination of active and passive devices, such as diodes, diode-connected transistors, and resistors that generate distinct thresholds is put forward. The contributions presented in this chapter include a reduced-constraint read-monitored-write scheme which supports multi-bit storage per cell and utilizes reduced hardware, aiming to reduce the feedback complexity and latency while still operating with CMOS-compatible voltages, a read technique that provides enough margins for state detection while allowing certain amount of resistance drift in the array cells (thus reducing the need for frequent refresh operations), a relaxed array biasing scheme that aims to facilitate read/write operations while reducing cell disturbances, and derivations of analytical relations to pave the path for a design methodol-

ogy in selecting peripheral device parameters. The outlined read/write methodology applies generally to 1D1R structures, but can be generalized to other structures with minor modifications.

In Section II, the approach adopted for modeling of the memory cells is presented. Section III details the memory architecture, our read/write methodology, and analytical expressions that guide the peripheral circuitry design. Section IV and V present our simulation results for read/write operations as well as the effects of variations on the programming voltages and the series resistance.

2.2 Multi-State Memory Architecture

2.2.1 Architecture

Proposed multi-level memory architecture is presented in Figure 2.1. The crossbar memory array is the main storage area that is composed of metal crossbars and resistive cells located at every intersection of these crossbars.

N-type and p-type access transistors enable the driving of the crossbars with adequate voltage levels. Row and column decoders activate the relevant access transistors depending on the location of the selected cell in the array. The voltage drivers provide various voltage levels to adequately bias the selected or unselected cells. The read interpreter circuitry is serially connected to the selected column through the access transistors, and is capable of actively monitoring the resistance of the selected cell. The interpretation results encoded in voltage levels are then fed into the comparators to detect whether the desired resistive state is reached. The memory controller is responsible for coordinating which memory operation to perform, and generates relevant control signals to activate peripheral blocks.

In this work we present the multiplexed read and write circuitries as parallel reads can introduce circuit overhead [50], however, our methodology can be modified

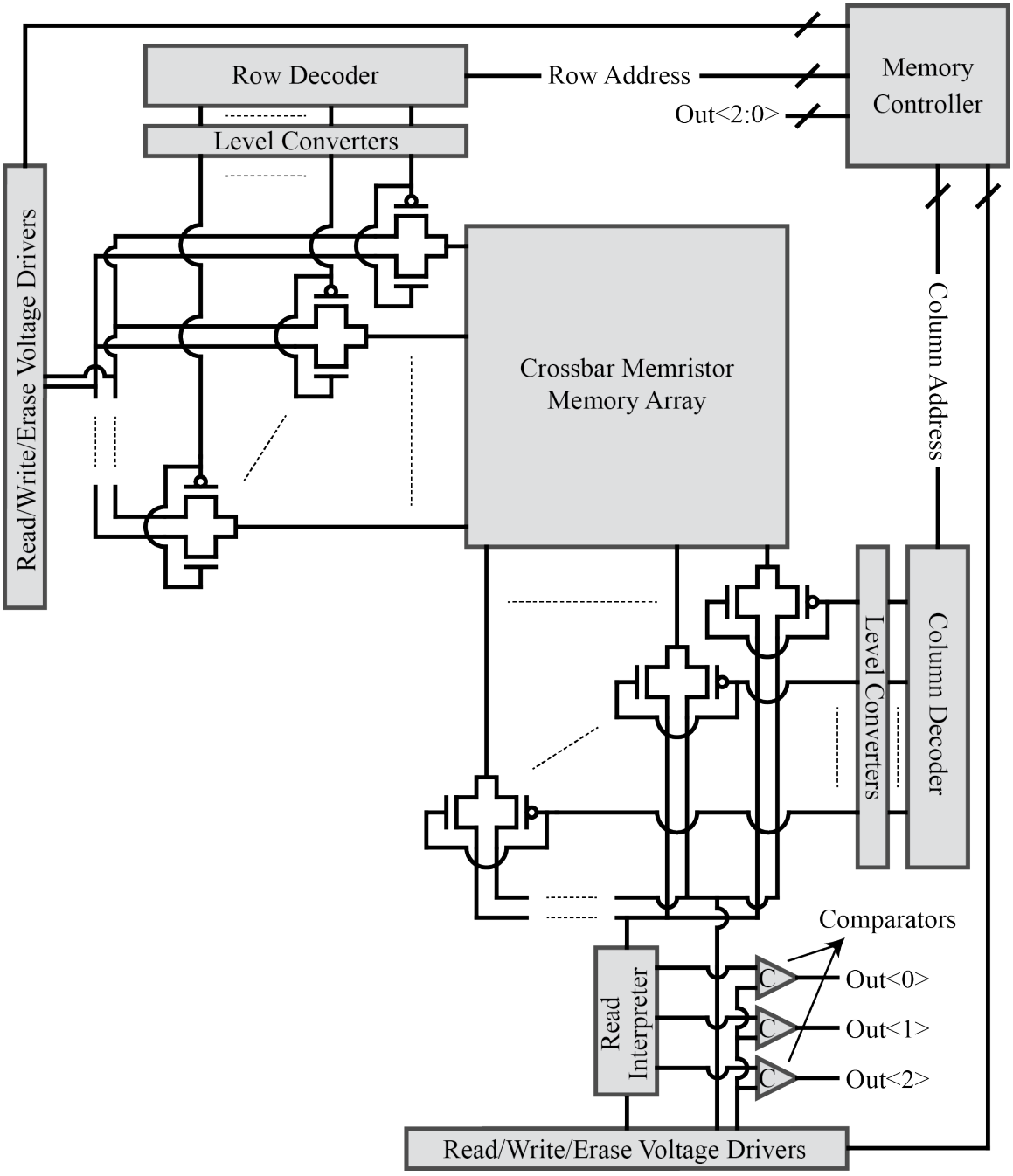

Figure 2.1: Multi-level RRAM architecture

to support parallel reading and writing of the cells on the selected row.

2.2.2 Read/Write Circuitry

Figure 2.2 shows the read interpreter circuitry which actively monitors the resistance change in real time during a write operation, and detects the encoded state during a read operation. The read circuitry is composed of a voltage division stage employing diodes and resistors to interpret the voltage change across the series resistor, and a comparison state employing fast comparators to detect if a desired state is reached. R_{series} is the series resistor, V_{int} is the voltage level on the intermediate node that is the node connecting the read interpreter and the selected column through the selection circuitry, Interp⟨2:0⟩signals are the outputs generated by the voltage division stage, and Out⟨2:0⟩are the corresponding comparator outputs generated in the comparison stage.

The series diode in the voltage division stage provides a close-to-constant voltage reduction in the voltage to be interpreted by the circuitry, providing compaction of the resistive states. The interpreter circuitry can be expanded depending on the number of bits to be stored in the memory cell. For n-bit storage, $2^n - 1$ diodes and comparators are needed. The resistors connected to the diode outputs have high resistances (1M Ω) in order to minimize the effect of the interpreter circuitry on the intermediate node voltage.

The Interp⟨2:0⟩signals are unique analog outputs, and their values decrease as the resistance of the cell increases. The read interpreter circuity ensures that the Interp⟨2⟩signal falls below the comparator threshold before the Interp⟨1⟩signal does, and Interp⟨1⟩signal falls below the threshold before the Interp⟨0⟩signal. Each Interp signal falling below the comparator threshold indicates that a particular resistive state is reached. We have adopted the convention such that the resistive states are '00', '01', '10' and '11'; where the states are listed in the order of increasing resistance.

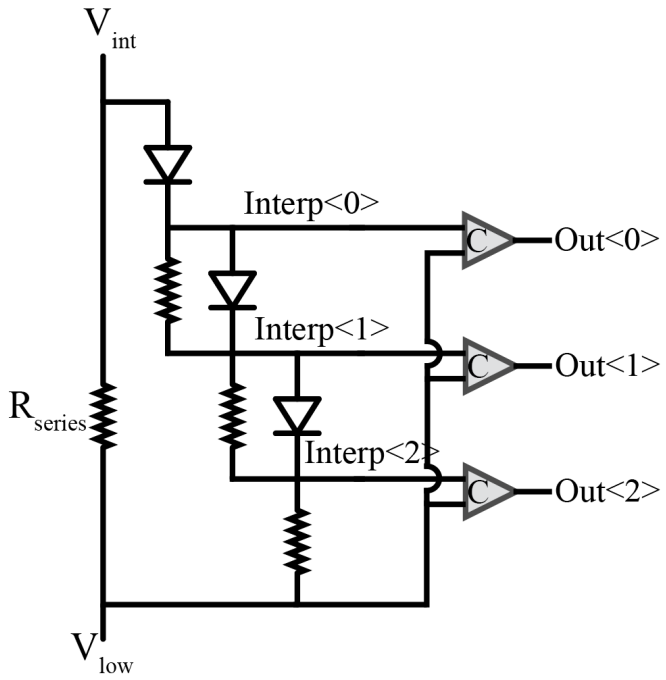

Figure 2.2: Read interpreter circuitry block

However, the adoption of the reverse convention where the states are in the order of decreasing resistance is also possible.

2.2.3 Array Voltage Bias Scheme

When performing a write operation, the voltage bias across the selected memory cell should exceed the cell threshold to achieve a fast write operation, whereas in order to minimize the resistance change of the unselected or half-selected cells, the voltage bias across these cells should be kept lower than the threshold of the memory cells.

To achieve this goal, we bias the array with four different voltage levels as shown in Figure 2.3. The selected row is applied $V_{select-row}$, the unselected rows are applied $V_{unselect-row}$, the unselected columns are applied $V_{unselect-col}$, and the selected column is applied $V_{select-col}$ through the interpreter circuitry which yields an applied voltage value of V_{int} on the intermediate node. These conditions are summarized as follows:

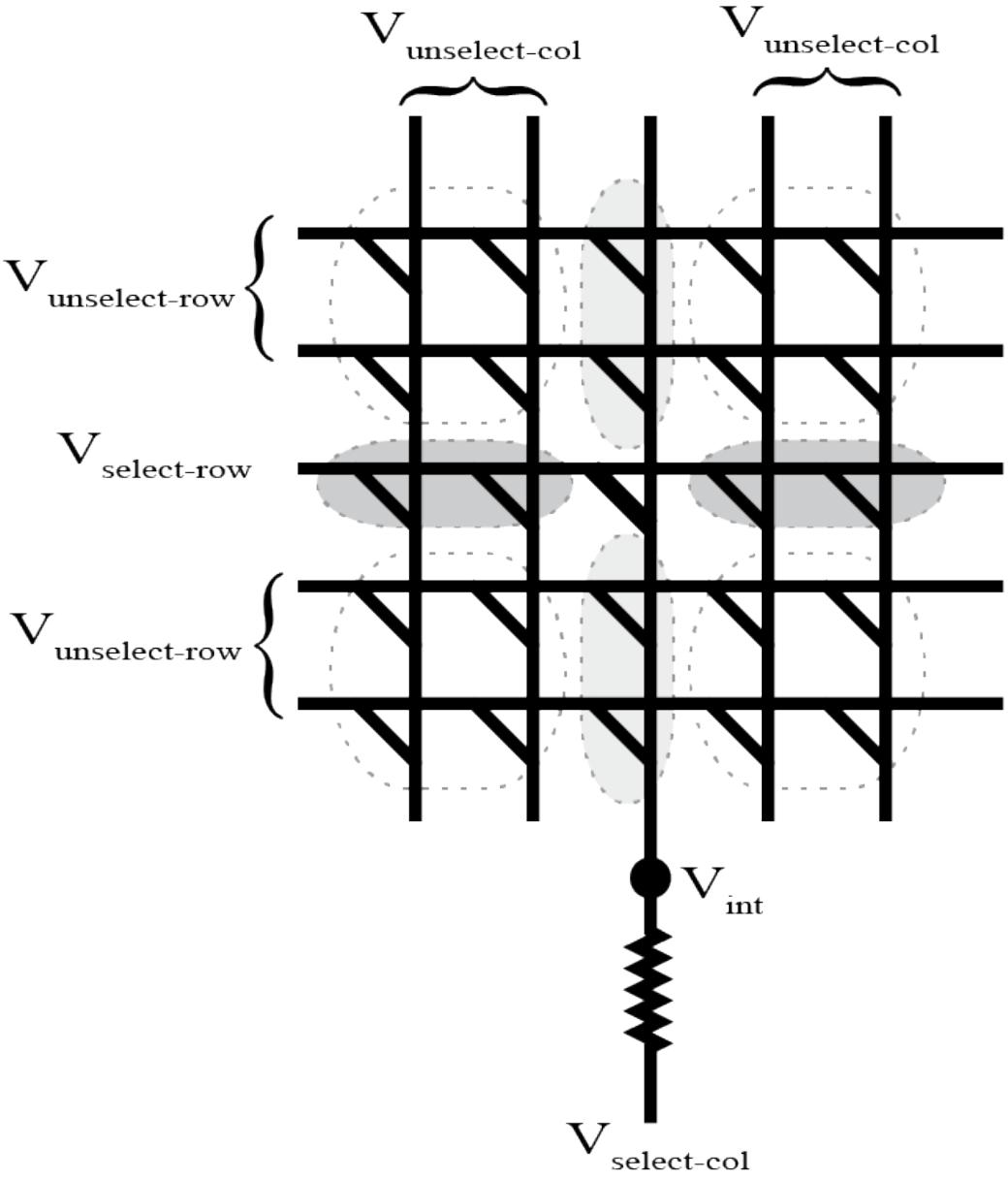

Figure 2.3: Memory array biasing scheme. The cells that lie in white regions see $V_{unselect-row} - V_{unselect-col}$, the cells that lie in light grey regions see $V_{unselect-row} - V_{int}$, the cells that lie in dark grey regions see $V_{select-row} - V_{unselect-col}$ across them. The selected cell represented with a thick line sees $V_{select-row} - V_{int}$ across it.

$$|V_{select-row} - V_{int}| > V_{mth} \tag{2.1a}$$

$$|V_{select-row} - V_{unselect-col}| < V_{mth} \tag{2.1b}$$

$$|V_{unselect-row} - V_{int}| < V_{mth} \tag{2.1c}$$

$$|V_{unselect-row} - V_{unselect-col}| < V_{mth} \tag{2.1d}$$

where V_{mth} is the memory cell threshold of the 1D1R cell. The above stated conditions can be satisfied by choosing voltage values that follow the following inequality:

$$V_{select-row} > V_{unselect-col} > V_{unselect-row} > V_{int} > V_{select-col} \tag{2.2}$$

Uneven biasing of the unselected rows and columns are also proposed in [51], where the array is biased with voltage levels that are VDD/3 apart. However, the scheme we propose does not have strict rules on the voltage levels, as long as the difference between the two consecutive voltage values is selected so that it is smaller than the magnitude of the memory cell threshold.

This biasing scheme yields four groups of cells that observe different voltage levels at their terminals as shown in Figure 2.3. Since V_{int} value changes during programming, the voltage difference at the terminals of the unselected cells connected to the same column as the selected cell is not constant. Therefore, it is important to pick the voltage levels such that the worst case voltage difference across these cells is below the cell threshold.

2.2.4 Read/Write Operations Flow

At the beginning of the write operation, it is assumed that the selected cell in the array is at the erase state, which corresponds to the lowest resistive state ('00')

in our convention. After the controller is prompted to perform a write, it signals the voltage drivers to apply relevant voltage levels on the array, and enables the row and column decoders to facilitate the application of the voltages on the selected and unselected rows and columns. As the voltages are applied, the interpreter circuitry generates distinct analog voltage levels (Interp) that directly depend on the resistance of the selected memory cell. As the cell resistance increases, the Interp signal levels begin to decrease. As soon as one of these signals reaches the comparator threshold, the corresponding comparator output (Out) signal flips. The controller checks if the comparator outputs indicate that the desired state is reached. If the state is reached, the controller immediately terminates application of voltages; if not, the controller keeps enabling the application of the write voltages on the array. The flow chart visualizing these steps is shown in Figure 2.4a. The chart also includes the possible write protection and failed cell detection mechanisms that can be adopted similar to flash memories.

Figure 2.4b, shows the flow chart for the read operation. The read operation is similar to a write operation, except the voltage levels and the interpreter circuitry used can have different characteristics, as will be discussed in the following sections.

When the controller is prompted to perform a read, it signals the voltage drivers to apply read voltage levels on the array, and enables the row and column decoders to facilitate the application of the these voltages on the selected and unselected rows and columns. As the voltages are applied, the interpreter circuitry generates three distinct Interp signals that are dependent on the resistance of the memory cell. Interp signals are fed into the comparators. Out signals generated by the comparators indicate which resistive state the memory is in, and the read operation completes. Unlike the write operation, the read operation has a fixed duration, and this duration should be kept as short as possible to reduce the read disturbance which can cause the memory resistance to drift.

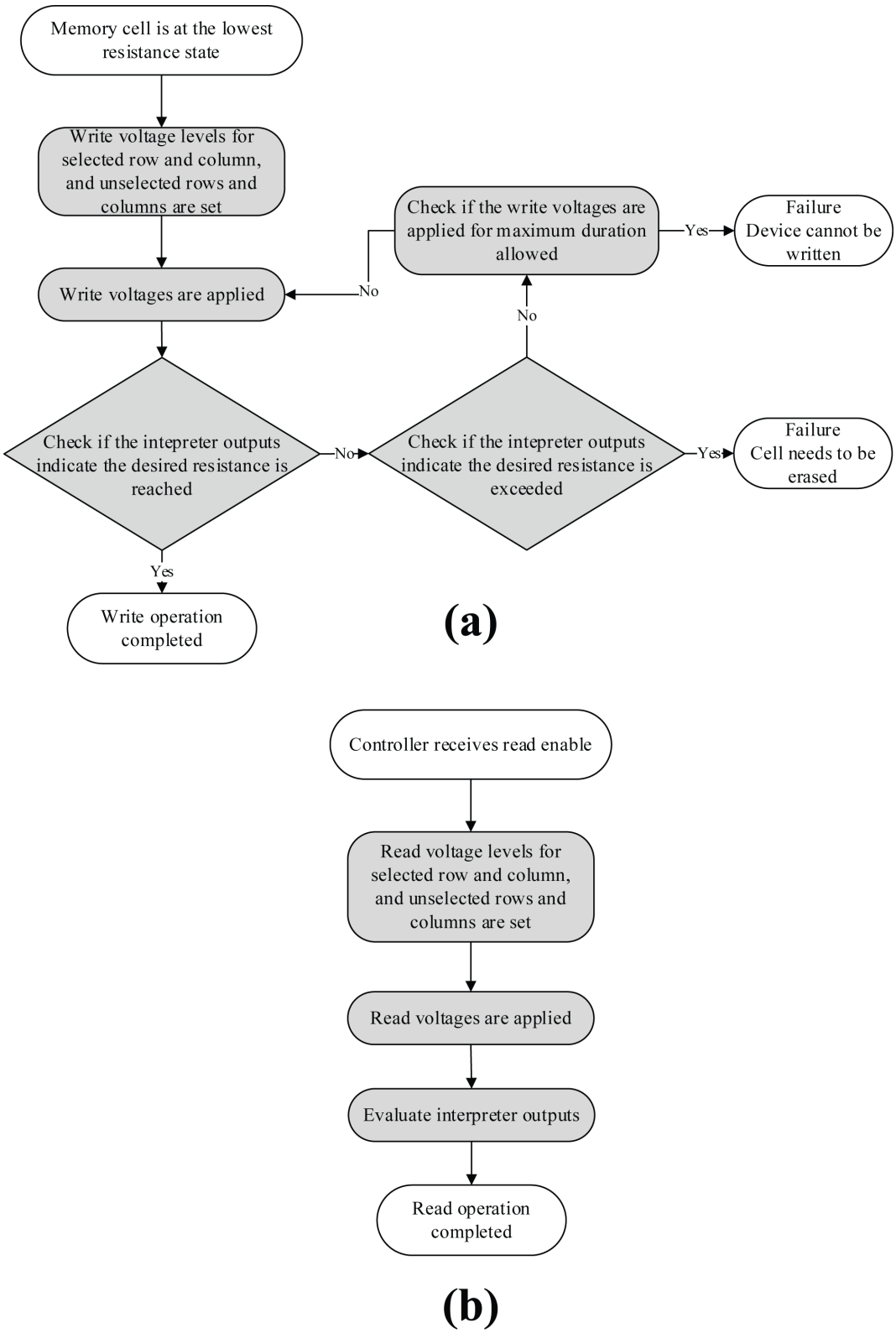

Figure 2.4: Flow chart explaining memory operations: a) Write Operation, b) Read Operation

2.2.5 State Derivations

It is important to characterize the behavior of the read interpreter circuitry together with the array elements in order to understand how the component parameters should be selected.

First, we characterize the dependence of the resistive state of the memory cell on the intermediate node voltage. Then, we characterize the dependence of the intermediate node voltage on the series diode threshold based on the detection threshold requirements of the comparison stage and the current-voltage (IV) characteristics of the series diode in the interpreter circuitry.

The analog voltage levels in the intermediate node that correspond to the encoded resistive states can be characterized by the following equation:

$$V_{int} = V_{select-col} + \frac{R_{series}(V_{select-row} - V_{select-col} - V_{th_m})}{R_{parasitics} + R_{cell(s)} + R_{series}} \tag{2.3}$$

where $R_{cell(s)}$ is the resistance of the selected cell at a given state and V_{th_m} is the cell threshold. Parasitic resistance sources represented by lumped $R_{parasitics}$ term consist of the effective resistances of the n-type and p-type access transistors and the crossbar resistance seen by the interpreter circuitry. The read interpreter voltage division stage outputs meet the following equality when a particular memory state is being programmed:

$$V_{Interp_{3-k}} = V_{select-col} + \frac{(V_{int} - V_{diode_1} - V_{select-col})}{3}k = V_{c-res} + V_{select-col} \tag{2.4}$$

where $V_{Interp_{3-k}}$ represents the voltage level at the corresponding voltage division stage output (Interp), and k is the index of the corresponding voltage division stage output, V_{diode_1} is the threshold of the series diode in the interpreter circuitry and V_{c-res} is the comparator threshold. For analysis simplicity, the resistors connected to

the outputs of diodes are assumed to have same resistances. However, it is possible to alter these resistances separately in order to tune the separation between resistive states of the memory cell. The decision on where to set the resistive states is highly dependent on the non-linear behavior of the device. The design decisions can also be made based on the trade-off between the programming time and the resistive margins.

Solving for V_{diode_1} in (2.4) yields:

$$V_{diode_1} = V_{int} - V_{select-col} - \frac{3}{k} V_{c-res} \tag{2.5}$$

The above expression relates the intermediate node voltage with the series diode threshold. We need one more expression that relates the two in order to be able to numerically solve both to obtain V_{int} value, which we can plug in (2.3) to obtain corresponding resistive state level. The additional expression can be obtained using the diode current equation and solving for the diode threshold. The final expression is:

$$V_{diode_1} = 3I_0R_h + V_{int} - V_{select-col} - \frac{kT}{nq} LambertW[\frac{3e^{\frac{3I_0R_h + V_{int} - V_{select-col}}{kT/nq}} I_0R_h}{kT/nq}] \tag{2.6}$$

where LambertW is the lambert omega function, R_h is the magnitude of the resistances connected to the diode outputs, I_0 is the reverse bias saturation current, n is the ideality factor, k is the Boltzmann constant, T is the absolute temperature and q is the magnitude of the charge of an electron.

Since I_0R_h term is extremely small, it can be omitted where it is an additive factor. The new simplified expression yields:

$$V_{diode_1} = V_{int} - V_{select-col} - \frac{kT}{nq} LambertW[\frac{3e^{\frac{V_{int} - V_{select-col}}{kT/nq}} I_0R_h}{kT/nq}] \tag{2.7}$$

The expressions (2.5) and (2.7) form a pair of equations with two unknown pa-

rameters. They can be evaluated together numerically to obtain a unique pair of V_{int} and V_{diode_1} values which satisfy both. A unique pair is obtained for each resistive state since expression (2.5) depends on the state to be encoded. After obtaining unique pairs, the next step is to evaluate expression (2.3) to obtain corresponding resistive states.

	Calculated(Ω)	Simulated(Ω)	% Error
'01'	10064	10166	1.003
'10'	20227	20590	1.763
'11'	25653	26067	1.588

Table 2.1: Calculated vs. Simulated Resistance Levels

Table 2.1 lists the resistances calculated using expressions (2.3), (2.5) and (2.7) vs the results obtained through SPECTRE simulations of a 16x16 array. The percent error in calculations are also listed. The disagreement between the simulated and the calculated results are less than 2% for each programmed state. The leakage through the half-selected cells also contributes to the intermediate node voltage V_{int}; however, the agreement between the simulated and calculated results indicate that this contribution to voltage mode reading is minimized especially owing to the 1D1R structure and the array biasing scheme used.

Another point to note is that the analytical models derived in this section are not dependent on the resistive device model used. The proposed circuitry behaves the same as long as the model meets the minimum and maximum resistances required for state encoding.

2.3 Read/Write Operations

2.3.1 Read/Write Simulations

Our simulations are performed on a 16x16 array with the adoption of distributed PI-model for the metal crossbars. The read interpreter circuitry is capable of per-

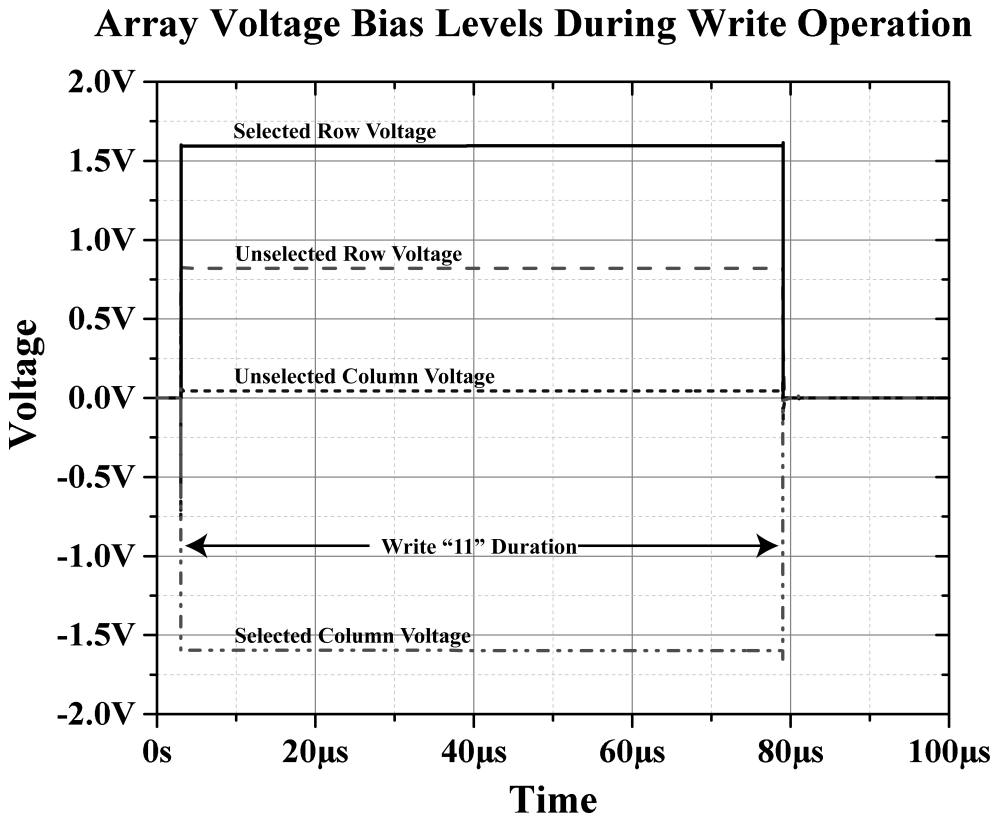

Figure 2.5: Array biasing levels during a write operation

forming a read operation during a write operation by actively monitoring the voltage change across the series resistor connected to the selected column via the selection circuitry.

The bias voltages applied to the array during a write operation is shown in Figure 2.5. As explained earlier, this biasing scheme ensures that the disturbances to the unselected cells are minimized while the selected cell is exposed to a large voltage bias higher than the cell threshold. Example voltage levels shown in Figure 2.5 are: $V_{select-row} = -V_{select-col} = 1.6V$, $V_{unselect-row} = 0.82V$ and $V_{unselect-col} = 0.045V$.

The operation of the interpreter circuitry is shown in Figure 2.6. The voltage division stage generates three distinct Interp signal levels which are then compared to the selection voltage that is also applied through the interpreter circuitry. Each Interp signal is connected to a comparator, and the comparator generates the corresponding Out signal that indicates if a certain state is reached. Interp signal levels

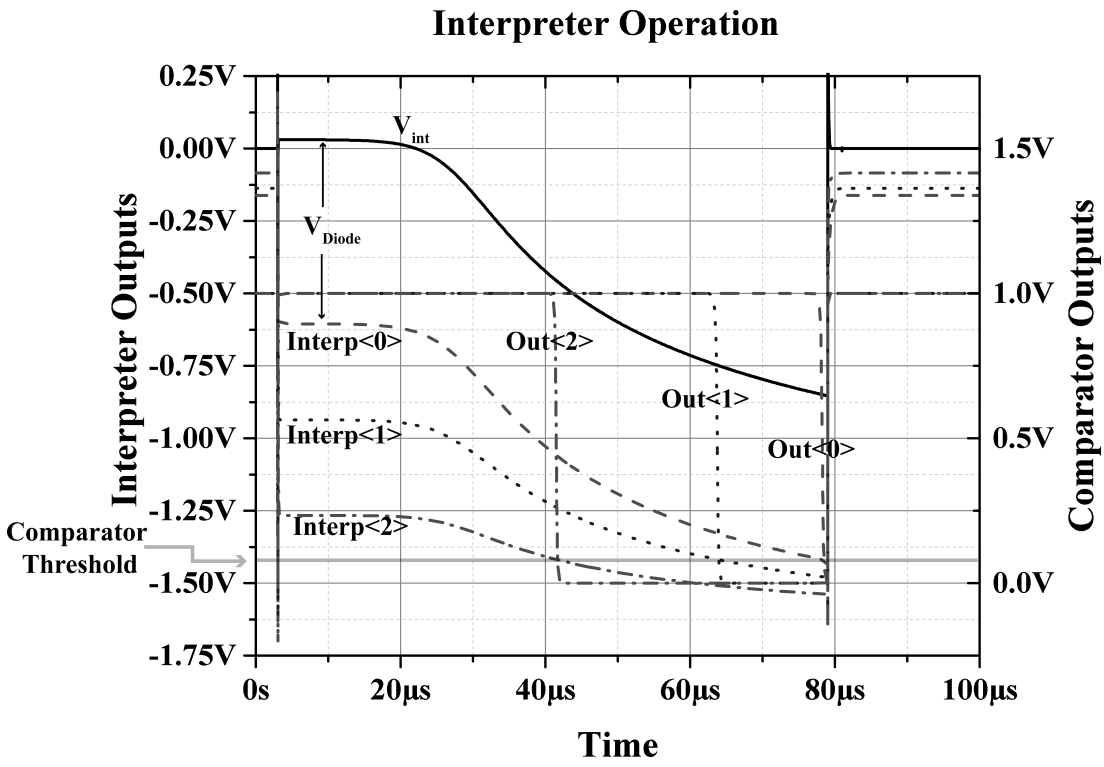

Figure 2.6: Interpreter operation during a write operation. The value being written is '11'

decrease as the resistance of the cell increases. Once the signal level reaches below the comparator threshold, the comparator output (Out) signal becomes low, signaling to the controller.

In our simulations Out⟨2⟩, Out⟨1⟩and Out⟨0⟩indicate that the states '01', '10' and '11' are reached, respectively. The series diode provides a close-to-constant voltage reduction of the intermediate voltage to be interpreted, allowing Interp signals to reach the comparator threshold more quickly; thus, reducing the write time and compacting the resistance levels.

Figure 2.7 shows writing of states '00', '01', '10' and '11'. Since state '00' is the erase state of the cell, the controller does not apply any voltage to the array to change the state, and the Out signals remain high.

In '01', '10' and '11' cases, as the resistance of the cell increases, the Out signals start to become low. Once the desired resistances are reached, the controller stops applying write voltages to the array, and the Out signals become high again.

Since the read interpreter circuitry monitors the state of the cell as it is being written, the read operation can be implemented using the same biasing scheme as the write operation while keeping the pulse duration short to prevent significantly altering the resistive state of the memory. Since the write operation programs each cell to the point of detection, there is a significant read margin when the cell drifts toward a higher resistance; however, there is no margin if the cell drifts toward a lower resistance and a wrong value can be read. In order to mitigate this problem, modifications on the read scheme by reducing the pulse magnitudes used, and/or increasing the series resistance value during a read operation are proposed. The optimizations to the read operation will be discussed further in the following sections.

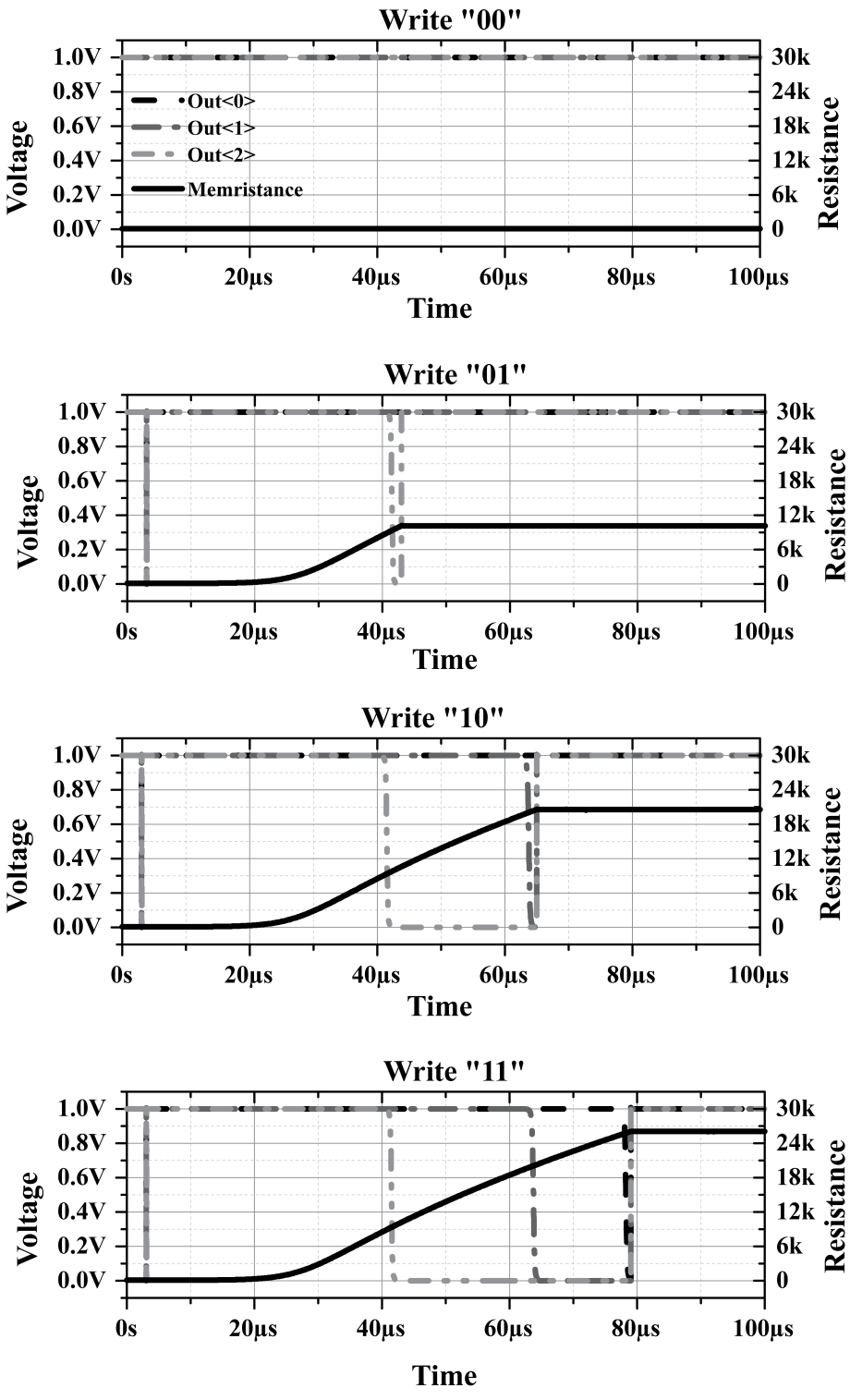

Figure 2.7: Writing of various values into the selected memory cell. Resistance change is overlapped with interpreter output signals.

2.3.2 Read Disturbances to the Neighboring Cells

Even though each memory cell has a built-in threshold, a voltage bias below this value still induces a trivial resistance change in the cell. This means that the cell resistance can drift over time due to the repeated reading of the cell or due to the read and write operations performed on other cells in the array.

In order to quantify the effects of read disturbances in the presented architecture, the resistance drift of the neighboring cells to a selected cell is simulated while the selected cell is read repeatedly. Since the amount of drift is state dependent, the simulations for different values stored in the memory cell and its neighbors are performed. We have assumed the same voltage levels and the series resistance as used during a write operation to simulate the worst case drift. As any reduction in read voltage and increase in series resistance would yield better drift characteristics as shown in Figure 2.8.

The results indicate that for the cases of '01', '10' and '11', the greatest resistance drift is observed in the cells that are connected to the same column as the selected cell. The cells connected to the same row as the selected cell observe a drift that is less-however, still non-zero. The remaining cells observe a close-to-zero drift within the simulated 100 consecutive read operations. Among these three resistive states, the greatest change is observed when the selected cell is storing '11' due to the fact that the voltage bias across the unselected cells connected to the same column as the selected cell is highest.

In the case when the memory cell is storing '00', no meaningful change in resistance was observed, hence the results are not listed.

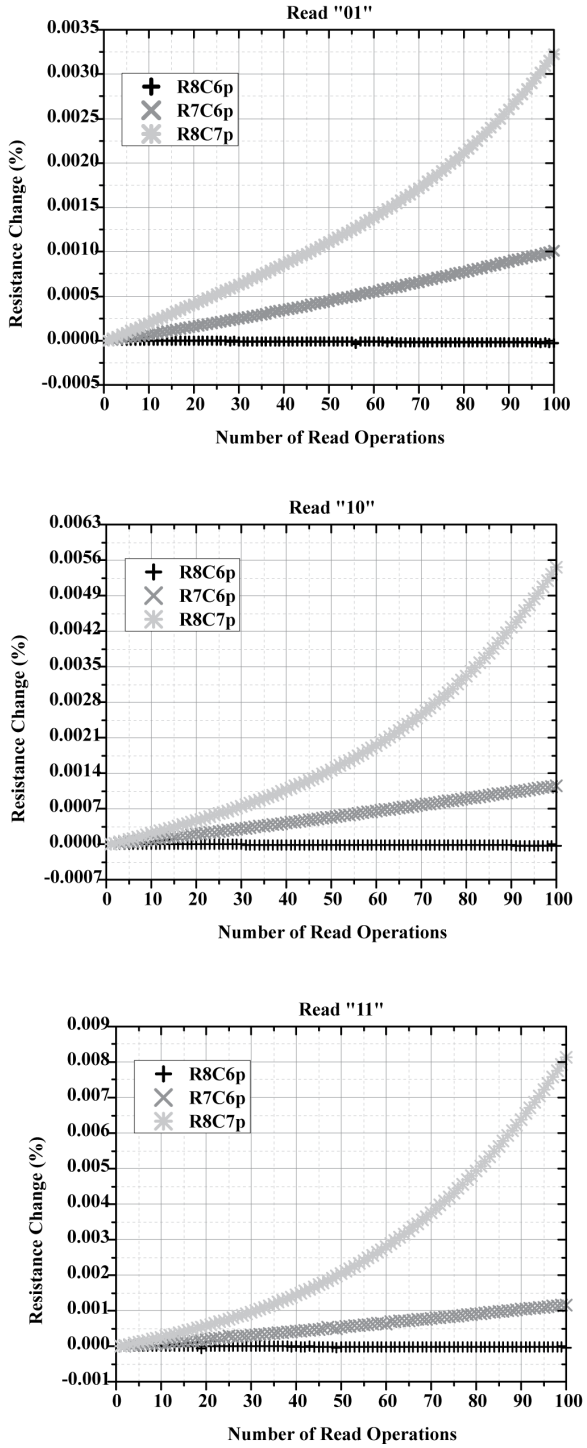

Figure 2.8: Percent change of resistance in neighboring cells vs. number of read operations. The cell located at the crossing of row 7 and column 7 (R7C7) is selected.

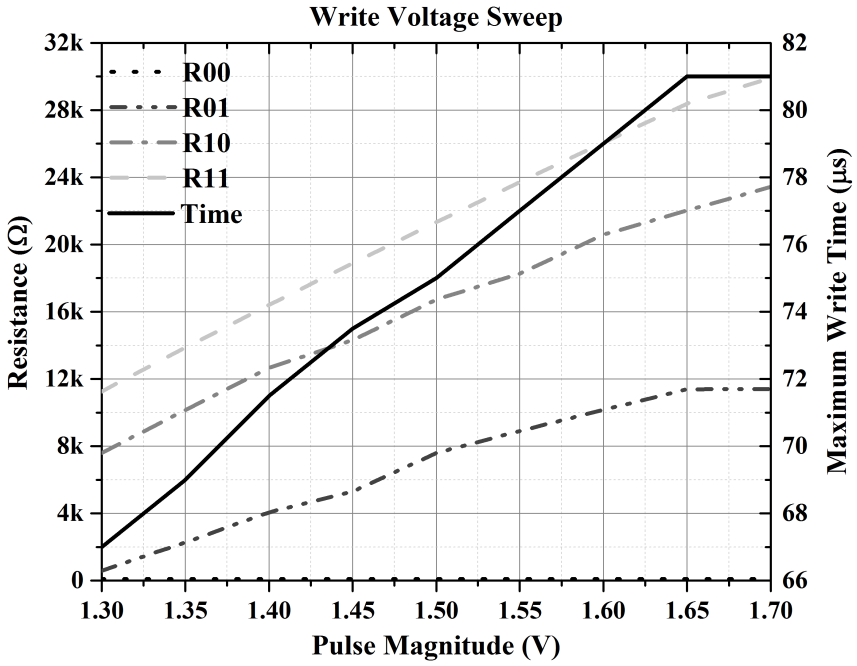

Figure 2.9: Write voltage sweep. How the resistive states change is shown with dashed and dotted lines. Maximum write time which corresponds to writing of state '11' is shown with straight line.

2.4 Effects of Variations

2.4.1 Variations in Programming Voltage

As indicated by expressions (2.3), (2.5) and (2.7), the programmed states are dependent on the voltage levels used. If the voltage bias levels are scaled, the programmed states change accordingly as shown in Figure 2.9.

The selected row and column pulse magnitudes are kept equal and are listed in the x-axis while their signs are opposite. Although it is possible to scale the unselected row and column voltages separately, they were reduced or increased the same amount as the selected row and column voltage magnitudes. It was found that this scaling scheme provided more even scaling of the applied voltages to the unselected and half selected cells.

The results show that as the voltage levels are increased, the programmed resistance levels increase. This also results in increase in the programming time of the cell. As the resistance levels increase, the separation between the states increases, which can allow for better margin of detection.

Since a read operation is a write operation with an extremely short duration, the results reported in Figure 2.9 have important consequences in terms of improving the tolerance for resistance drift in both increasing and decreasing directions. If a cell is written at a pulse magnitude of 1.6V, it can still be read at, for example, 1.55V while allowing drift in its resistance. This can be further clarified by comparing the programmed resistances for both voltage levels as shown in Table 2.2.

	1.6V	**1.55V**	**$\mathbf{MARGIN_H}$**	**$\mathbf{MARGIN_L}$**
'00'	100 Ω	100 Ω	8801 Ω	-
'01'	10166 Ω	8901 Ω	8099 Ω	1265 Ω
'10'	20590 Ω	18265 Ω	3125 Ω	2325 Ω
'11'	26067 Ω	23715 Ω	76285 Ω	2352 Ω

Table 2.2: Comparison of Resistive States

The '11' state programmed at 1.6V has resistance of 26067 Ω. However, if this programmed resistance is read at 1.55V, it would yield a value of '11' since 26067 Ω is greater than the '11' value written at 1.55V which is 23715 Ω. In fact, reading at 1.55V allows the programmed resistance of 26067 Ω to drift up to the device maximum, which is assumed to be 100 $K\Omega$, or down to the boundary of state '10', which is 23715 Ω.

Table 2.2 lists the high and low resistance margins that are the distances of the programmed resistances of the memory states at higher voltage level to the detection boundaries of the lower voltage reads. It is possible to optimize these margins by scaling the circuit parameters that contribute to the resistive states as characterized in Section 2.2.5.

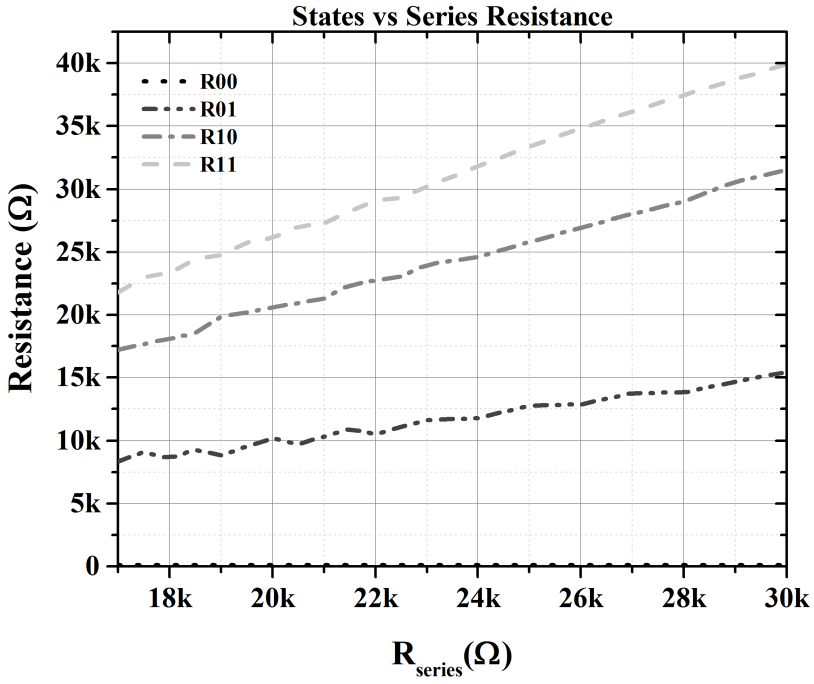

Figure 2.10: Series Resistance Sweep. The change in programmed states vs the series resistance value is shown for fixed write voltage levels.

2.4.2 Variations in Series Resistance

The programmed states are also dependent on the series resistance used, similar to the case of pulse magnitudes. The change in programmed resistances versus the series resistance is shown in Figure 2.10.

For fixed write pulse magnitude of 1.6V, the increase in series resistance results in increase in programmed resistances. The same trend as in the case of increased voltage levels is observed. The separation between the states increases with increased series resistance, which can allow for better margin of detection.

Similar to the case of voltage reduction, the reduction of series resistance when reading allows for generation of high and low margins, which in turn allows for drift in both increasing and decreasing directions. An example is not provided for this case in order to avoid repetition; however, similar results as in Table 2.2 can be deduced from values plotted in Figure 2.10.

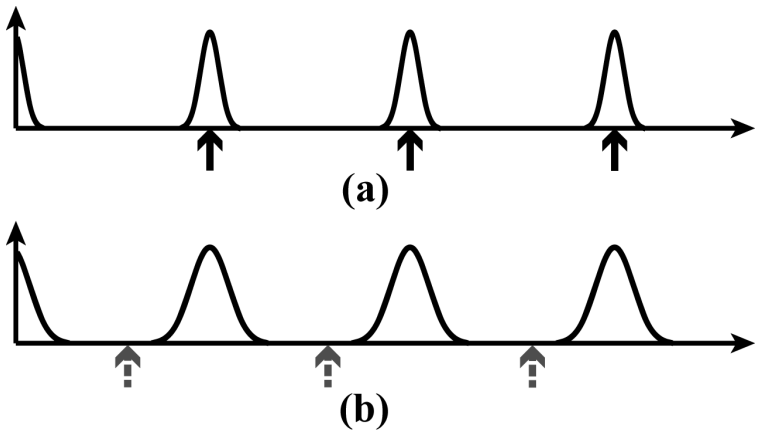

Figure 2.11: Resistance distributions when a single device is programmed in the whole array (a), after all the cells are programmed in the whole array (b). The black arrows indicate a sample cell resistances. The grey dashed arrows indicate where the lower boundaries of the resistive states shift when reduced-impact read scheme is used.

2.4.3 Reduced-Impact Read Scheme

Even more powerful benefits can be obtained when both the read voltages and the series resistance are scaled simultaneously.

Figure 2.9 indicates that it is not possible to scale the read pulse magnitudes to 1.5V because the lower boundary for '11' state is 18891 Ω which is lower than the '10' boundary (20590 Ω) for 1.6V writes. This means '10' written at 1.6V would be evaluated as '11' at 1.5V reads. However, the increase in series resistance shows the opposite trend as in the case of decrease in read voltage. Therefore, it is possible to perform reads even below 1.5V pulse magnitude with the help of increased series resistance.

The simulations indicated that it is possible to write the values at 1.6V with $20K\Omega$ resistance, and then read at 1.5V pulse magnitude with $25K\Omega$ series resistance or at 1.25V pulse magnitude with $45K\Omega$ series resistance.

Figure 2.11 aims to visualize how the reduced-impact read scheme works together with the write scheme. When a single cell is programmed in the array, its resistance can land anywhere in the distributions shown in Figure 2.11a, depending on its loca-

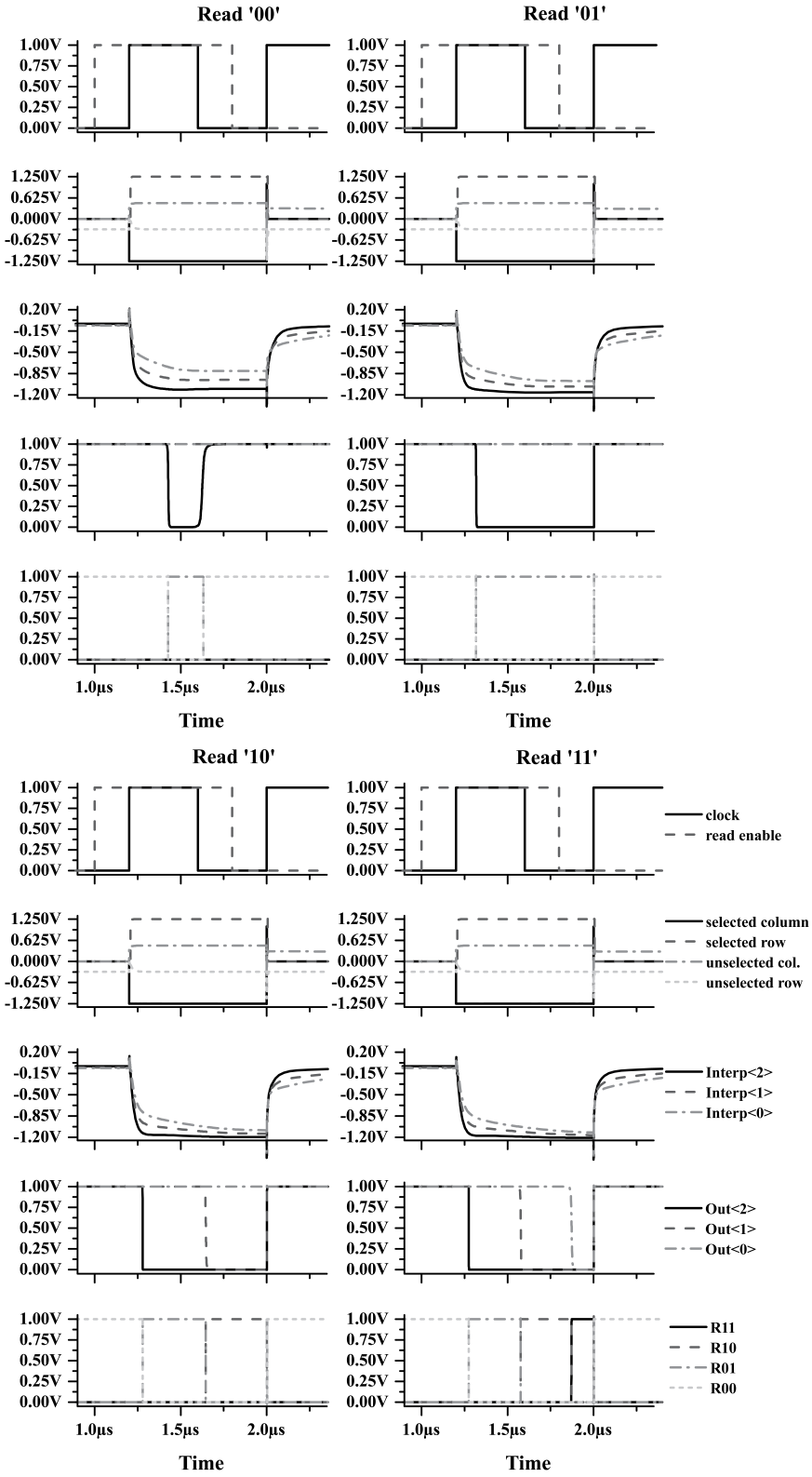

Figure 2.12: Reduced impact read operation. Relevant signals are shown for each state.

tion in the array. In this case, the variation in resistances is due to the change in the lumped crossbar parasitic resistance seen by the interpreter circuitry, depending on the location of the memory cell in the array. As more cells are programmed in the array, the resistances of the previously-programmed cells start to drift due to write disturbances, causing the spreading of the resistance distributions as shown in Figure 2.11b. Most approaches in literature fail to address this spreading; however, write simulations on the whole array are performed sequentially to show the spreading and shifting of the states, and the results are presented in the next part of this section.

The reduced-impact scheme aims to shift the lower detection margins such that the new distributions fall completely within their intended resistive states.

Figure 2.12 shows the reduced-impact read operations performed on cells storing the four possible values. A cell located at the middle of the array is selected. The scheme uses 1.25V pulse magnitude with $45K\Omega$ series resistance. In each case, when the controller receives the read enable signal, it facilitates the application of the bias voltages on the array. How Interp signals settle depending on the value that the cell is storing is shown. Interp signals settle in decreasing order, hence Out signals flip in the decreasing order as well. Signals R11, R10, R01 and R00 are the controller signals indicating whether a particular state is detected. The sequential flipping of Out signals also cause these signals to flip; however, the correct result is obtained at the end of the read operation.

2.4.4 Resistance Distributions after Array Writes

The consecutive write operations on the neighboring cells cause the cell resistances to drift over time. Even though unselected and half selected cells are biased with low voltages, the leakages during the lengthy write durations add up and cause the array resistances to drift.

In order to quantify this drift, consecutive write operations are performed on every

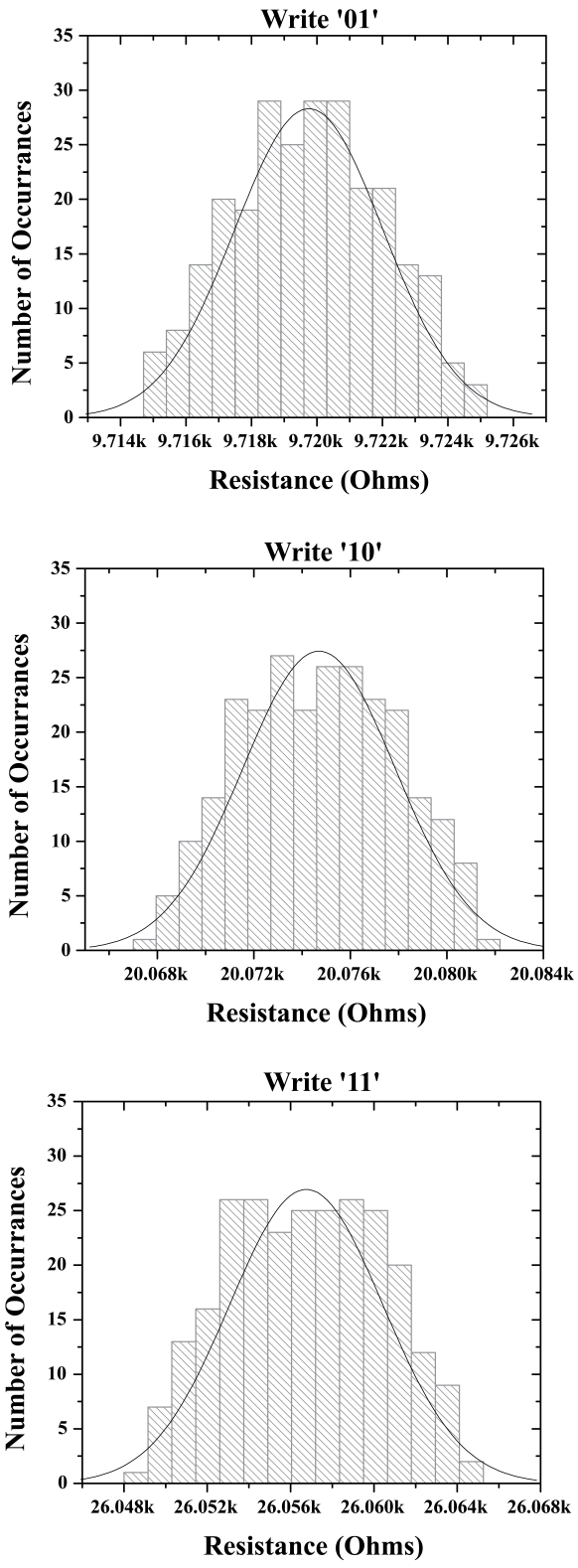

Figure 2.13: Resistance distributions after programming the entire array with the same value.

cell of the array. The resulting resistance distributions are listed in Figure 2.13. The distributions include the effects of writes to the neighboring cells and the parasitic crossbar resistances as these values change depending on the location of the selected memory cell. The variations from the nominal value are within the margins that can be obtainable with the reduced-impact read scheme proposed in this section. Therefore, it is concluded that it is possible to write to the cells in the whole array, and then read the values using the presented read scheme while allowing margin for additional drift that can be caused by repeated read operations.

2.5 Conclusion

In this chapter a multi-level memristor memory architecture that incorporates a novel read-monitored-write scheme was presented. The architecture allows for programming of the cells to the detection thresholds with very tight state distributions. In addition, an analytical model for state derivations which can be utilized to decide on the component parameters to be used in the design was presented. Various read schemes where voltage reduction or series resistance increase or a combination of both can be used to evaluate the resistive state of the memory cell were also presented. Finally, the resistive distributions after the entire array is written were shown. These distributions fall within the detection margins of the proposed read methods.

CHAPTER III

Image Processing by a Programmable Artificial Retina Comprising Quantum Dots and Variable Resistance Devices

3.1 Introduction

Feature extraction is a fundamental task in vision systems as extracted features provide bases for correlation. In digital general purpose processors, many image processing applications require an immense number of operations per second, albeit these applications do not require floating point accuracy [52]. Use of fast, simple and relatively accurate extraction systems in vision machines directly reduces the processing time and required iterations. The main processor element can thereby rely on the reduced dataset that provides quality information on the extracted features for decision making.

Inherent parallel processing capabilities of Cellular Nonlinear Network (CNN) based architectures make them an efficient platform for various image processing tasks [53, 54]. Real time operation provides fast processing times, and local connections provide simplicity, scalability and power efficiency for VLSI implementations [55]. Therefore, much effort has been put into developing novel methods and finding adequate CNN templates to perform detail extraction tasks in vision systems, such

as edge detection [56, 57, 58] which benefit greatly from immense parallelism and computational efficiency.

Resistive grid based architectures are shown to provide simple yet efficient ways to perform many image processing tasks and motion detection, and they are simple forms of CNNs [53]. Additional advantages including compact area, noise immunity and lower power consumption compared to digital computation structures, make them attractive for researchers. They are also relatively insensitive to mismatches in component values in VLSI chips [59]. However, most of the resistive grid based architectures in literature are static application specific structures and do not have the functional flexibility of their digital counterparts. Therefore, novel methods and devices should be introduced in these architectures to achieve functional versatility.

Resonant tunneling diodes (RTDs) have been employed in many applications including various CNN architectures due to their negative differential resistance (NDR) and fast switching characteristics. In [60] RTDs have been introduced as variable resistors to introduce versatility and compactness to CNN unit cells. In [61] a CNN architecture employing RTDs is investigated for its operation and it is shown that RTDs support fast settling times for various image processing applications.

In this chapter, a variable resistance grid-based architecture which improves the velocity tuned filter (VTF) architecture proposed by our group [62] is presented. It is demonstrated that when variable resistance connections are incorporated, various diffusion characteristics are obtained, and the developed architecture can be programmed for different image processing applications such as edge detection and line detection providing flexible analog processing environment that can perform various tasks. In addition RTDs are utilized to provide high speed signal detection and amplification. A method to program resistive connections in four directions is also demonstrated.

3.2 CNN Architecture

3.2.1 Resonant Tunneling Diode Model and Biasing

RTDs have been employed in many circuit applications utilizing their fundamental characteristic of negative differential resistance (NDR). NDR implies that for certain range, the increase in applied voltage across an NDR device will result in decreased current through it, indicating increased resistance with increased voltage.

RTD conductance is determined by two mechanisms: The first mechanism is resonant tunneling, which provides the NDR characteristic, and the other mechanism is diode conduction.

The NDR property of the RTD I-V characteristics is shown in Figure 3.1 utilizing the physics-based model laid out in [63]. The RTD current $J_{RTD}(V)$ is given by:

$$J_1(V) = \frac{qm^*kT\Gamma}{4\pi^2\hbar^3} ln(\frac{1+e^{(E_F-E_r+n_1qV/2)/kT}}{1+e^{(E_F-E_r-n_1qV/2)/kT}}) * (\frac{\pi}{2} + arctan(\frac{E_r - n_1qV/2}{\Gamma/2})) \quad (3.1a)$$

$$J_2(V) = H(e^{n_2qV/kT} - 1) \quad (3.1b)$$

$$J_{RTD}(V) = J_1(V) + J_2(V) \quad (3.1c)$$

where $J_1(V)$ is the current due to resonant tunneling and $J_2(V)$ is the diode conduction current. E_F is the Fermi energy, E_r is the resonant level energy, Γ is the resonant width, n_1 and n_2 are empirical model parameters. q, m^*, k, T, $\hbar$ are electron charge, effective mass, Boltzmann constant, absolute temperature, and reduced Planck constant respectively. V is the voltage across the device.

The main advantage of the NDR characteristic becomes apparent when RTD is biased with a static current source. If current magnitude of the source is selected such that it intersects RTD I-V curve in three places as shown in Figure 3.1, two stable voltage points are obtained. This result indicates that for the same amount

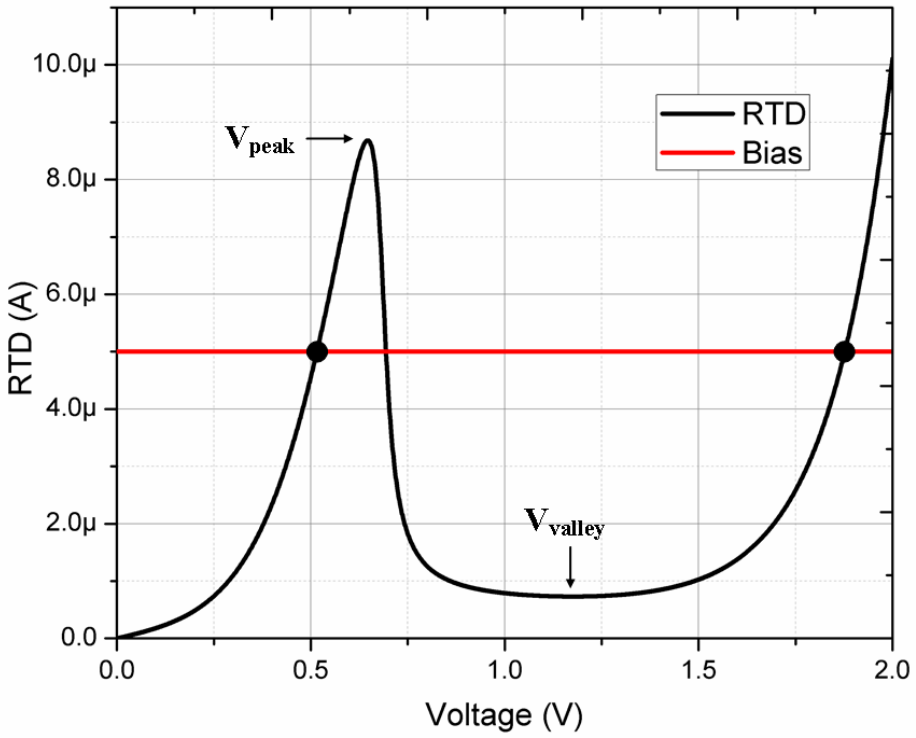

Figure 3.1: RTD I-V curve. The red line corresponds to the bias level. The intersections of the two lines represent the stable operating points.

of current passing through RTD, the voltage across it can take two stable values which correspond to the lowest and highest voltage intersection points. RTD does not stabilize in the middle intersection point, since any small disturbance causes it to switch to one of the outer intersection points.

The bistable characteristic of this structure can be utilized to build voltage level detectors since any voltage below switching threshold results in stabilizing in the low state, and any voltage above threshold results in stabilizing in high state. RTD switching threshold can be approximated as:

$$V_{th_{RTD}} = \frac{V_{peak} + V_{valley}}{2} \tag{3.2}$$

where V_{peak} and V_{valley} are the peak and the valley voltages of the RTD, respectively. When used in the detection mode, as the system starts all the RTDs are biased to the low voltage state, and a controlled disturbance toward a higher voltage results in the RTDs stabilizing at the higher stable level, allowing the detection and locking of the signal state.

3.2.2 Unit Cell Structure

Figure 3.2a shows the proposed unit cell structure. It is composed of variable resistance devices to provide resistive connections to neighboring cells, diodes to introduce unidirectionality to these connections, and RTDs to detect and latch signal levels. The proposed cell has an input node denoted by $I_{n,m}$, a center node $C_{n,m}$, and an output node $O_{n,m}$. The input is driven by voltage signals that correspond to the pixel intensity level which can be generated by a photo-detector. The connections employing variable resistance devices provide programmability for the realization of different characteristics including isotropic, anisotropic symmetrical and

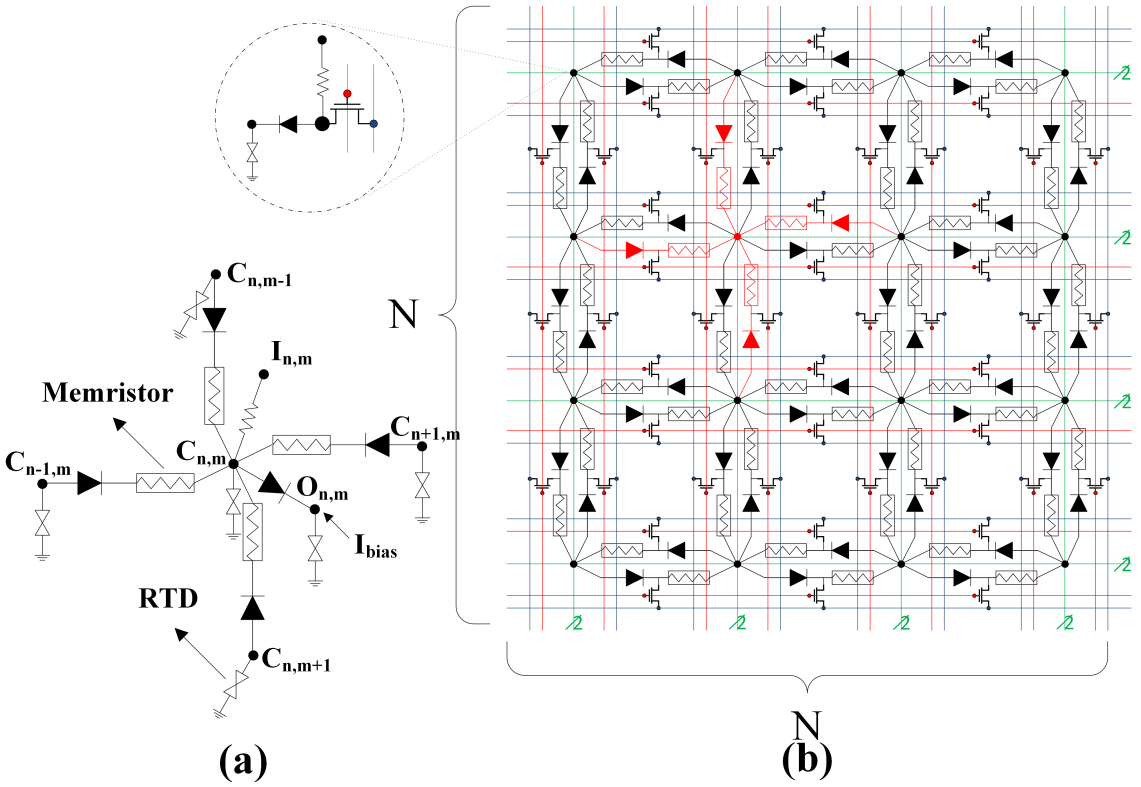

Figure 3.2: (a) Variable resistance unit cell (b) Top view of the processing array. A unit cell is highlighted in red. Red and green lines denote programming connections to access transistors.

asymmetrical diffusion in grid architecture giving way to various spatiotemporal filter implementations.

Four variable resistors are connected to the center nodes of the unit cell and its neighbors, making the center node voltage a function of the center node voltages of the neighboring cells. Resistances of connections determine how much neighbors' center voltages contribute to the center voltage of the cell. Series diodes allow current in one direction separating how outputs of the two neighboring cells affect each other. The output node is isolated from the center node by a diode providing a voltage barrier equal to the diode threshold. RTDs enable detection and latching of output signals. When biased with a current source, RTDs initially settle at the lower stable voltage. When the voltage level on the center node goes above detection threshold,

RTDs settle at the higher stable voltage. Two stable states provide a binary output. The detection threshold is equal to the sum of the diode threshold and the switching threshold of the RTD.

Figure 3.2b shows unit cells connected in a 2D array fashion. A top view for a 4x4 sample processing array is provided to show the neighboring connections. The unit cell shown in Figure 3.2a is highlighted in red. In order to program certain functionalities in the array, the resistances need to be altered. The Green lines in Figure 3.2b indicate the programming connections to the cells. Each green connection denotes programming-enable signal and voltage driver connections. Access transistors are used to isolate the connections during normal array operations. Programming connections can also be made to share the same connections as cell inputs, thus reducing number of access transistors if the input resistances are designed to be small at the expense of increased programming time or increased programming voltages due to voltage drop across the input resistor. Connections shown in red and blue as well as access transistors are needed during cell erase to bypass reverse biased diodes. During erase operation voltage polarity across the variable resistance device is reversed.

3.3 Programming Variable Resistance Connections

To be able to implement different processing tasks in the same array, we need a procedure to program the resistances of the resistive connections.

Figure 3.3 shows the programming flow for an N x N array. Programming is performed in four directions (left-to-right, right-to-left, top-to-bottom, and bottom-to-top) one direction at a time. Programming of the whole array is completed in four passes across the array in different directions to change the resistances in these directions. While a pass is being made in one direction, the resistances are set in a column by column fashion. Programming in this fashion reduces the total required

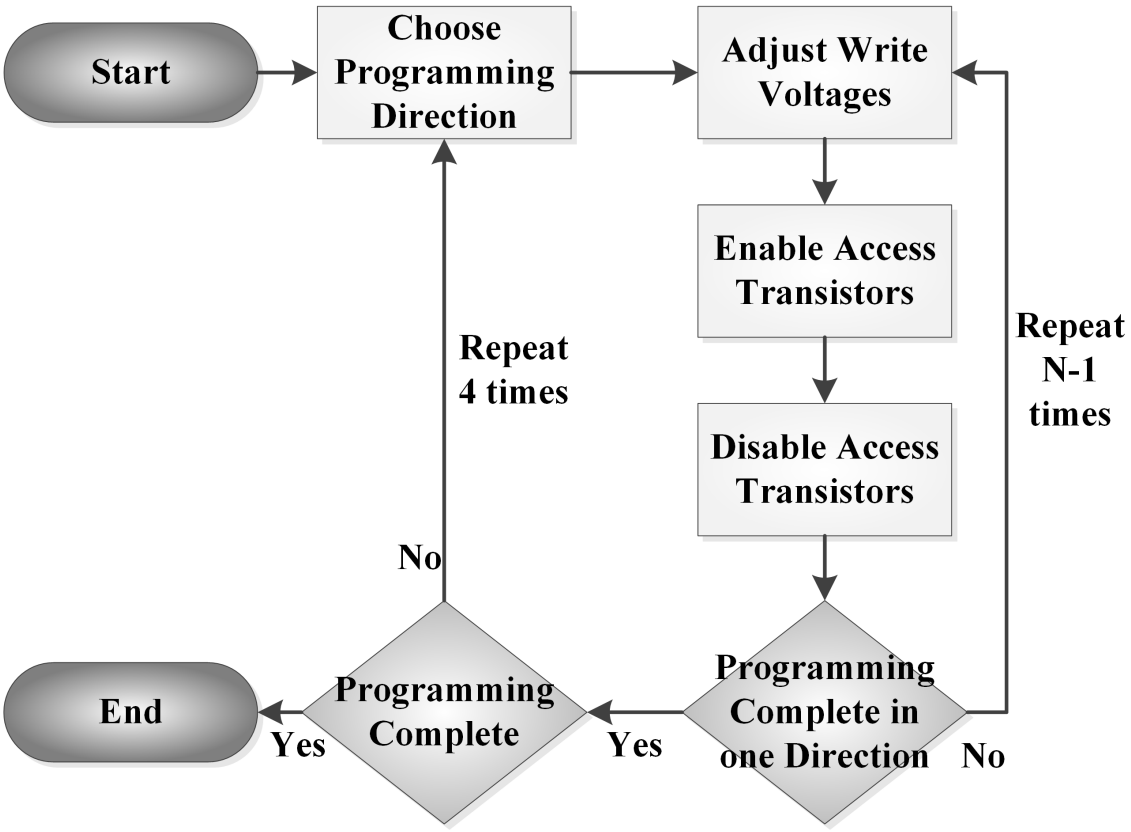

Figure 3.3: Array programming flow

time significantly compared to programming every resistive cell in the array individually. The duration and voltage amplitude of the write pulses determine the resistance to be stored in the connections.

Within one direction, same voltage amplitudes and pulse durations are used. However, pulse characteristics can be changed in different directions to program different resistances, hence to program different functionalities to the array.

A sample programming operation in the left to right direction is shown in Figure 3.4. All the connections in the array are initially at the low resistance state. The programming begins by setting the first column write voltage to high (indicated with a red line) and the remaining columns (indicated with a green line) to low (0V in our implementation).

In this configuration the first column of connections observe a non-zero voltage difference across, whereas the remaining connections observe zero voltage difference. In the first column, only half of the connections are programmed due to the fact that half of the series diodes are forward-biased conducting high currents, and the other half are reverse-biased.

Once the resistances of the first column connections reach the desired level, the second column write voltage is set to high, making the voltage difference across these connections zero, thus stopping their programming. The rise of the voltage levels on the second column in turn causes the voltage difference across the next column connections to be non-zero. Once these connections reach the desired resistance, the next column's voltage is raised. This process is repeated until all the connections in the selected direction are programmed.

When programming in the selected direction is completed, another direction is selected and the same process is repeated in this new direction. The use of different voltages or change of voltage raise-durations result in programming of different resistances in this direction.

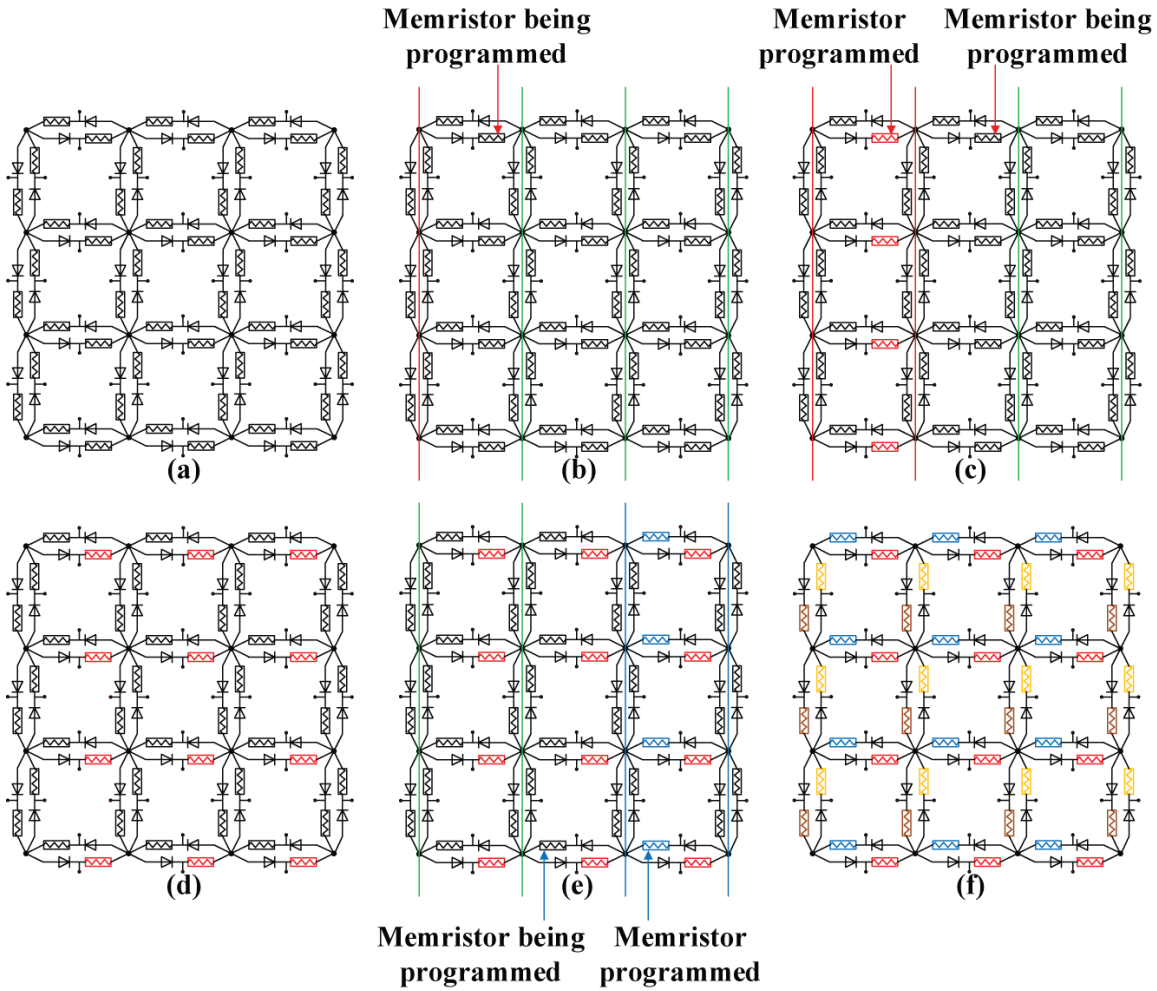

Figure 3.4: Programming in one direction. The green lines indicate the low voltage level (0V); other colors indicate altered high voltage levels. The different colors of variable resistance devices indicate different final resistances: a) Array in initial state, b) Programming started in left to right direction, c) Programming of first column completed in left to right direction, d) Programming of all connections completed in left to right direction, e) Programming of first column completed in right to left direction, f) Whole array after all connections are programmed in all directions

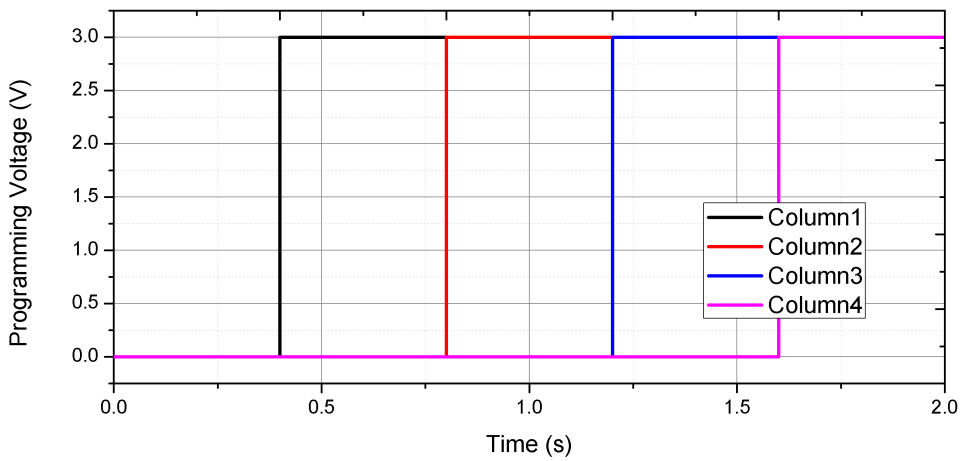

Figure 3.5: Programming in one direction in a 4 by 4 array: Programming voltages

Figure 3.5, sample left-to-right direction programming voltages to the array are shown. As described earlier, write voltages are applied per column basis. Voltage levels are increased with same time intervals.

Figure 3.6 shows how the resistances of the connections change. The programming scheme succeeds in tuning all the connections in the same direction to the same resistive state.

In order for the proposed method to be feasible, two critical requirements must be met: The first requirement is that the variable resistive device should be able to be programmed even when there is a forward-biased diode connected in series. The second requirement is that the resistive state of the device should not change or should change negligibly when there is a reverse-biased diode connected in series.

Figure 3.7 shows effect of having a series diode with the variable resistance device while performing a programming operation. The results indicate that having a forward-biased series diode with the device causes the device to be programmed to a lower resistance than when programmed with no series diode. This reduction in resistance can be compensated for by increasing programming time or voltage amplitude. This result indicates that it is still possible to program connections with series diode.

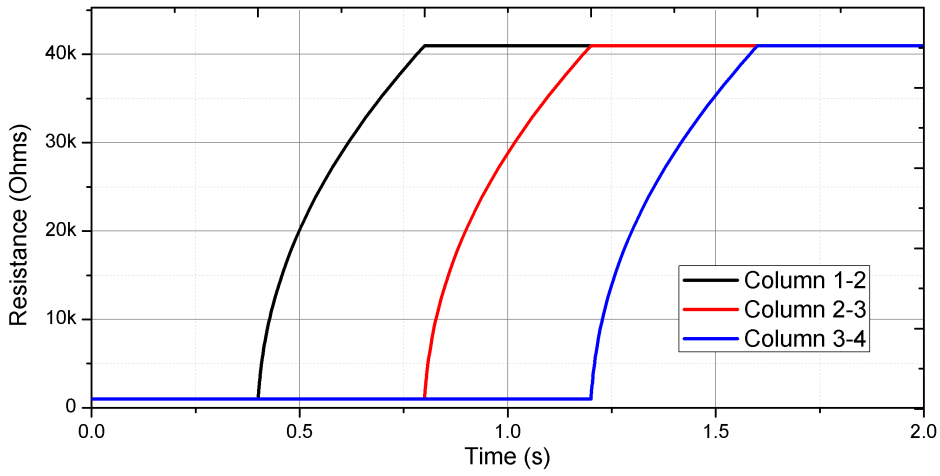

Figure 3.6: Programming in one direction in a 4 by 4 array: Resistances in the same row

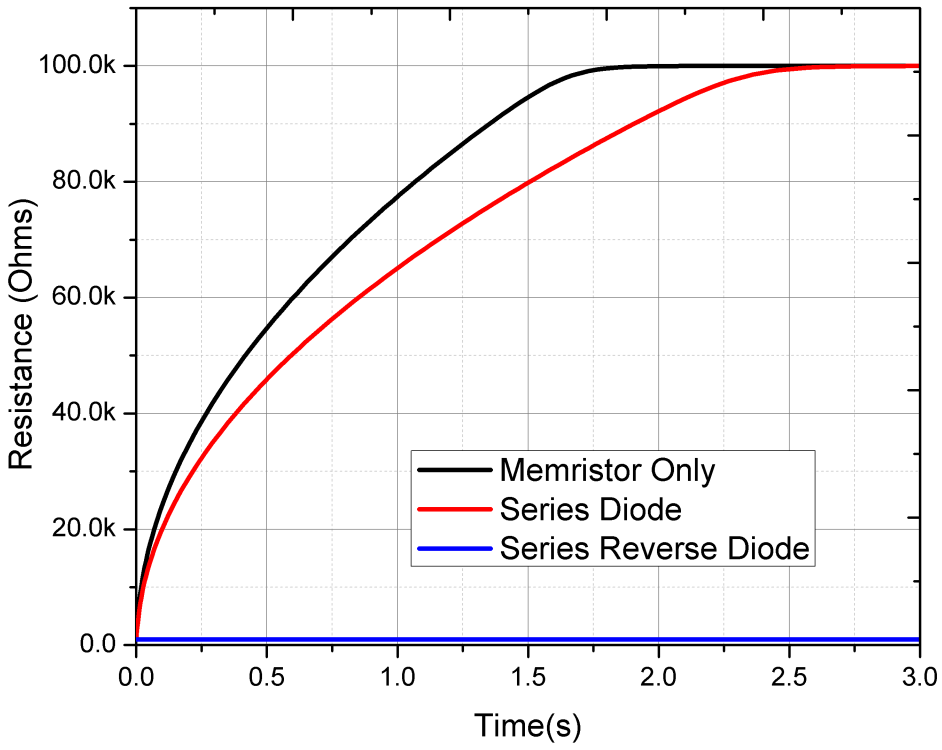

Figure 3.7: Connections under different bias conditions

A series reverse-biased diode causes no significant change in the resistance of the device during programming, effectively shielding it from the high voltage bias. This property enables programming connections in opposite directions possible, which is crucial in our proposed scheme.

3.4 Analytical Modeling

3.4.1 Edge Detection

Edge detection provides physical information about object boundaries in processed images and is a fundamental feature extraction task in vision systems. An edge is located at the transition points between two different intensity levels.

The grid provides diffusion characteristics that can be adjusted by controlling the resistances. These characteristics combined together with bistable RTD biasing can be used to implement various image processing tasks including edge detection. When all the memristors are programmed to the same resistance, the grid shows symmetric diffusion properties that can be applied to detect edges or contours of an input image.

An edge exists whenever a low input is neighbored by a high input, since an edge is defined where the discontinuity between the input voltages occur.

A simplified analysis on one dimensional connection (Figure 3.8) is carried out to show that this structure can be used for edge detection. In the figure, $I_{n,m}$ is the input voltage level, $C_{n,m}$ is the center node voltage level and $O_{n,m}$ is the output voltage level for the nth node.

It is assumed that there is an edge between inputs $I_{n,m}$ and $I_{n+1,m}$. Thus, $I_{n,m}$ and inputs before it are high and $I_{n+1,m}$ and inputs after it are low (0V for this analysis purposes).

The analysis starts by applying Kirchoff's Current law to nodes $C_{n,m}$ and $C_{n+1,m}$ to obtain the node voltages. Initially the effects of the nearest neighborhood are

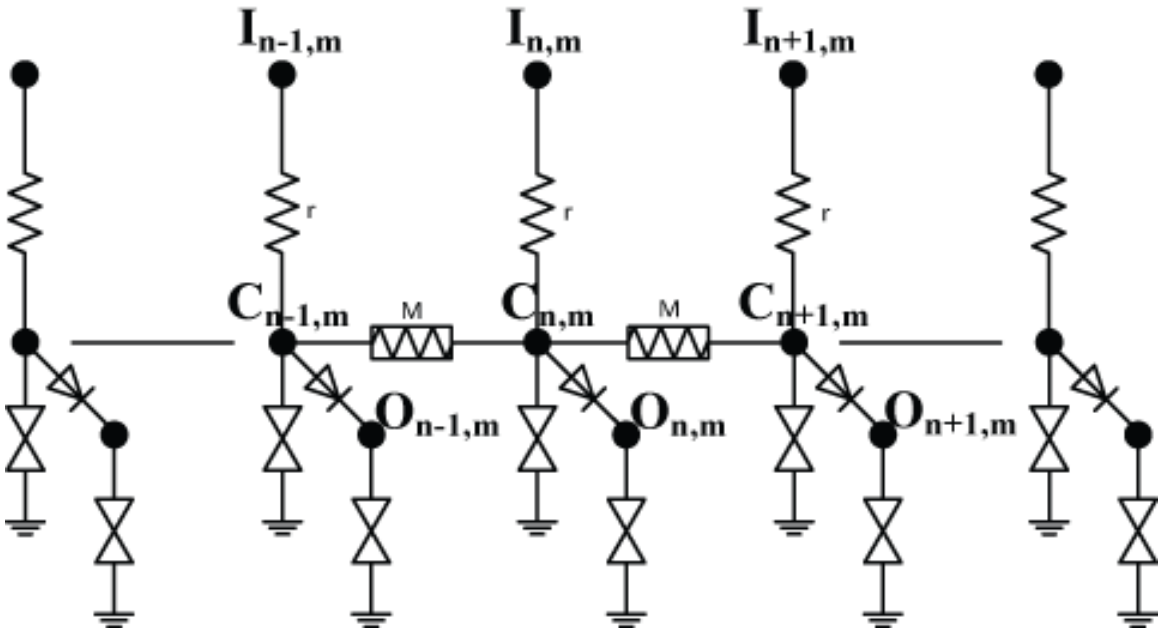

Figure 3.8: CNN circuitry in 1D case

ignored.

$$\frac{I_{n,m} - C_{n,m}}{r} = \frac{C_{n,m} - C_{n+1,m}}{M} + \frac{C_{n,m}}{R_{RTD_1}} \tag{3.3}$$

where r is the input resistance, M is the resistance of the device, R_{RTD_1} is the effective resistance of center node RTD, and $C_{n,m}/R_{RTD_1}$ indicates current through RTD. The current branch through the diode to the output nodes is also ignored, since the diode current is order of magnitude less until the center node voltage reaches the switching threshold at the output node. When the RTD current equation is inserted, the equation becomes:

$$\begin{aligned} &\frac{I_{n,m} - C_{n,m}}{r} = \frac{C_{n,m} - C_{n+1,m}}{M} + \frac{qm^* kT\Gamma}{4\pi^2\hbar^3} ln(\frac{1 + e^{(E_F - E_r + n_1 qC_{n,m}/2)/kT}}{1 + e^{(E_F - E_r - n_1 qC_{n,m}/2)/kT}}) \\ &* (\frac{\pi}{2} + arctan(\frac{E_r - n_1 qC_{n,m}/2}{\Gamma/2})) + H(e^{n_2 qC_{n,m}/kT} - 1) \end{aligned} \tag{3.4}$$

Similarly at node $C_{n+1,m}$:

$$\frac{C_{n,m} - C_{n+1,m}}{M} = \frac{C_{n+1,m}}{r} + \frac{C_{n+1,m}}{R_{RTD_2}} \tag{3.5}$$

and

$$\begin{aligned} &\frac{C_{n,m} - C_{n+1,m}}{M} = \frac{C_{n+1,m}}{r} + \frac{qm^*kT\Gamma}{4\pi^2\hbar^3} ln(\frac{1 + e^{(E_F - E_r + n_1 q C_{n+1,m}/2)/kT}}{1 + e^{(E_F - E_r - n_1 q C_{n+1,m}/2)/kT}}) \\ &* (\frac{\pi}{2} + arctan(\frac{E_r - n_1 q C_{n+1,m}/2}{\Gamma/2})) + H(e^{n_2 q C_{n+1,m}/kT} - 1) \end{aligned} \tag{3.6}$$

R_{RTD_2} is the effective resistance of output node RTD. Equations (3.4) and (3.6) can be evaluated numerically to obtain the intermediate node voltages $C_{n,m}$ and $C_{n+1,m}$. Designing what these voltages will be is essential to obtaining edge detection functionality. When there is an edge, the center node voltage $C_{n,m}$ should rise to disturb the detection node RTD.

Parameters should be picked such that:

$$C_{n,m} > V_{threshold} when I_{n,m} = V_{high} \tag{3.7a}$$

$$C_{n+1,m} < V_{threshold} when I_{n+1,m} = V_{low}(0V) \tag{3.7b}$$

$$V_{threshold} = V_d + V_{RTD} \tag{3.7c}$$

V_d is the diode threshold, and V_{RTD} is the switching threshold of the RTD. In this way, the output $O_{n,m}$ will switch to high stable point, indicating there is an edge and $O_{n+1,m}$ will remain at low stable point.

If the effects of the nearing neighbors are considered, one can see that since $I_{n+2,m}$ is also low, the actual voltage on the node $C_{n+1,m}$ will be lower than the above calculated value, hence not violating the condition $C_{n+1,m} < V_{threshold}$, but instead further helping to meet it. Similarly, $I_{n+1,m}$ helps node $C_{n,m}$ to be higher than

$V_{threshold}$. In 2-D case, the state equation of the diffusion circuitry can be obtained as

$$\frac{I_{n,m} - C_{n,m}}{r} + \frac{C_{n-1,m} - C_{n,m}}{M} + \frac{C_{n,m-1} - C_{n,m}}{M} + \frac{C_{n+1,m} - C_{n,m}}{M} + \frac{C_{n,m+1} - C_{n,m}}{M} - I_{RTD} = c\frac{ds_{n,m}}{dt} \tag{3.8}$$

where:

$$-I_{RTD} = \frac{C_{n,m}}{R_{RTD}} \tag{3.9}$$

and c is parasitic capacitance of the RTD. Assuming RTD has a finite resistance, replacing (3.9) in (3.8):

$$I_{n,m} = C_{n,m}(1 + \frac{4r}{M} + \frac{r}{R_{RTD}}) - \frac{r}{M}(C_{n-1,m} + C_{n,m-1} + C_{n+1,m} + C_{n,m+1}) + rc\frac{dC_{n,m}}{dt} \tag{3.10}$$

Taking Fourier transform, the transfer function is:

$$H(f_m, f_n, f_t) = \frac{S(f_m, f_n, f_t)}{E(f_m, f_n, f_t)} = \frac{1}{(1 + \frac{4r}{M} + \frac{r}{R_{RTD}}) - \frac{2r}{M}(cos(2\pi f_m) + cos(2\pi f_n)) + rc2\pi j f_t} \tag{3.11}$$

As the RTD I-V curve indicates, it acts as a positive variable resistor, indicating that the real part of the denominator of the transfer function is always positive.

3.4.2 Line Detection

Introducing anisotropy in the vertical and horizontal directions in the resistive grid allows the implementation of line detection. In order to detect lines, the center node voltages should be made a weaker function of the neighboring cells' center node voltages in one direction and a stronger function in the other. For example, high

resistance in the vertical, and low resistance in the horizontal direction limits the effects of the neighboring cells in the vertical direction and enables diffusion in the horizontal direction, which means the detection of lines in the horizontal direction.

Low resistance in the vertical and high resistance in the horizontal direction limits the effects of the neighboring cells in the horizontal direction and enables diffusion in the vertical direction, which means the detection of vertical lines. In this case, the diffusion network state equation becomes:

$$I_{n,m} = C_{n,m}(1 + \frac{2r}{M_{high}} + \frac{2r}{M_{low}} + \frac{r}{R_{RTD}}) - \frac{r}{M_{high}}(C_{n-1,m} + C_{n+1,m}) \\ - \frac{r}{M_{low}}(C_{n,m-1} + C_{n,m+1}) + rc\frac{dC_{n,m}}{dt} \tag{3.12}$$

where M_{high} is the resistance of the variable resistance device when programmed to high, and M_{low} is the resistance when programmed to low. The state equation is symmetrical for vertical and horizontal line detection cases.

3.5 Simulation Results

Simulation results verifying various diffusion configurations and demonstrating edge detection and line detection operations are presented in this section. Simulations are carried out on a 64x64 array. RTDs based on device characteristics shown in Figure 3.1 as well as the variable resistance device model from [8] are used. Simulations are carried out with nominal parameters to provide proof of concept. However, studies carried out in [59] show that resistive grid-based architectures are variation tolerant and can operate with non-optimal values. Therefore, the proposed architecture is expected to be variation tolerant. For example, for the edge detection case, this tolerance depends on how much margin is left between the designed high/low voltages and the threshold.

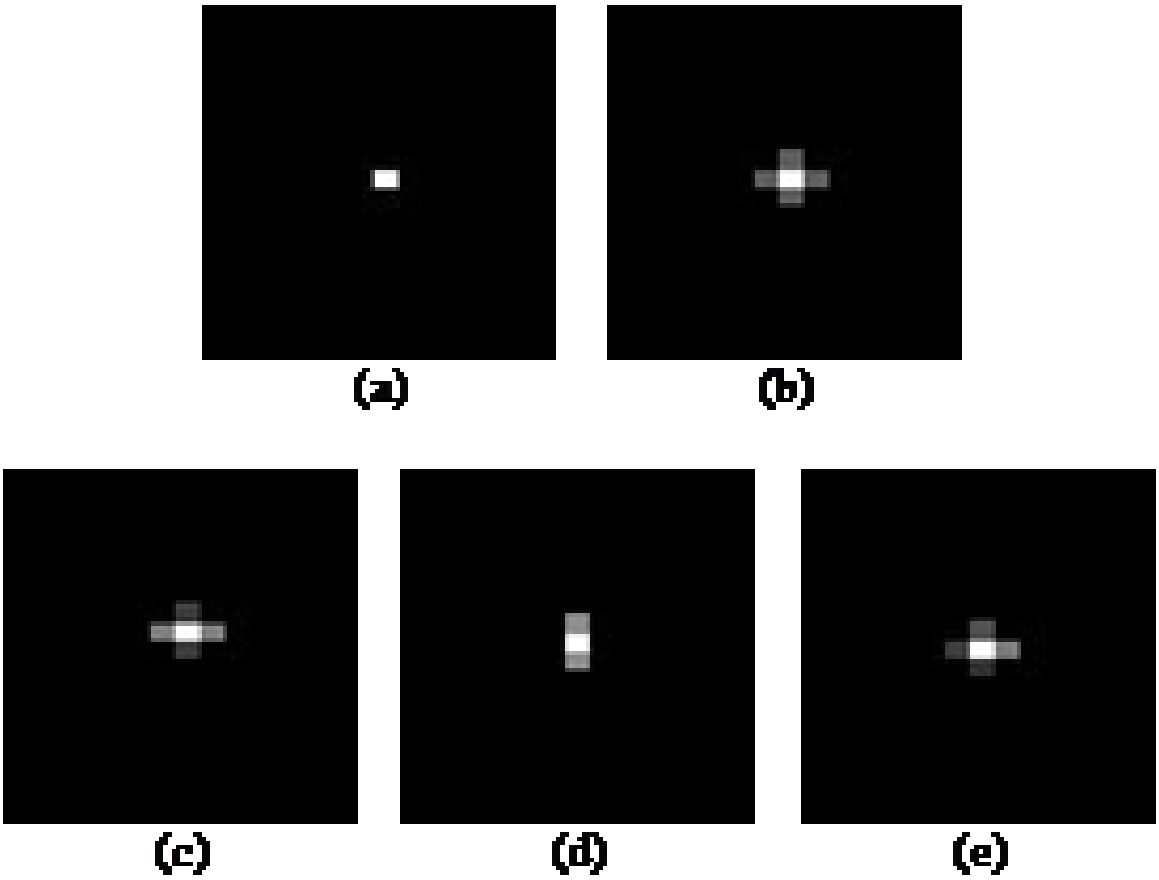

Figure 3.9: Various diffusion characteristics that can be implemented in proposed architecture. a) Input, b) Isotropic Diffusion, c) Anisotropic symmetrical diffusion in horizontal direction, d) Anisotropic symmetrical diffusion in vertical direction, e) Anisotropic asymmetrical diffusion

The proposed memristive grid can support the different diffusion characteristics mentioned above. These characteristics are important in many image processing applications. Anisotropic diffusion can be used for edge extraction applications [64], anisotropic symmetrical diffusion characteristics can be used for line detection, and anisotropic asymmetrical diffusion characteristics can be used for motion detection [65].

Figure 3.9 shows diffusion characteristics that can be realized in the proposed architecture. When all the connections are programmed to the same resistance, isotropic diffusion (Figure 3.9b) is obtained. When horizontal connectiions are programmed

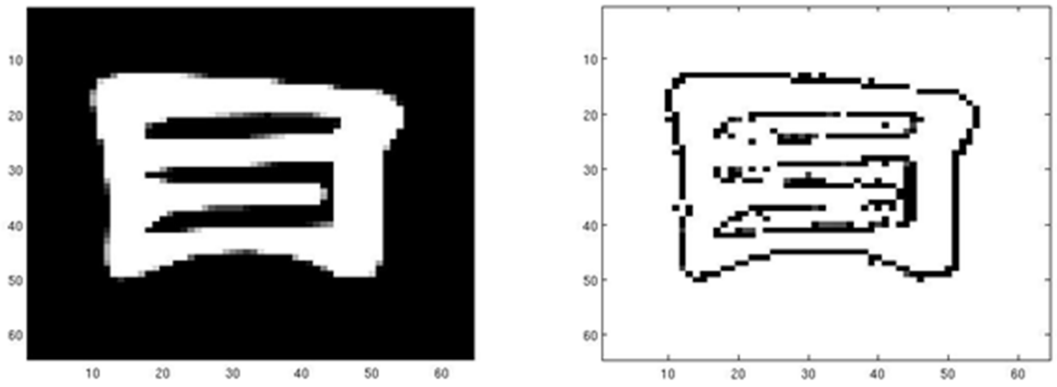

Figure 3.10: (a) Edge detection sample input with irregular edges, (b) Output result

to low and vertical ones are programmed to high resistance, anisotropic symmetrical diffusion in horizontal direction (Figure 3.9c) is obtained. When the resistances of the horizontal and vertical memristors are programmed reverse with respect to the previous case, anisotropic symmetrical diffusion in vertical direction (Figure 3.9d) is achieved. Finally when all resistances are programmed to different resistances, anisotropic asymmetrical diffusion (Figure 3.9e) is achieved. These characteristics or combinations of them can be utilized to perform various vision tasks.

3.5.1 Edge Detection

Edge detection simulations are carried out on two types of input images: First, with irregular edges and intermediate pixel intensity values around the edges shown in Figure 3.10; second, with regular edges and maximum pixel intensity difference around the edges shown in Figure 3.11.

In edge detection mode, the proposed architecture is initiated with high and low input values corresponding to black and white pixels with scaled voltages in between corresponding to shades of gray. When the first input type shown in Fig. 9a is applied to the array, the edge pattern shown in Figure 3.10b is observed at the $O_{n,m}$ nodes of

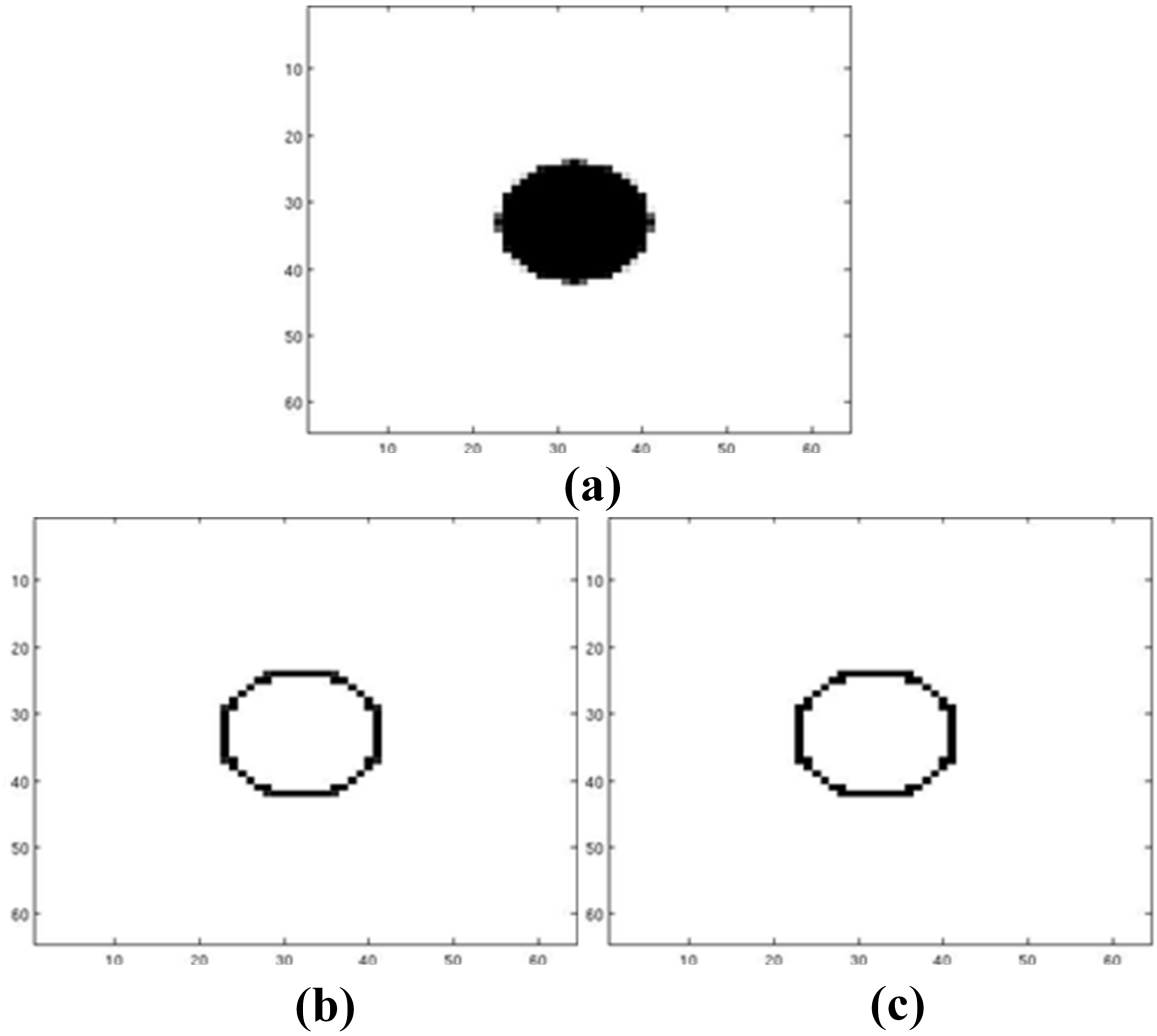

Figure 3.11: (a) Edge detection sample 2 with regular edges (b) Output at 2.5ns (c) Output at 100ns

the architecture. In Figure 3.10b, black lines correspond to high RTD voltage level and white lines correspond to low RTD voltage level on $O_{n,m}$ nodes.

The results of the above simulation suggest that edges are extracted with relative accuracy. In the regions with thicknesses of a few pixels or where the borders include shades of gray, discontinuities or jumps in the border lines are observed. However, the quality of the results can be improved by fine tuning grid resistance as well as RTD design parameters.

The second set of sample filtering is provided in Figure 3.11a-c. The second type of image contains a circle with large continuous areas of the same color pixels. In this image there is no region with several pixels thickness (except the outer line due to finite number of pixels available to represent a circle). The architecture is able to clearly outline the edges without any discontinuities.

Although the speed of operation can be tuned by scaling the current capabilities of the active devices, the above simulation shows that the results obtained after $2.5ns$ into the operation are almost exactly the same as the results obtained after $100ns$.

Simulations on the variable resistance devices using the model provided in [8] indicated that a read pulse of $100ns$ duration and $2V$ amplitude does not change the resistance detectably, and a write process usually takes in the range of a few seconds depending on the resistance to be encoded. Therefore, the architecture can perform the tuned operation repeatedly without detectably altering the tuning, thus minimizing (effectively eliminating) the need for a refresh operation.

The effect of component mismatch is more significant in input resistors (i.e vertical resistors) compared to grid resistors (i. e. horizontal resistors) [59]. A set of simulation results are listed in Figure 3.12a-f, showing how the variation in input resistance changes edge detection results. Simulation results indicate that edges are detected less accurately when the input resistance deviates from the optimum value obtained through simulations. Exact resistance values depend on various circuit and device parameters such as voltage levels used, RTD and diode current characteristics. Therefore, the resistance variation is presented in percentages. Edge detection quality directly depends on the variation of the input resistance mainly due to two factors. The first factor is that input resistance changes the spatial frequency tuning of the architecture making it less sensitive to edges. The second factor is that large input resistances cause input voltage drops, thus putting the architecture off the operating region.

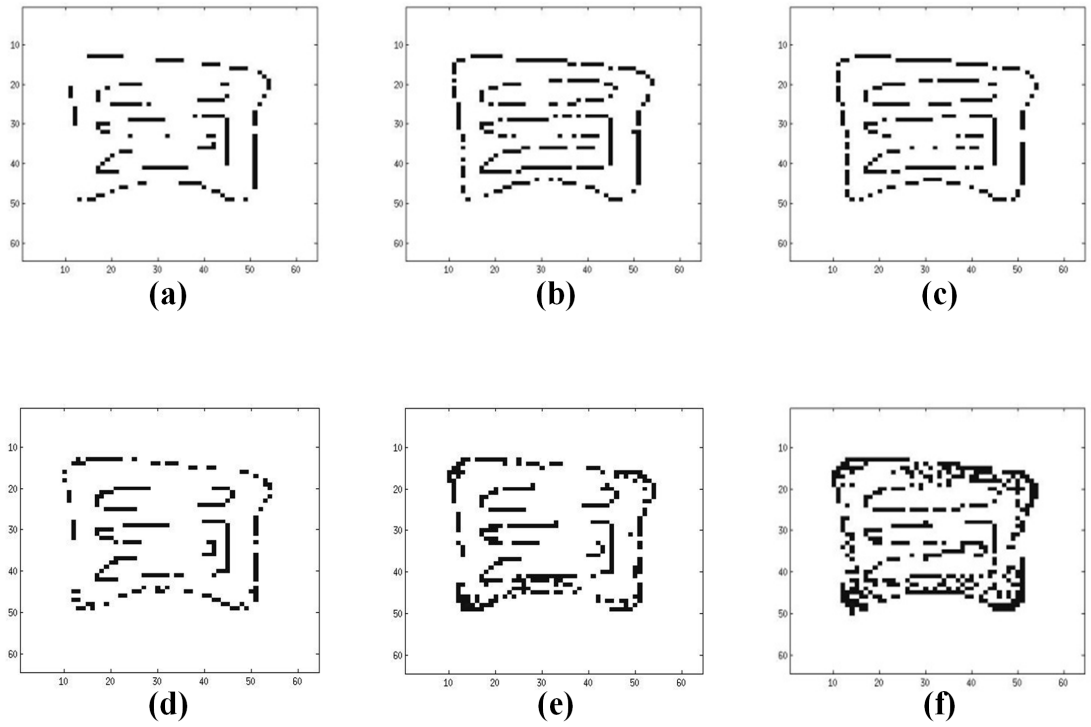

Figure 3.12: Edge detection results with input resistance variation: a) 50% resistance, b) 75% resistance, c) 100% resistance (nominal case), d) 500% resistance, e) 1000% resistance, f) 2000% resistance

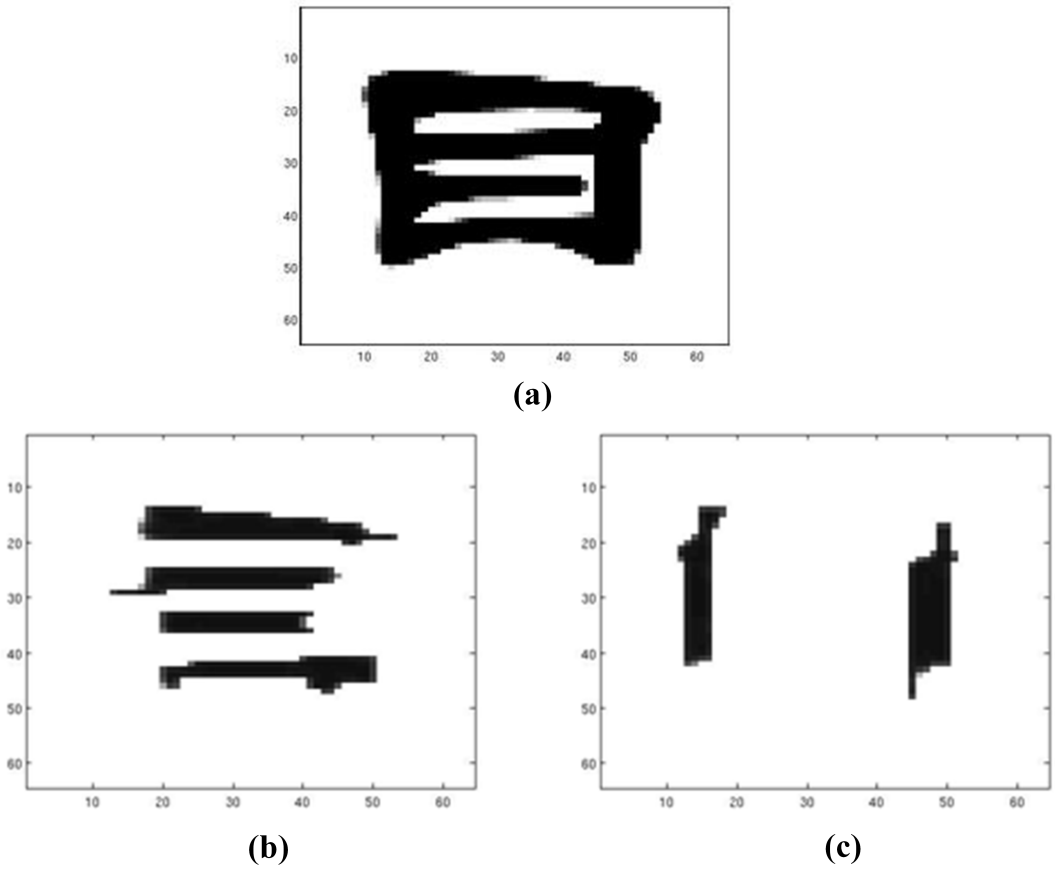

Figure 3.13: (a) Line detection sample input (b) Horizontal line detection (c) Vertical line detection

3.5.2 Line Detection

Figure 3.13 shows line detection results. The plotted results are obtained $500ns$ after system initialization. When the image in Figure 3.10a is considered, if we assume the black pixels represent the high voltages and white pixels represent the low voltages, the diffusion from high voltages to low voltages will result in the low voltages increasing and stabilizing at the higher voltage level. The results presented in Figure 3.13 are inverted to clearly show the detected lines.

3.6 Conclusion

An analog grid-based architecture incorporating variable resistance connections for programmability and RTDs for signal detection and latching was revealed in this chapter. The architecture can be programmed to perform various image processing tasks. A method to change the resistive state of the memristors in an array configuration was also provided and demonstrated through circuit simulations. Analytical models characterizing edge detection and line detection configurations were discussed and simulation results were provided to verify functionality. The simulation data establish that the presented architectural configuration incorporating programmable analog resistive elements can be reused to perform a wide gamut of image processing functions at extremely high speeds.

CHAPTER IV

Optimal Control via Neural Network Approximators

4.1 Introduction

Living organisms interact with their environment and they respond to the environmental stimuli. Some of the actions they perform as responses are rewarding, some of them are unrewarding, and some of them result in negative rewards or loss of rewards. These actions can include finding the best/largest amount of food, surviving predators, finding the best place for shelter, etc. Organisms learn from the results of their actions and adjust them to maximize their rewards.

This type of learning is commonly referred to as the 'Reinforcement Learning' [66]. Reinforcement learning has inspired many works in the field of control theory to mimic the optimal behavior observed in nature in the engineered systems[67, 68, 69, 70, 71, 72, 73].

A practical implementation of reinforcement learning is the 'Adaptive Dynamic Programming (ADP)'[74, 75]. ADP encompasses various iterative algorithms where value or policy iterations are performed to converge on the optimal control policy.

In this work, the main focus is on a particular type of ADP which is based on 'Actor-Critic Networks' where the actor network tries to perform the optimal action

and the critic network assesses the reward(value) of this action. The actor and critic networks can be realized via the utilization of 'Neural Networks'[76, 77, 78, 79, 80].

'Heuristic Dynamic Programming' (HDP) is a practical implementation of adaptive dynamic programming which is a form of reinforcement learning. HDP uses adaptive critics and actors to learn the optimum control action and the associated cost function in order to minimize or maximize a goal for a given system [81]. Neural networks trained using HDP algorithm can effectively learn and approximate arbitrary linear or non-linear behaviors of functions providing means for powerful neuro-control. The adaptive critics enable improvement of the performance over time during system operation through environmental interactions and past experience [82]. HDP has traditionally been adopted in software implementations of neural networks as it requires derivative calculations and large amounts of storage for weight calculations.

HDP has been used in various neuro-control applications. It is very powerful in terms of learning desired robotic behaviors defined by complex non-linear functions. It has been shown that HDP can be used to control robotic joints generating the optimum value for the force needed with respect to the mass and desired trajectory of the robotic arm [81]. Oscillations in a power system poses a threat in the overall stability of the system and can limit the power transfer capabilities [83]. HDP has been proposed to be adopted by these systems to effectively stabilize the system. The adaptive capabilities of the HDP enable the controller to adapt to the changing conditions and parameters with time [84]. HDP can also be used in bio-chemical plant control to ensure optimum process conditions are sustained during industrial processes [85]. HDP can provide online adaptation, enabling the controller to re-optimize to maximize the process performance [86].

The goal of this work is to provide an adaptive hardware for objective minimization/maximization, which will successfully perform HDP algorithm on chip. There has been great interest in hardware implementation of neural networks and their train-

ing mechanisms [87, 88, 89, 90, 91, 92, 93, 94]. However, to the best of our knowledge, a hardware implementation for HDP has not been proposed in literature. Such a hardware is critical for real-time learning tasks where a software implementation cannot provide the required learning speed. This work focuses on the development of a digital hardware composed of two neural networks (actor-critic) trained to approximate arbitrary non-linear functions which enable them to be utilized as global controllers for a wide range of discrete time applications.

A digital ASIC approach has been adopted, where the digital neurons have complex polynomial activation functions that allow the neural networks to approximate complex mathematical representations of non-linear control. Prior work suggests that digital neural network implementations provide higher accuracy, noise immunity, ease of testing and verification, adaptability, flexibility, as well as consistent repeatability [95, 96]. An analog or mixed signal design approach has not been incorporated in this work as analog designs do not scale very well to the latest fabrication technologies [97]. They are noisy and imprecise [98]. Also Analog/mixed signal design is mainly used for continuous time operation. It is hard, if not impossible, to make them correspond to mathematical models since they rely on transistor characteristics[99]. Therefore, analog design is not very suitable for discrete time applications.

The rest of this chapter is organized as follows: Section II provides a theoretical background on optimum control problem and heuristic dynamic programming. Section III goes into details of the proposed hardware architecture. Section IV provides simulation results to compare the hardware performance and accuracy versus software implementation. Finally, Section V provides concluding remarks.

4.2 Optimal Control via Adaptive Dynamic Programming for Discrete-Time Systems

A non-linear discrete-time system can be represented by the difference equation [66]:

$$x_{k+1} = f(x_k) + g(x_k)u_k \tag{4.1}$$

where k is the discrete-time index, $x_k \in R^n$ is the state and $u_k \in R^m$ is the control input. In order to obtain control inputs, a control policy should be assumed, which is a function defined from state space to contol space ($h() : Rn \rightarrow R^m$). Then the state dependent control action is:

$$u_k = h(x_k) \tag{4.2}$$

In ADP, this control policy is learned based on the stimuli the system observes in the environment.

4.2.1 Control Optimization

In order to asses the optimality of an action, a measure of performance called the cost-to-go function is defined as:

$$V_h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i) \tag{4.3}$$

where γ is the discount factor and suffices $0 < \gamma \leq 1$. $r(x_i, u_i)$ is the utility function and represents the cost of each action at a given time step. The expression in (4.3) is the sum of discounted costs for the current and future actions starting from current time and going to the infinity. The main goal of optimal control theory is to find a

control policy that minimizes the cost and yields the 'optimal cost':

$$V^*(x_k) = \min_{h()}(\sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i)) \tag{4.4}$$

The 'optimal control policy' that results in the optimal cost is therefore:

$$h^*(x_k) = \arg\min_{h()}(\sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i)) \tag{4.5}$$

Optimizing the cost for a single step is easy, however a decision based on only the current time step may not be the optimum decision when the sum of all costs from all time steps is considered. A decision based on optimizing the cost of current action can increase the costs of future actions, increasing the overall cost. Therefore, a decision should be made based on its effects on optimizing all costs in all time steps. However, obtaining the exact solution to this problem is close to impossible in difficulty for a general nonlinear system.

4.2.2 Bellman Equation

In order to transform the infinite sum presented in (4.3) to a more computation friendly form, one can write it as:

$$V_h(x_k) = r(x_k, u_k) + \gamma \sum_{i=k+1}^{\infty} \gamma^{i-(k+1)} r(x_i, u_i) \tag{4.6}$$

which is equivalent to the difference equation:

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}), V_h(0) = 0 \tag{4.7}$$

This equation is known as the Bellman equation. Solving the Bellman equation is the critical step in ADP to assess the cost of the current policy.

4.2.3 The Discrete-Time Hamilton-Jacobi-Bellman (HJB) Equation

The Bellman equation as presented in (4.7) is backward in time, meaning it has to be solved offline starting from the terminal state and by tracing the time-steps backward. $r(x_k, u_k)$ can be represented in the form of quadratic energy function, $r(x_i, u_i) = x_i^T Q x_i + u_i^T R u_i$. In this case the cost function becomes [100]:

$$V(x_k) = \sum_{i=k}^{\infty} (x_i^T Q x_i + u_i^T R u_i) \tag{4.8}$$

where Q and R are positive definite matrices with $Q \in R^{nxn}$ and $R \in R^{nxm}$. And the discrete-time HJB equation is

$$V^*(x_k) = \min_{u_k} (x_k^T Q x_k + u_k^T R u_k + V^*(x_{k+1})) \tag{4.9}$$

The optimal control u^* is

$$u^*(x_k) = \frac{1}{2} R^{-1} g(x_k)^T \frac{\partial V^*(x_{k+1})}{\partial x_{k+1}} \tag{4.10}$$

By substituting optimal control equation (4.10) in (4.9), the HJB equation can be rewritten in the form:

$$V^*(x_k) = x_k^T Q x_k + \frac{1}{4} \frac{\partial V^{*T}(x_{k+1})}{\partial x_{k+1}} g(x_k) R^{-1} g(x_k)^T \frac{\partial V^*(x_{k+1})}{\partial x_{k+1}} + V^*(x_{k+1}) \tag{4.11}$$

$V^*(x_k)$ is the cost function associated with the optimal control policy $u^*(x_k)$. For general nonlinear systems, the exact solution to HJB is unattainable. Therefore, various iterative approximation techniques are proposed in literature with their corresponding proofs of convergence to the optimal control [66, 75, 84, 100, 101].

4.2.4 Heuristic Dynamic Programming

Heuristic Dynamic Programming (HDP) is an iterative algorithm originally proposed by Werbos [75] which aims to solve the discrete-time HJB equation to reach the optimal control. The algorithm has the following steps [101]:

First, one starts with an initial value for the value function. For example, $V_0(x) = 0$.

Next, one solves for $u_0(x_k)$:

$$u_0(x_k) = \arg\min_{u()}(x_k^T Q x_k + u^T R u + V_0(x_{k+1})) \tag{4.12}$$

After obtaining u_0, one iterates over the value:

$$V_1(x_k) = x_k^T Q x_k + u_0^T(x_k) R u_0(x_k) + V_0(f(x_k) + g(x_k)u_0(x_k)) \tag{4.13}$$

Then, one keeps iterating between $u_i(x)$ and $V_i(x) \geq 0$:

$$\begin{aligned} u_i(x_k) &= \arg\min_{u()}(x_k^T Q x_k + u^T R u + V_i(x_{k+1})) \\ &= \arg\min_{u()}(x_k^T Q x_k + u^T R u + V_i(f(x_k) + g(x_k)u)) \\ &= \frac{1}{2} R^{-1} g(x_k^T) \frac{\partial V_i(x_{k+1})}{\partial x_{k+1}} \end{aligned} \tag{4.14}$$

$$\begin{aligned} \hat{V}_{i+1}(x_k) &= \min_{u()}(x_k^T Q x_k + u^T R u + V_i(x_{k+1})) \\ &= x_k^T Q x_k + u_i^T(x_k) R u_i(x_k) + V_i(f(x_k) + g(x_k)u_i(x_k)) \end{aligned} \tag{4.15}$$

where i is the value iteration index and k is the time index. HDP is an online forward in time algorithm.

4.2.5 Neural Network Approximation of HDP for Non-linear Systems

u_i and v_i at each iteration are hard to calculate exactly. Therefore, approximating each with neural networks (NN) have been proposed. A critic NN is used to approximately solve equation (4.15):

$$\hat{V}_i(x) = \sum_{j=1}^{L} w_{vi}^j \Phi_j(x) = W_{vi}^T \Phi(x) \tag{4.16}$$

And an action NN is used to approximately solve (4.14):

$$\hat{u}_i(x) = \sum_{j=1}^{M} w_{ui}^j \sigma_j(x) = W_{ui}^T \sigma(x) \tag{4.17}$$

where L and M are the number of hidden layers, w_{vi} and w_{ui} are the weights in the critic and action neural netwrks respectively. $\Phi(x)$ and $\sigma(x)$ are vector activation functions. W_{vi} and W_{ui} are the weight vectors at iteration i.

4.3 Hardware Implementation of Action-Critic based ADP

4.3.1 Actor-Critic Networks

The proposed hardware is based on actor-critic structure as shown in Fig. 4.1 where the actor network approximates the optimum control action and the critic network approximates the cost of this action. Both networks are provided the system state $X(k)$ as the input. The optimum action $U(k)$ generated by the actor is then provided to an external model of the system which provides the system state $X(k+1)$ (response) at the next time step $(k+1)$ based on the optimum action and the current system state. The optimum policy provided by the actor and the response from the model is evaluated by the critic which can update its weights and triggers an update for the actor.

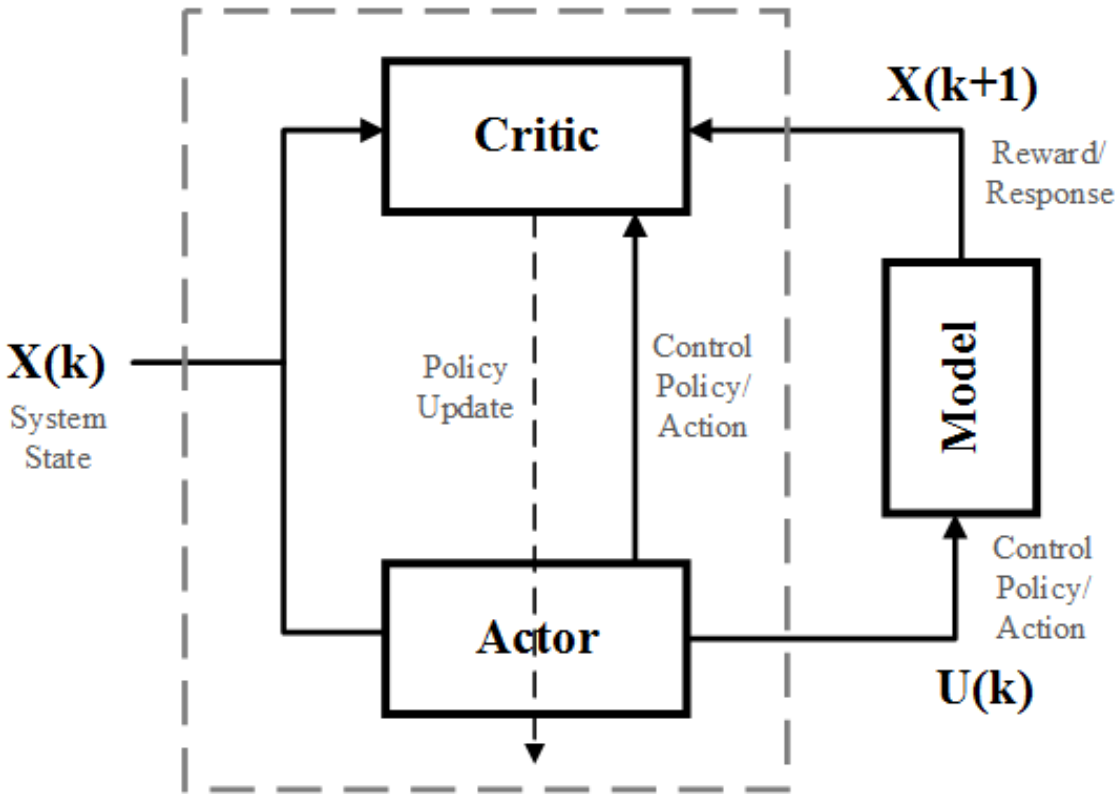

Figure 4.1: Actor-Critic Network structure.

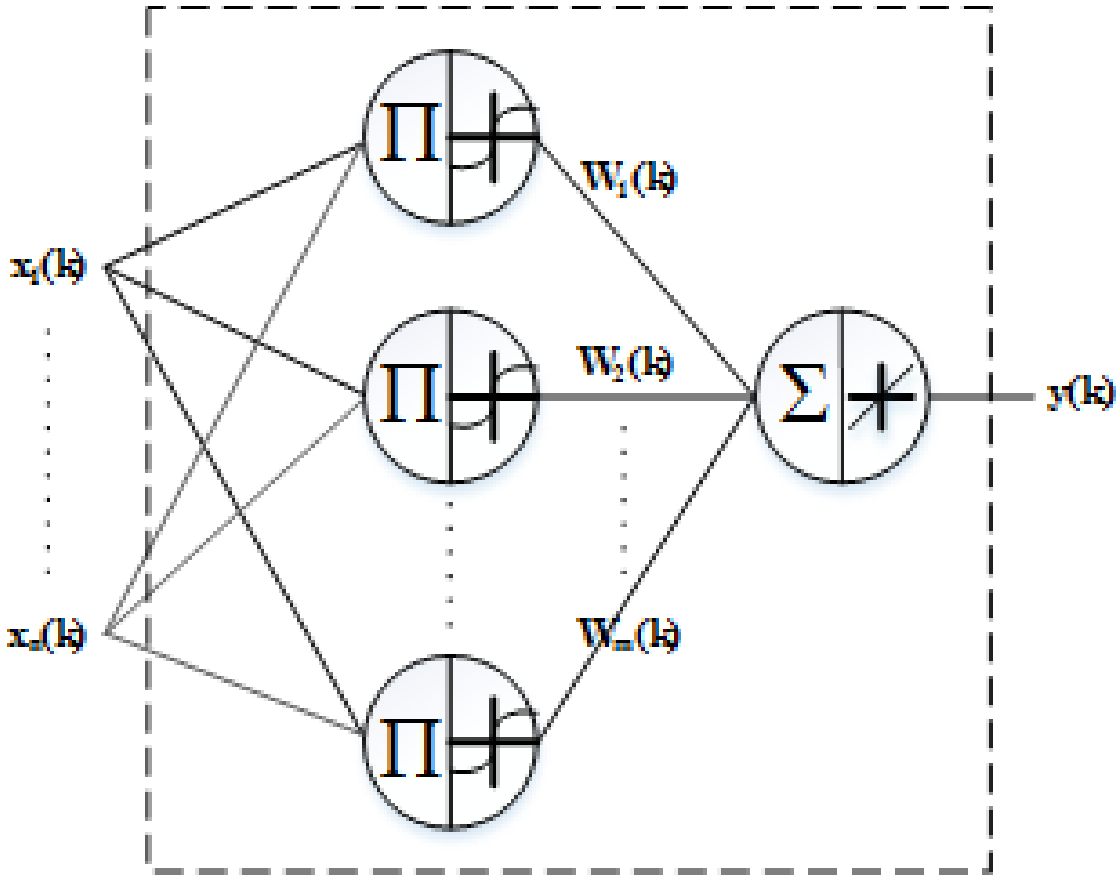

Figure 4.2: The neural network structure. $x_1(k)$, ..., $x_n(k)$ are neural network inputs, $W_1(k)$, ..., $W_m(k)$ are neural network weights, $y(k)$ is the neural network output at discrete time step k.

Actor and the critic are approximated by two neural networks with the connections as shown in Fig. 4.2. We have chosen the neurons to have polynomial activation functions as in [101, 102] due the fact that complex neuron behavior can provide better approximation of non-linear functions. The neurons are excited based on the input stimuli $x_1(k), .., x_n(k)$ and each neuron output is then multiplied with the corresponding network weight $W_1(k), ..., W_m(k)$ and summed to generate the network output $y(k)$.

4.3.2 Hardware HDP Algorithm

The version of the algorithm that is adopted by our hardware is presented in Algorithm 1.

Algorithm 1 Hardware HDP Algorithm

Require: $V_0(x_k) = 0$ and $u_0(x_k) = 0$

```
for each iteration i = 1 to n do
    for all x_k ∈ critic training set do
        estimate cost
    end for
    update critic weights and cost
    for all x_k ∈ actor training set do
        update actor weights and policy
    end for
end for
```

In this algorithm, the critic network is trained to improve the approximate cost and then the actor network is trained to improve the approximate policy at each iteration. In order to further visualize the algorithm flow, Fig. 4.3 is presented.

First the hardware is configured with the user specified parameters such as the training set size, number of state variables, linear or non-linear control mode. The hardware is then initialized the neural network weights of '0', which then yields an initial control action of '0'. The hardware, then evaluates the cost of the initial control action given the elements of the critic network training set by evaluating equation 4.13. Once a vector of estimated costs are obtained for the critic training set, the hardware moves on to solve the least squares problem to find the critic weights that best fit the estimated costs in the previous step. Once the critic weights are obtained, the hardware uses Least Mean Squares (LMS) method to train the action network to improve the policy. The training of the actor network, completes an iteration of the algorithm. If the user specified number of iterations is not reached, the hardware moves on with reestimating the costs for the critic training set using the updated policy, updates the critic weights and then updates the action network weights using

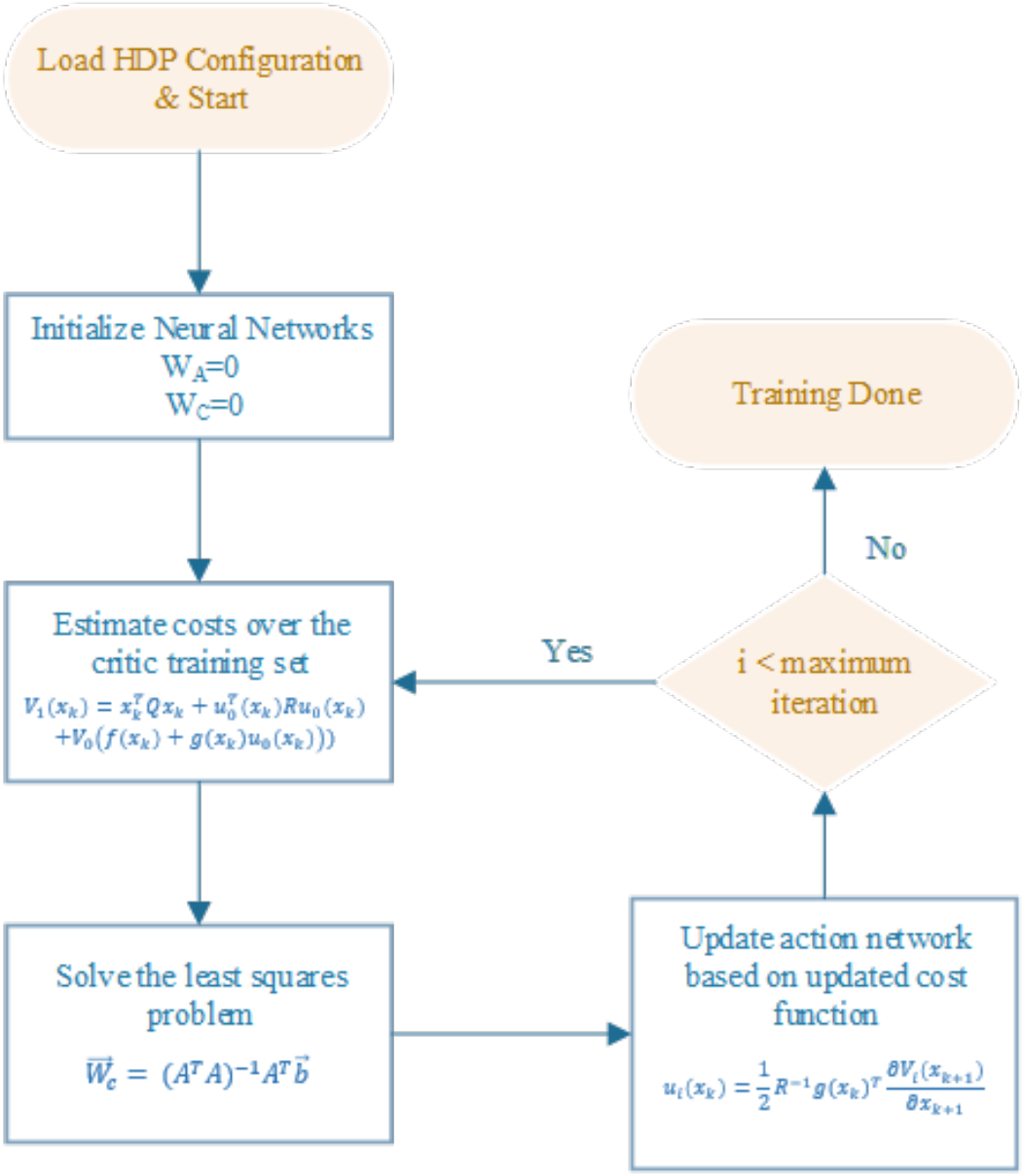

Figure 4.3: Hardware heuristic dynamic programming algorithm flow.

the improved cost function. The hardware completes training when the user specified number of iterations is reached.

4.3.3 Hardware Architecture and Specifications

The proposed hardware can currently provide adaptive control for linear and non-linear systems with up to two state variables, although the design can be extended to provide support for higher number of state variables. The proposed hardware structure is shown in Fig.4.4. The main on-chip components include the actor-critic neural network estimators, least squares fit generator,and training set generator. The model is an external component providing the dynamics of the system to be controlled which can be implemented in hardware or software. The SoC summary is presented in Table 4.2.

Training set generator produces a linearly spaced training set based on the user configurable parameters. The user can specify number of state variables, beginning value and the linear spacing for each variable as well as the training set size desired. There are two sets of parameters for the critic and the actor networks, thus their training can be customized separately.

Scan chain block is used to scan in the configuration data for the training set generator, actor-critic neural network estimators, the least squares fit generator, and the iteration controller.

Iteration controller supervises the learning and weight calculation processes. It triggers the improvement of actor and critic network weights at each iteration. The iteration number is programmed by the scan chain.

Serial Interface divides the off-chip communicated data in byte-size segments and allows a serial communication protocol to reduce the pin count of the chip. It also parallelizes incoming data to be sent to the on-chip components.

Actor-critic neural network estimators are the core of the hardware and implement

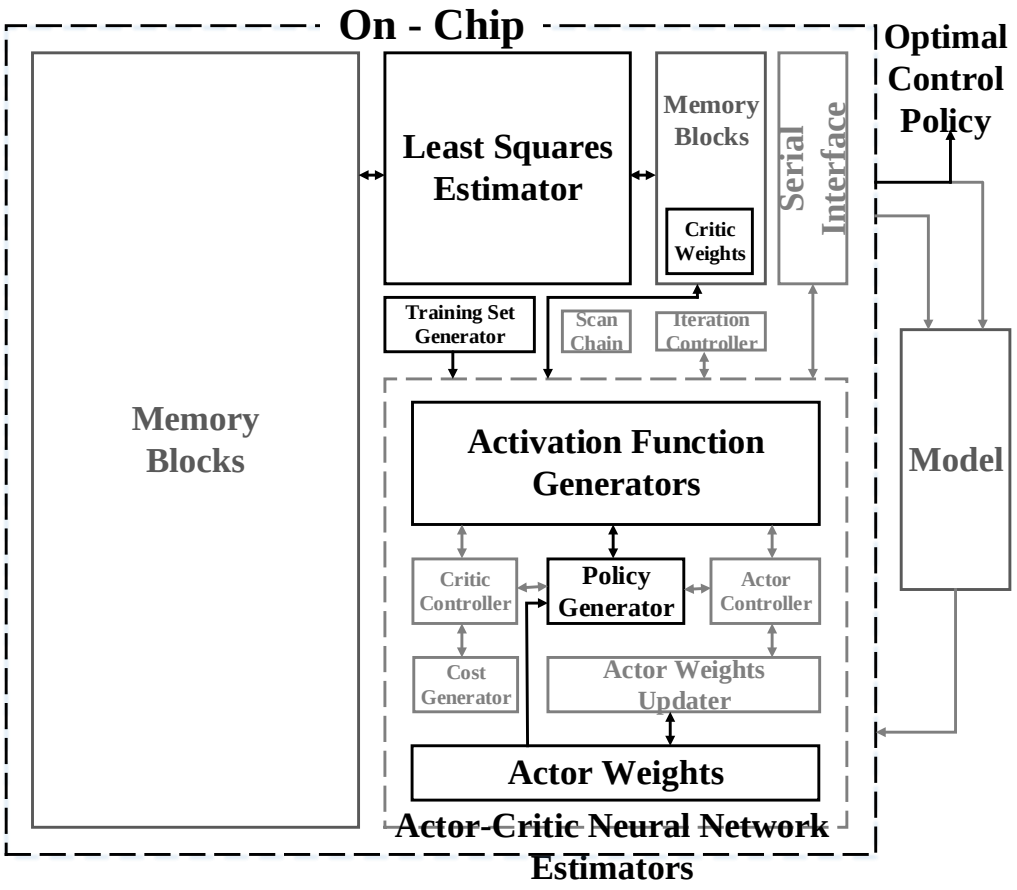

Figure 4.4: Block diagram showing the overall hardware architecture.

Table 4.1: Hardware Configurations

	Linear Mode		**Non-linear Mode**	
	Single State Variable	Two State Variables	Single State Variable	Two State Variables
Critic N.	1 Neuron	3 Neurons	3 Neurons	15 Neurons
Actor N.	1 Neuron	2 Neurons	3 Neurons	12 Neurons

the actor and critic neural networks. The neural networks are programmable in the sense that the user can configure the number of neurons the networks use depending on if the number of state variables of the system to be controlled and its linear or non-linear classification. The possible configurations are listed in Table 4.1. The variable number of neurons are achieved by time multiplexing the neuron operation and reusing the common hardware resources between neurons. 24-bit fixed point representation for weights and neuron outputs was chosen to reduce hardware complexity while still maintaining accuracy compared to the software implementation. This block also includes on-chip memory blocks to store the neural network weights and neuron outputs. The total memory storage in this block is 252 Bytes.

Actor and Critic Controllers oversee the training and operation of the networks. They are also responsible for the managing the time multiplexing of the neuron operation.

Activation function generators generate and store the different activation outputs resulted from the neuron operation. The work presented in [95] recommends the use of look-up tables (LUTs) for fast operation. However, the use of LUTs proved to be too expensive in terms of chip area so arithmetic blocks have been used instead and the common blocks between neurons have been shared.

Policy generator block performs the main weighted sum operation required by the actor network. The block computes the optimal policy based on the actor network weights and the actor neuron outputs generated by the activation function generator.

Cost generator block is essentially the critic network version of the policy generator. It performs the main weighted sum operation required by the critic network. It computes the cost based on the critic network weights and the critic neuron outputs generated by the activation function generator.

Actor weights updater tunes the actor weights based on the training set using the least mean squares method [101, 103] and stores the resulting weights in the *Actor weights* block. Tuning the critic network weights, on the other hand, is not trivial as will be explained in the following subsection.

4.3.4 Least Squares Regression Calculation

In order to generate the weights for the critic neural network, least squares regression (LSR) can be used to approximate the vector of weights that correlate the neuron outputs with the associated costs of a set of possible states [101, 85].

LSR accuracy can be increased by increasing the number of sample states, which in turn increases the number of neuron output vectors and the number of associated costs that need to be stored. However, in order to ensure a solution, the minimum requirement is to use as many samples as the number of neurons in the network to generate a square matrix. One can understand this requirement by thinking about a linear system with n unknowns. In order to solve for this system, at least n different equations with n unknowns are required. The critic weights can be estimated by solving the least squares problem:

$$\vec{W_c} = (A^T A)^{-1} A^T \vec{b} \tag{4.18}$$

where $\vec{W_c}$ is the vector of critic weights, A is the square matrix generated by the sampled neuron outputs, $\vec{b}$ is the vector of sampled costs. The hardware is capable of generating the approximate solution to the LSR problem on-chip. In order to minimize the memory requirements and reduce the computation time, the minimum

number of samples that suffice the solution criteria are used.

Algorithm 2 Modified Gaussian Elimination Algorithm

Require: matrix $A \in R^n x R^n$

```
1: for each column i = 1 to n do
2:    for each row k = i to n do
3:       if (a_{k,i} ≠ 0) ∧ (a_{k,i} ≠ 1) then
4:          divide row k by a_{k,i}
5:       end if
6:    end for
7:    for each row k = i to n do
8:       if (a_{k,i} = 1) then
9:          if k = i then
10:            break
11:         else
12:            add row k to i
13:            break
14:         end if
15:      end if
16:   end for
17:   for each row k = i + 1 to n do
18:      if (a_{k,i} = 1) then
19:         subtract row i from row k
20:      end if
21:   end for
22: end for
23: for each column i = 2 to n do
24:    for each row k = 1 to i − 1 do
25:       if (a_{k,i} ≠ 0) then
26:          subtract a_{k,i} times row i from row k
27:       end if
28:    end for
29: end for
```

Least squares estimator block uses the square matrix and the vector of associated costs generated by the critic network and solves the least squares problem of equation (4.18) using various matrix operations. Basic matrix operations such as matrix multiplication are trivial to implement in hardware. However inverting a matrix involves calculating computationally and memory-wise expensive determinants. In order to save hardware resources, the Gaussian elimination algorithm was modified for the

proposed hardware to solve the LSR problem on chip as shown in Algorithm 2. The algorithm aims to make the Gaussian elimination which is usually implemented in software more hardware friendly by only using arithmetic operations.

Least squares fit generator block encompasses least squares estimator block as well as the required memory banks to store the intermediate results of the matrix operations. The total amount of memory required is about 5.56 KBytes As will be shown in the next section, the hardware is capable of accurately approximating the cost function using minimum number of samples to solve the least squares problem on-chip.

4.4 Simulation Results

4.4.1 Simulation Setting

In order to be able to judge the proposed hardware performance, the HDP algorithm was also implemented in software. The software implementation was performed using MATLAB.

For both implementations the actor network is trained using a training set of 126x126 (15876) elements linearly spaced in the range (-1,1). The critic network is trained using a training set of 5x3 (15) elements linearly spaced within the range (-1,1).

The software implementation was run on a 64-bit Intel Xeon processor with 1.2 GHz clock frequency. The total execution time for 100 iterations of the algorithm is measured to be 1 hour 29 minutes 39 seconds. The hardware implementation has clock frequency of 10 MHz, and the run-time is 22.63 seconds.

The hardware power estimation is obtained using SPICE simulations and the power consumption is 2.1 mWatts for 10 MHz clock.

4.4.2 Error Quantization

To quantize the weight mismatch between the hardware and the software, lumped absolute error was defined as the ratio of the sum of the absolute distances between the observed and the expected weights over the sum of the absolute values of the expected weights:

$$LumpedAbsoluteError = \frac{\sum_{i}^{N} |W_H - W_T|}{\sum_{i}^{N} |W_T|} \tag{4.19}$$

where W_H is the hardware (observed) weights, W_T is the software (expected) weights, N is the number of neurons, i is the neuron index. The average lumped absolute error is defined as the lumped absolute error divided by the number of neurons:

$$AverageLumpedAbsoluteError = \frac{\sum_{i}^{N} |W_H - W_T|}{N \sum_{i}^{N} |W_T|} \tag{4.20}$$

Finally, the average lumped absolute error is multiplied by 100 to obtain the error percentages:

$$AverageAbsoluteErrorPercentage = \frac{\sum_{i}^{N} |W_H - W_T|}{N \sum_{i}^{N} |W_T|} x100 \tag{4.21}$$

4.4.3 Non-linear System Example

The following non-linear system is used to compare the hardware and software controllers. The system is defined as:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} = f(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k) + g(x_k)u_k \tag{4.22}$$

where

$$f(x_k) = \begin{bmatrix} 0.25(x_1 e^{x_1} - x_2^4) \\ 0.49 x_1^2 x_2 \end{bmatrix} \tag{4.23}$$

and

$$g(x_k) = \begin{bmatrix} 0 \\ -0.375 \end{bmatrix} \tag{4.24}$$

4.4.4 Actor and Critic Network Weights during Training

The non-linear system of equation (4.22) is used as the model for both the software implementation of HDP and our proposed SoC. The change in actor network weights over 100 iterations are shown in Fig. 4.5 for both the hardware and software implementations. For the actor network, the weights fall within a similar range for both the hardware and software, however they do not match perfectly. The average absolute error percentage calculated using equation (4.21) is 5.94%.

The main source of differences between the hardware and software weights is the limited bits available to the hardware, which directly affects the response of the neural network causing some neurons to be over-trained and most to be under-trained compared to their software counterparts. Even though the hardware weights start to diverge from the software weights, it is still possible to approximate the main output of the network, the optimum policy, such that the hardware output follows the software output as will be shown in the following subsection.

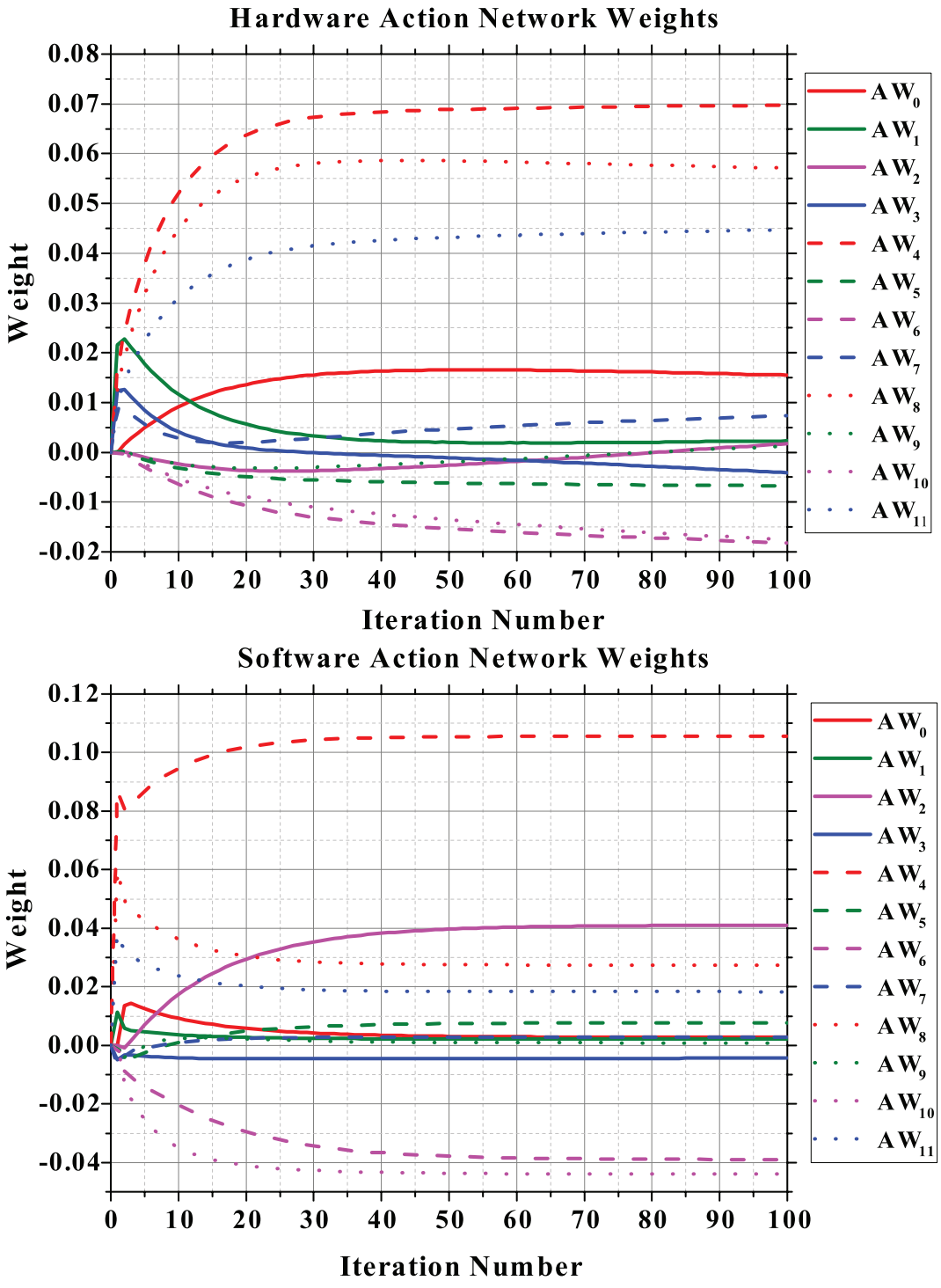

Figure 4.5: Hardware vs software actor network weights over iterations.

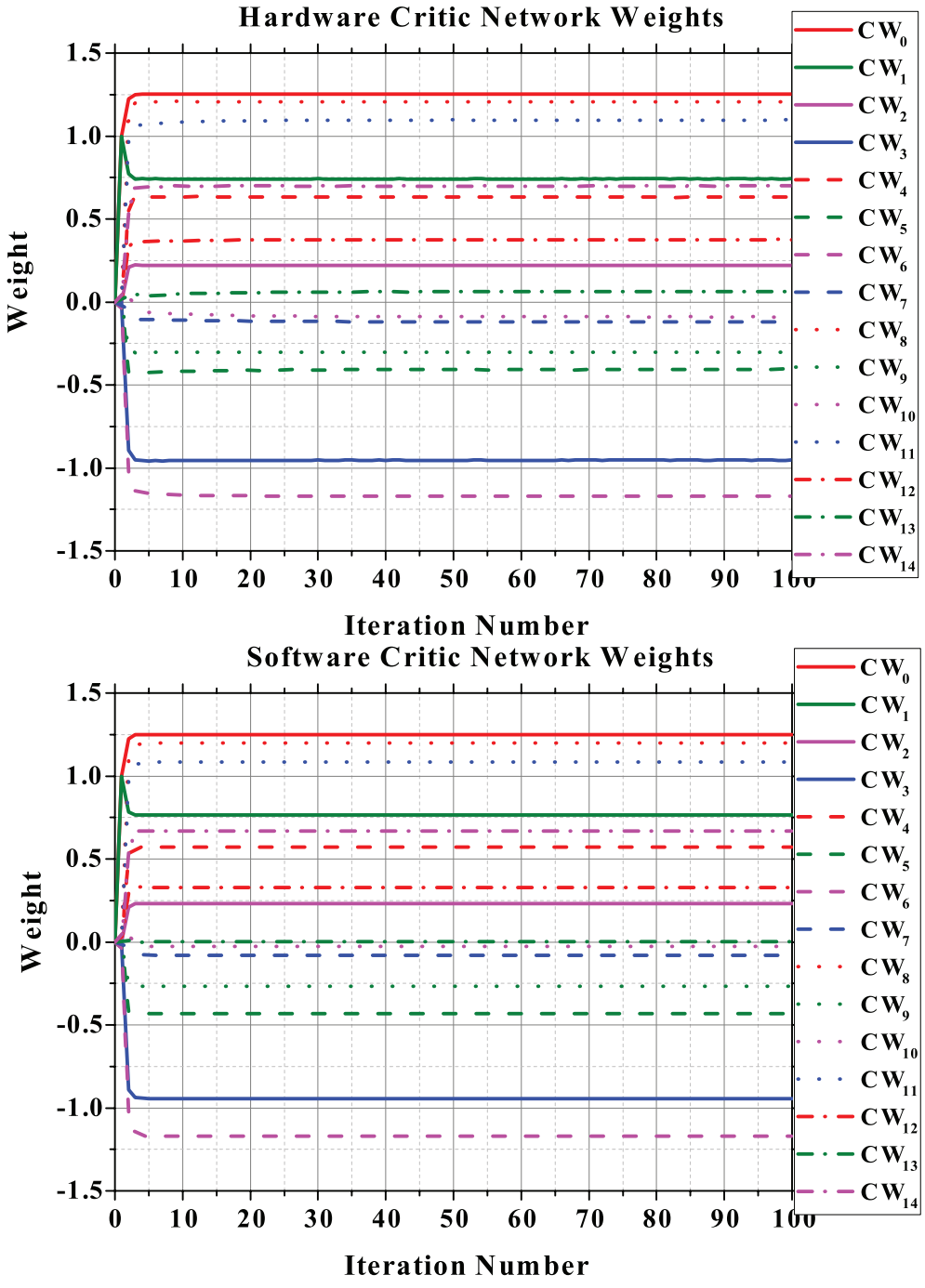

Figure 4.6: Hardware vs software critic network weights over iterations.

For both the hardware and the software implementations, the neural network weights are initialized as zero. In hardware case, the weights start to converge around 50th iteration. For software case some weights start converging as early as 30th iteration, whereas few of them start converging around 50th iteration.

Fig. 4.6 shows the change in critic network weights for both the hardware and software implementations. The critic network weights for the hardware follow the software implementation very closely. The average absolute error percentage is 0.4%. The bit limitation as well as divergence of the actor weights also cause the hardware weights to diverge from the software weights. However, the diversion is smaller than the case of the actor network. The critic weights start converging earlier around 4th iteration.

4.4.5 Optimum Policy and Associated Cost during Training

The optimum policy for the software implementation starts converging early on around 20th iteration with an initial sharp training response within the first few iterations as shown in the first row of Fig. 4.7. However, a slow transition still takes place until around 50th iteration. The hardware policy undergoes a much slower but more uniform transition, starting to converge around 50th iteration. The limited number of bits available to the hardware, limits the training of the actor network causing the hardware policy to exhibit a much slower initial response whereas the software policy displays a fast initial response. The difference between the software and the hardware optimum policies is 10.2% at 100th iteration.

The associated cost shows a much steeper response for both the hardware and the software for the first few iterations as shown in the second row of Fig. 4.7. The steep response is expected for the software as the same trend is observed in the case of the policies. For the hardware, more bits are available for the generation of an accurate least squares fit estimation as shown in Table 4.2; hence, the hardware response is

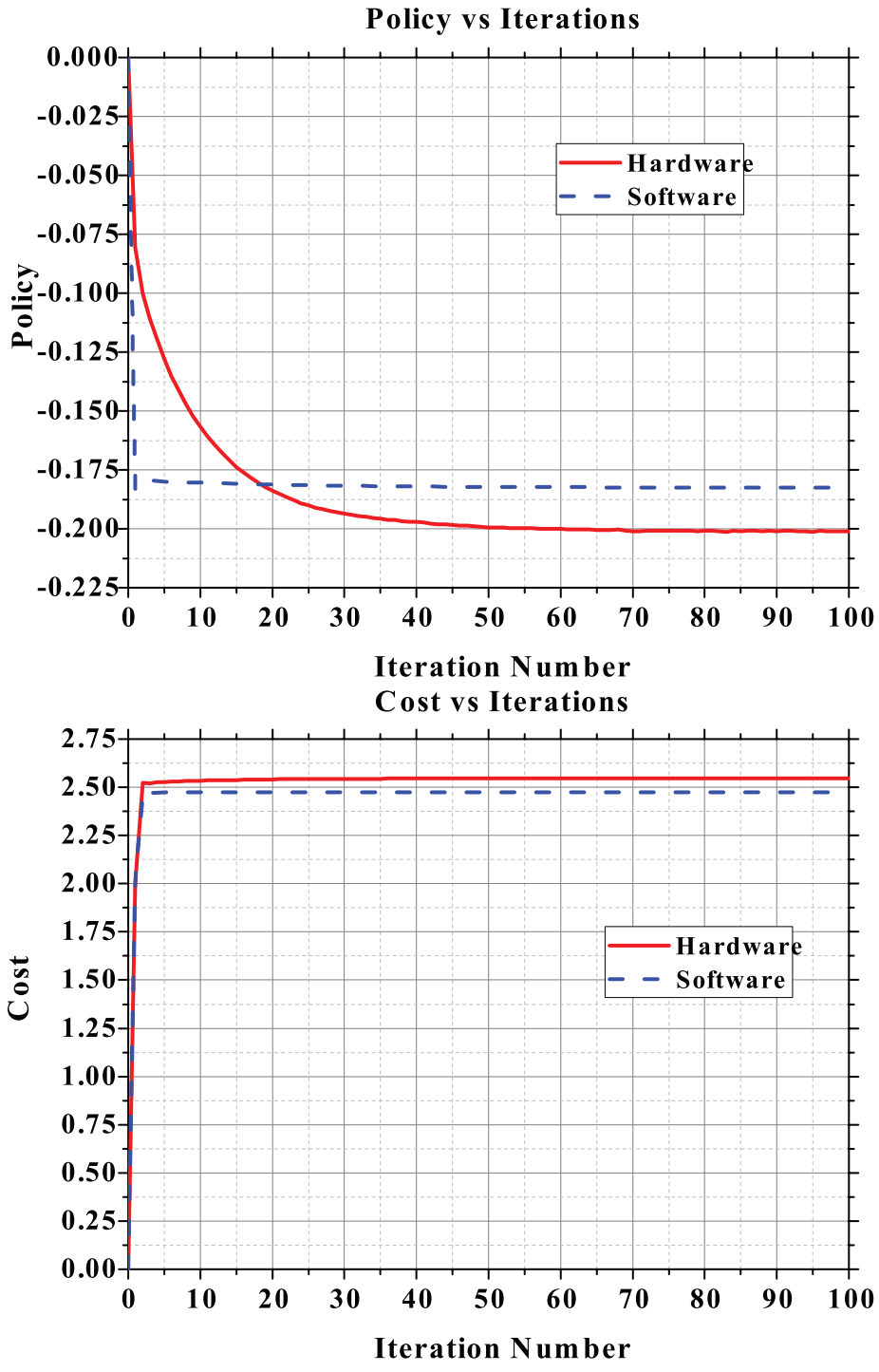

Figure 4.7: Hardware vs software optimum policy and its cost over iterations.

also much steeper. The costs start to converge around 10 iterations although some slight changes are observed in the hardware cost due to the slow settling of the actor network. The difference between the software and the hardware costs is 2.96% at 100th iteration.

4.4.6 Discrete Time System Response

Once the training phase is over, the system response as well as the controller outputs are compared for both the hardware and software over six time steps. The system defined in (4.22) is given initial condition [1,-1] for the state variables x_1 and x_2 respectively. Even though any admissible initial condition within the training range can be picked, we picked this condition as an example and also because '1' and '-1' form the extremes of the hardware training range.

The initial states are shown in Fig. 4.8 and the initial hardware and software control policies are shown in Fig. 4.10. At time step **0**, the initial states [x_1,x_2] are [1,-1], and the hardware and software policies are -0.2011 and -0.1825 respectively. The difference in the policies are shown in Fig. 4.11 and the resulting state errors are also plotted on Fig. 4.9. Since the initial conditions are the same for both the hardware and software controlled systems, the initial state errors are '0' at time step **0**. The error on state variable x_2 maximizes on time step **1**, which is the result of the different initial policies at time step **0**. However, in both cases the system reaches the target state [0, 0] around the same discrete time step. The error between the policies and the state variables become approximately '0' by the discrete time step **5**.

4.4.7 The Proposed Hardware Layout

The results indicate that the proposed hardware can perform very similar to the purely software based controller in controlling a discrete non-linear system, even though the hardware has reduced precision. The hardware can be trained at a much

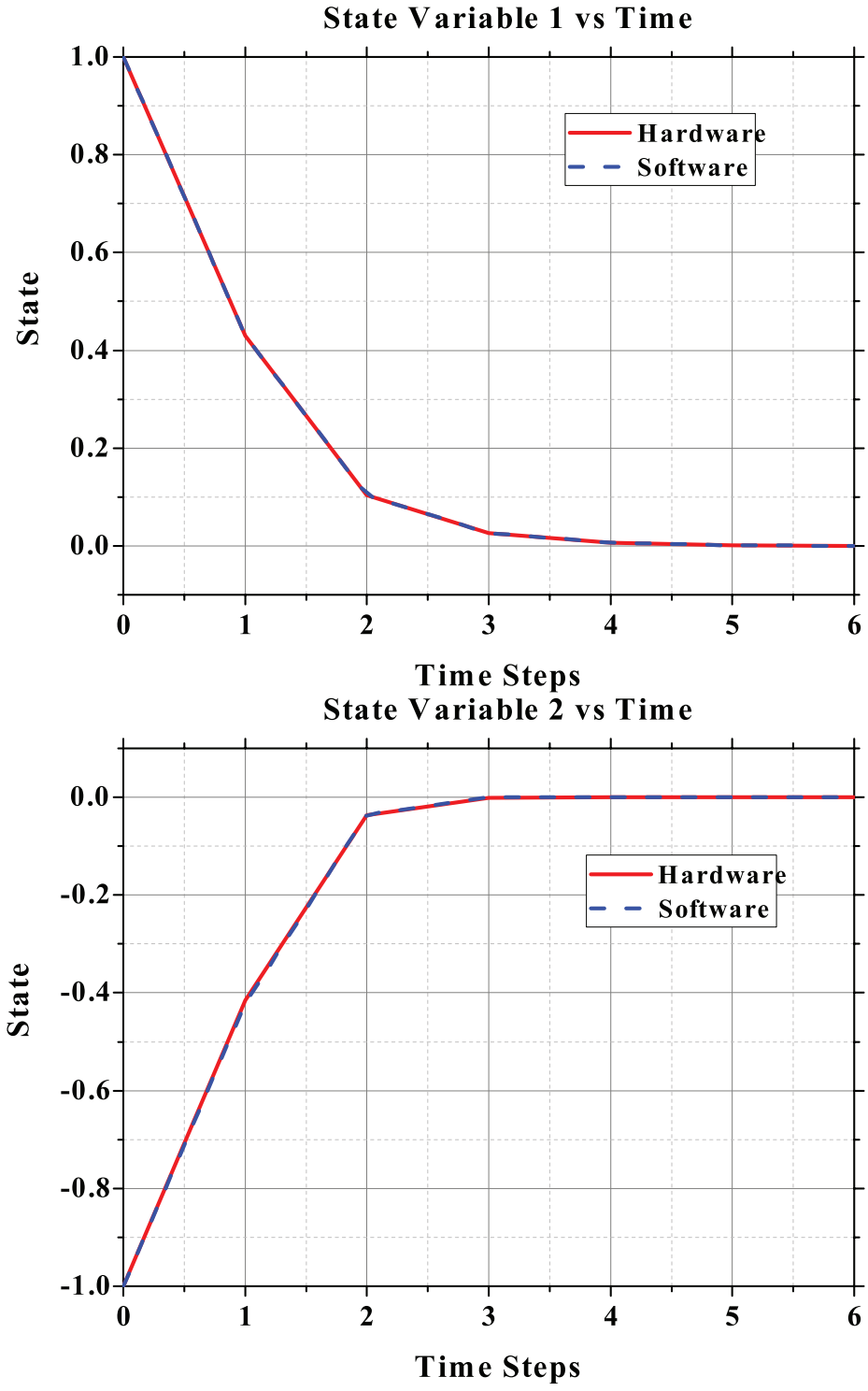

Figure 4.8: Discrete time system response with hardware and software policies over time. The change in system state is plotted for both the hardware and software policies.

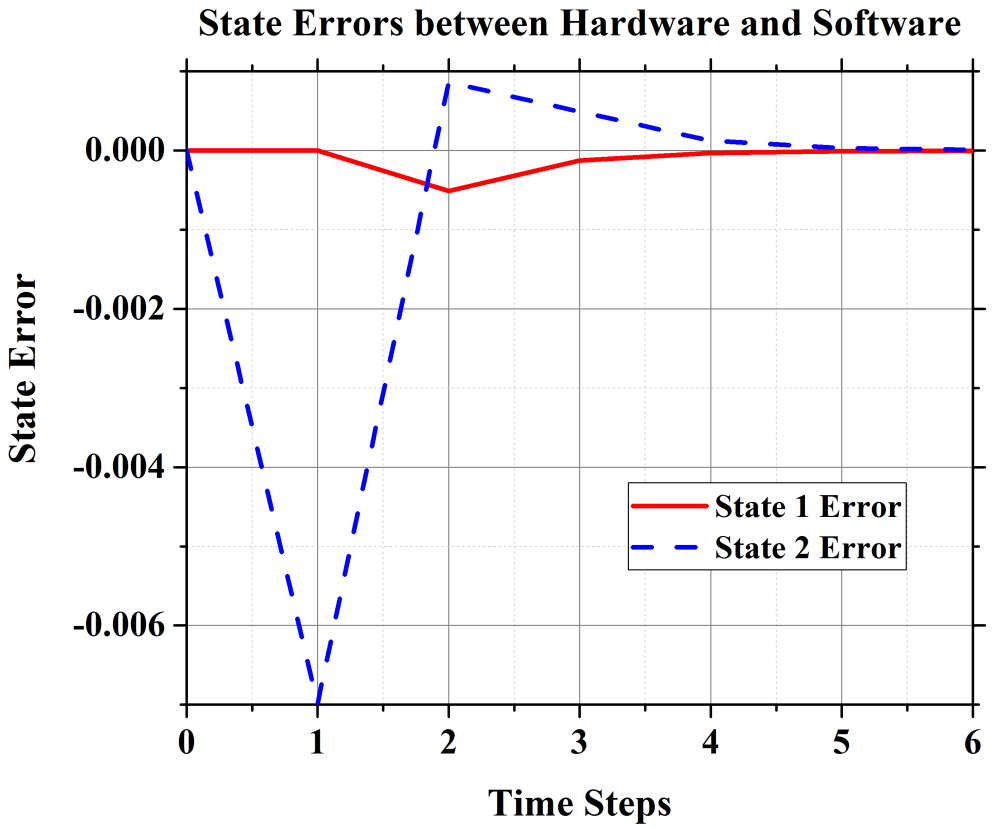

Figure 4.9: The error in system states for the hardware policy compared to the software policy.

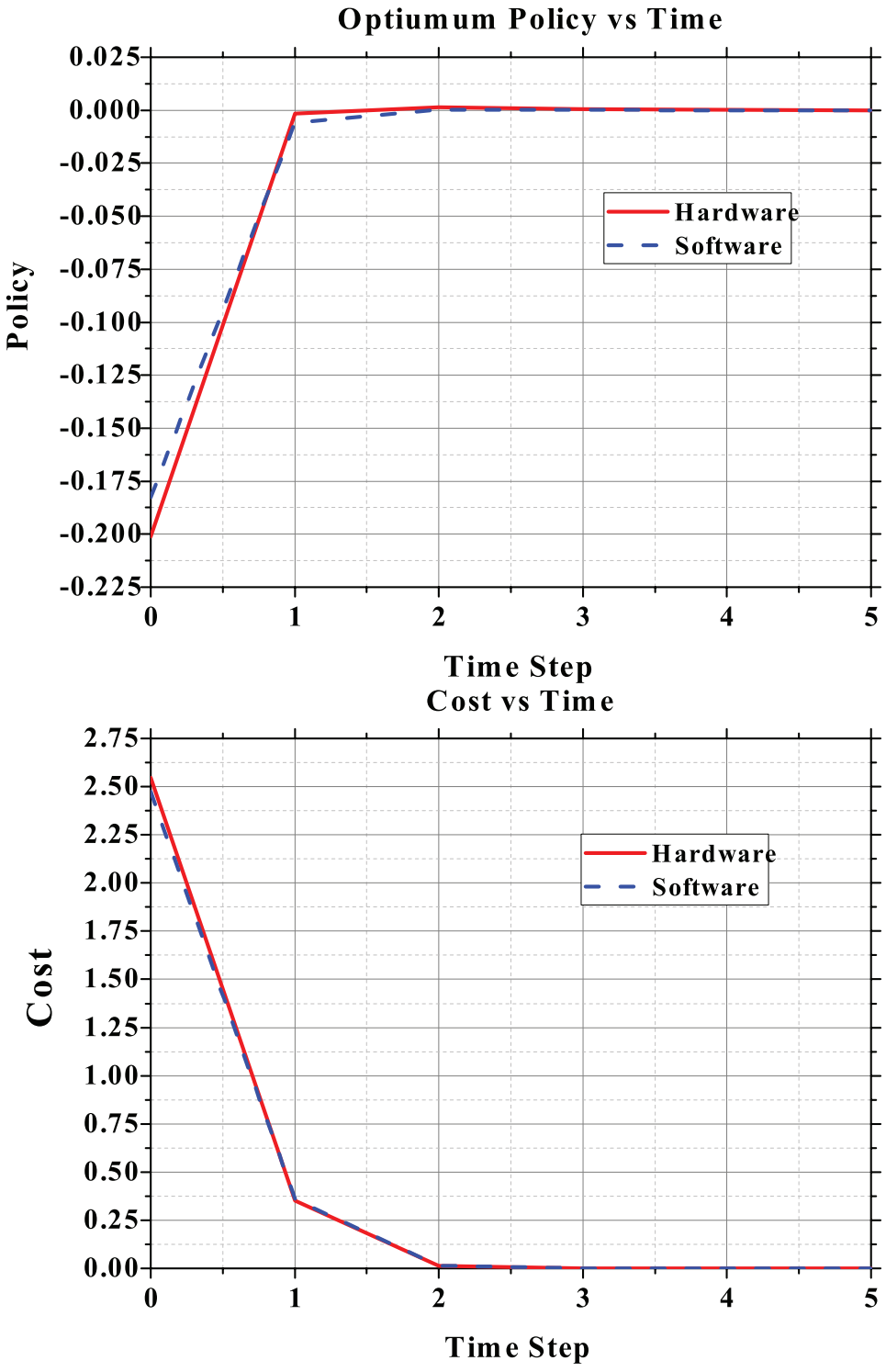

Figure 4.10: The hardware and software policies over time.

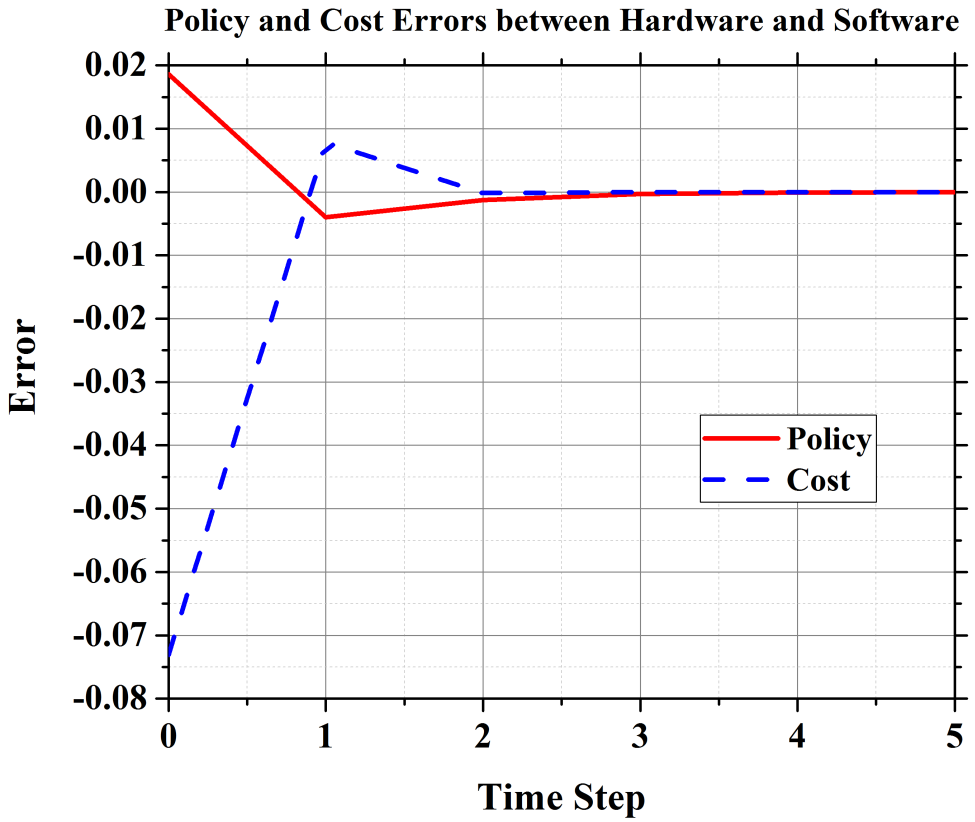

Figure 4.11: The differences between the hardware policy and cost compared to the software counterparts.

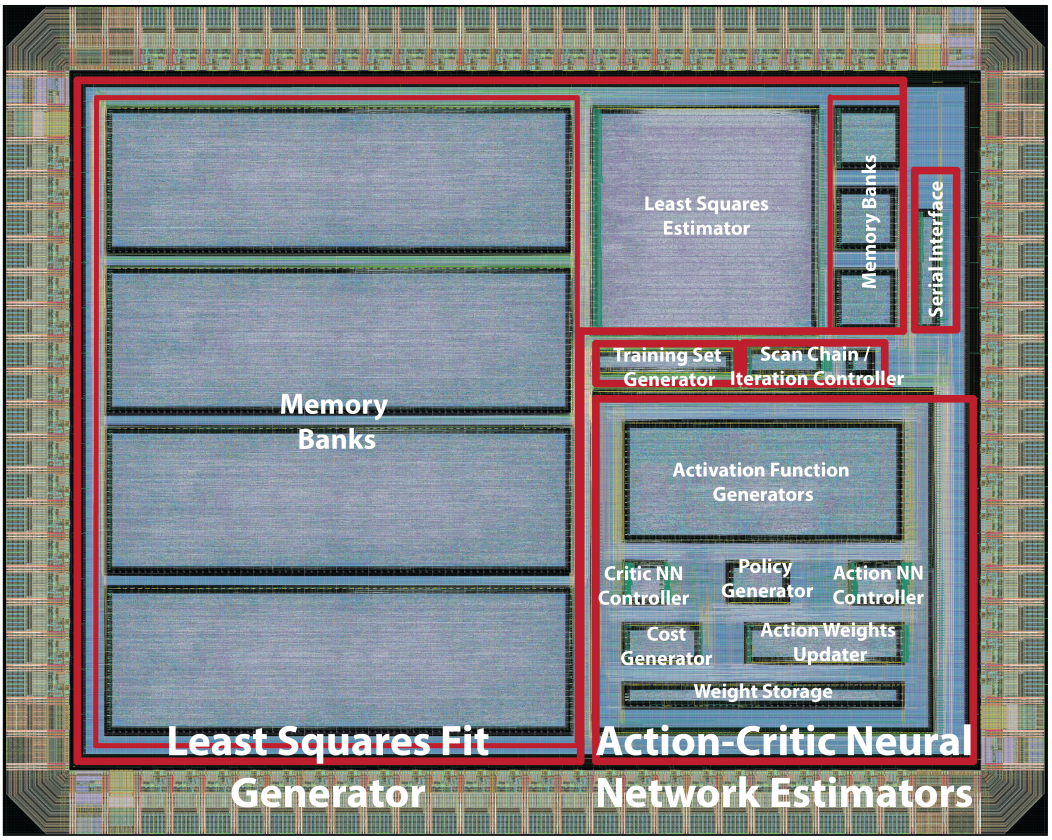

Figure 4.12: Proposed heuristic dynamic programming (HDP) chip layout. The SoC dimensions are $1950\mu m x 1550\mu m$ in 65nm CMOS LP technology including the pads. The core dimensions are $1700\mu m x 1300\mu m$.

faster rate than the software allowing the application of heuristic dynamic programming to real-time learning tasks. At 10 MHz the hardware can be trained about 237 times faster than the software run on a 1.2 GHz Intel Xeon processor.

The final chip layout is shown in Fig. 4.12. Individual components are annotated. The chip is designed using TSMC's 65nm LP CMOS technology using high-Vt transistors.

Table 4.2: HDP SoC Summary

Technology	65 nm LP CMOS
Die Size	$1950\mu m$ x $1550\mu m$
Supply Voltage	1.2 V
Operation Frequency	10 MHz
Training Set Range	(-1,1)
Neural Network Resolution	2^{-13}
Neural Network Word Length	24-bits
Least Squares Fit Block Word Length	96-bits

4.5 Conclusion

In this chapter a hardware capable of implementing heuristic dynamic programming on chip for general non-linear systems was presented. How the least squares problem using minimum number of samples can be accurately solved on chip was shown, thus reducing on-chip memory requirements. Finally, the simulation results for the hardware implemented in 65 nm LP CMOS technology using high-Vt transistors were presented. The hardware policy shows 10.2% and the cost shows 2.96% divergence compared to the software counterparts while the hardware can complete the training 237 times faster than the software while consuming mere 2.1 mW of power.

CHAPTER V

Conclusions

In this dissertation bio-inspired hardware structures aimed at enhancing/replacing conventional VLSI systems, and their applications have been presented.

In the second chapter of this dissertation, a crossbar memory structure capable of storing multi-bit information per cell is presented. A reduced constraint read-monitored-write scheme was also presented to accurately program the resistive states of the memory cells. The simulation results indicate that the architecture programs the cells to the detection thresholds with very tight state distributions. An analytical model for state derivations was also presented to guide the designers on how to select the component parameters. The proposed hardware allows for reliable memory state detection by generating detection margins via scaling of read voltages and series resistance values.

In the third chapter of this dissertation, an analog grid-based structure consisting of variable resistance connections for programmability, and resonant tunneling diodes for signal detection and latching was presented. The flexible fabric of this structure, provides the ability to program it for various image processing tasks. A programming scheme for the connections was also presented. The architecture was characterized analytically for edge detection and line detection applications. Extensive simulations were performed to verify the different functions of the structure and demonstrate its

operations at extremely high speeds.

In the fourth chapter of this dissertation, a hardware implementation for heuristic dynamic programming was presented. The hardware is capable of providing neuro-control for general non-linear systems. A hardware solution to the least squares problem while using minimum number of samples, thus reducing the on-chip memory requirements was also presented. The hardware was designed using 65 nm LP CMOS technology high-Vt transistors. The hardware policy is 10.2% and the hardware cost is 2.96% different compared to their software results using the same training set and model. The hardware on the other hand can complete the training 237 times faster than the software while consuming 2.1 mW of power according to SPICE simulation results.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[2] Y. Yilmaz and P. Mazumder, "Threshold Read Method for Multi-bit Memristive Crossbar Memory," in *2011 International Symposium on Electronic System Design*. IEEE, 2011, pp. 217–222.

[3] I. Ebong, D. Deshpande, Y. Yilmaz, and P. Mazumder, "Multi-purpose neuro-architecture with memristors," in *2011 11th IEEE International Conference on Nanotechnology*. IEEE, 2011, pp. 431–435.

[4] a. Gelencsér, T. Prodromakis, C. Toumazou, and T. Roska, "Biomimetic model of the outer plexiform layer by incorporating memristive devices," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 85, no. 4, pp. 1–10, 2012.

[5] K. H. Kim, S. Hyun Jo, S. Gaba, and W. Lu, "Nanoscale resistive memory with intrinsic diode characteristics and long endurance," *Applied Physics Letters*, vol. 96, no. 5, pp. 2013–2016, 2010.

[6] L. O. Chua, "Memristor - The missing circuit element," *Circuit Theory, IEEE Transactions on*, vol. 18, no. 5, pp. 507–519, 1971.

[7] Y. N. Joglekar and S. J. Wolf, "The elusive memristor: properties of basic electrical circuits," *Eur. J. Phys.*, vol. 30, no. 4, pp. 661–675, 2009.

[8] Z. Biolek, D. Biolek, and V. Biolková, "SPICE model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, no. 2, pp. 210–214, 2009.

[9] Y. V. Pershin and M. Di Ventra, "SPICE model of memristive devices with threshold," *Radioengineering*, vol. 22, no. 2, pp. 485–489, 2013.

[10] G. H. Liu and G. Y. Wang, "An improved SPICE macromodel of memristor for general applications," *Applied Mechanics and Materials*, vol. 303-306, pp. 1854–1858, feb 2013.

[11] W. Fei, H. Yu, W. Zhang, and K. S. Yeo, "Design Exploration of Hybrid CMOS and Memristor Circuit by New Modied Nodal Analysis," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 6, pp. 1012–1025, 2012.

[12] "NAND Flash Revenue to Grow by 13% Y/Y to $28 Billion in 2014," 2013. [Online]. Available: http://www.storagenewsletter.com/rubriques/market-reportsresearch/nand-flash-revenue-to-grow-by-13-yy-to-28-billion-in-2014-dramexchange/

[13] "$1.2 Billion in Revenue by 2015 for All Flash Arrays," 2013. [Online]. Available: http://www.storagenewsletter.com/rubriques/market-reportsresearch/all-flash-arrays-idc-2015/

[14] K. Takeuchi, "Scaling challenges of NAND flash memory and hybrid memory system with storage class memory & NAND flash memory," in *Custom Integrated Circuits Conference (CICC), 2013 IEEE*, 2013, pp. 22–25.

[15] A. Chung, J. Deen, J.-S. Lee, and M. Meyyappan, "Nanoscale memory devices," *Nanotechnology*, vol. 21, no. 41, p. 412001, oct 2010.

[16] R. Scheuerlein, W. Gallagher, S. Parkin, A. Lee, S. Ray, R. Robertazzi, and W. Reohr, "A 10ns read and write non-volatile memory array using a magnetic tunnel junction and FET switch in each cell," in *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, 2000, pp. 128–129.

[17] R. Takemura, T. Kawahara, K. Miura, H. Yamamoto, J. Hayakawa, N. Matsuzaki, K. Ono, M. Yamanouchi, K. Ito, H. Takahashi, S. Ikeda, H. Hasegawa, H. Matsuoka, and H. Ohno, "A 32-Mb SPRAM with 2T1R memory cell, localized bi-directional write driver and '1'/'0' dual-array equalized reference scheme," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 4, pp. 869–879, 2010.

[18] F. Bedeschi, R. Bez, C. Boffino, E. Bonizzoni, E. Buda, G. Casagrande, L. Costa, M. Ferraro, R. Gastaldi, O. Khouri, F. Ottogalli, F. Pellizzer, A. Pirovano, C. Resta, G. Torelli, and M. Tosi, "4-Mb MOSFET-selected phase-change memory experimental chip," in *Solid-State Circuits Conference, 2004. ESSCIRC 2004. Proceeding of the 30th European*, 2004, pp. 207–210.

[19] X. Wu, D. Cha, M. Bosman, N. Raghavan, D. B. Migas, V. E. Borisenko, X. X. Zhang, K. Li, and K. L. Pey, "Intrinsic nanofilamentation in resistive switching," *Journal of Applied Physics*, vol. 113, no. 11, p. 114503, 2013.

[20] G. Csaba and P. Lugli, "Read-out design rules for molecular crossbar architectures," *Nanotechnology, IEEE Transactions on*, vol. 8, no. 3, pp. 369–374, 2009.

[21] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (imply) logic: Design principles and methodologies," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 10, pp. 2054–2066, 2014.

[22] J. Rajendran, H. Manem, R. Karri, and G. S. Rose, "Memristor based programmable threshold logic array," *Proceedings of the 2010 IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2010*, pp. 5–10, 2010.

[23] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[24] M. Itoh and L. O. Chua, "Memristor cellular automata and memristor discrete-time cellular neural networks," *International Journal of Bifurcation and Chaos*, vol. 19, no. 11, pp. 3605–3656, 2009.

[25] H. Kim, M. P. Sah, C. Yang, T. Roska, and L. O. Chua, "Neural synaptic weighting with a pulse-based memristor circuit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 1, pp. 148–158, 2012.

[26] Y. Yilmaz and P. Mazumder, "Image processing by a programmable grid compromising quantum-dots and memristor," *Nanotechnology, IEEE Transactions on*, vol. 12, no. 6, pp. 879–887, 2013.

[27] S. Shin, K. Kim, and S. M. Kang, "Memristor applications for programmable analog ICs," *IEEE Transactions on Nanotechnology*, vol. 10, no. 2, pp. 266–274, 2011.

[28] Y. V. Pershin and M. Di Ventra, "Practical approach to programmable analog circuits with memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 8, pp. 1857–1864, 2010.

[29] J. Cong and B. Xiao, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration," *Proceedings of the 2011 IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2011*, pp. 1–8, 2011.

[30] W. Wei, T. T. Jing, and B. Butcher, "FPGA based on integration of memristors and CMOS devices," *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 1963–1966, 2010.

[31] M. S. Qureshi, M. Pickett, F. Miao, and J. P. Strachan, "CMOS interface circuits for reading and writing memristor crossbar array," *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pp. 2954–2957, may 2011.

[32] C. E. Merkel, N. Nagpal, S. Mandalapu, and D. Kudithipudi, "Reconfigurable N-level memristor memory design," *The 2011 International Joint Conference on Neural Networks*, pp. 3042–3048, jul 2011.

[33] H. Manem and G. S. Rose, "A read-monitored write circuit for 1T1M multi-level memristor memories," *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pp. 2938–2941, may 2011.

[34] H. Manem, G. S. Rose, X. He, and W. Wang, "Design considerations for variation tolerant multilevel CMOS/Nano memristor memory," *Proceedings of the 20th symposium on Great lakes symposium on VLSI - GLSVLSI '10*, p. 287, 2010.

[35] H. Kim, M. P. Sah, C. Yang, and L. O. Chua, "Memristor-based multilevel memory," *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*, vol. 1, no. 5, pp. 1–6, feb 2010.

[36] R. Berdan, T. Prodromakis, and C. Toumazou, "High precision analogue memristor state tuning," *Electronics Letters*, vol. 48, no. 18, pp. 1105–1107, aug 2012.

[37] W. Yi, F. Perner, M. S. Qureshi, H. Abdalla, M. D. Pickett, J. J. Yang, M.-X. M. Zhang, G. Medeiros-Ribeiro, and R. S. Williams, "Feedback write scheme for memristive switching devices," *Applied Physics A*, vol. 102, no. 4, pp. 973–982, jan 2011.

[38] N. P. Jouppi, "Design implications of memristor-based RRAM cross-point structures," *2011 Design, Automation & Test in Europe*, pp. 1–6, mar 2011.

[39] J. J. Huang, G. L. Lin, C. W. Kuo, W. C. Chang, and T. H. Hou, "Room-temperature TiOx oxide diode for 1D1R resistance-switching memory," *2009 International Semiconductor Device Research Symposium, ISDRS '09*, pp. 9–10, 2009.

[40] S. H. Jo and W. Lu, "CMOS compatible nanoscale nonvolatile resistance switching memory." *Nano letters*, vol. 8, no. 2, pp. 392–7, mar 2008.

[41] Y. Ho, G. M. Huang, and P. Li, "Dynamical properties and design analysis for nonvolatile memristor memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 724–736, 2011.

[42] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176–183, 2013.

[43] A. Shadaram, S. Mirzakuchaki, and F. Zakerian, "A one-memristor cell implementation of a non-volatile memory system," *Canadian Journal on Electrical and Electronics Engineering*, vol. 2, no. 7, pp. 325–331, 2011.

[44] P. O. Vontobel, W. Robinett, P. J. Kuekes, D. R. Stewart, J. Straznicky, and R. Stanley Williams, "Writing to and reading from a nano-scale crossbar memory based on memristors." *Nanotechnology*, vol. 20, no. 42, p. 425204, oct 2009.

[45] H. Manem, J. Rajendran, and G. S. Rose, "Design considerations for multilevel CMOS/nano memristive memory," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 8, no. 1, pp. 1–22, feb 2012.

[46] S. H. Jo, T. Kumar, M. Asnaashari, W. D. Lu, and H. Nazarian, "3D ReRAM with field assisted super-linear threshold (FAST TM) selector technology for super-dense, low power, low latency data storage systems," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, 2015, p. 575.

[47] D. Rinerson, C. J. Chevalier, S. W. Longcor, W. Kinney, E. R. Ward, and S. Kuo-Ren Hsia, "Re-writable memory with non-linear memory element," 2005.

[48] I. E. Ebong and P. Mazumder, "Self-controlled writing and erasing in a memristor crossbar memory," *IEEE Transactions on Nanotechnology*, vol. 10, no. 6, pp. 1454–1463, nov 2011.

[49] W. Lu, K. H. Kim, T. Chang, and S. Gaba, "Two-terminal resistive switches (memristors) for memory and logic applications," in *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 2011, pp. 217–223.

[50] X. Zhu, Y.-H. Tang, C.-Q. Wu, J.-J. Wu, and X. Yi, "Impact of multiplexed reading scheme on nanocrossbar memristor memory's scalability," *Chinese Physics B*, vol. 23, no. 2, p. 028501, feb 2014.

[51] S. J. Ham, H. S. Mo, and K. S. Min, "Low-Power VDD/3 write scheme with inversion coding circuit for complementary memristor array," *IEEE Transactions on Nanotechnology*, vol. 12, no. 5, pp. 851–857, 2013.

[52] T. Roska, "Analogic CNN Computing: Architectural, Implementation, and Algorithmic Advances - a Review," in *Cellular Neural Networks and Their Applications Proceedings, 1998 Fifth IEEE International Workshop on*, no. April, 1998, pp. 3–10.

[53] B. E. Shi and L. O. Chua, "Resistive grid image filtering: Input/output analysis via the CNN framework," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 39, no. 7, pp. 531–548, 1992.

[54] P. Kinget and M. S. J. Steyaert, "A programmable analog cellular neural network CMOS chip for high speed image processing," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 235–243, 1995.

[55] H. Li, X. Liao, C. Li, H. Huang, and C. Li, "Edge detection of noisy images based on cellular neural networks," *Communications in Nonlinear Science and Numerical Simulation*, vol. 16, no. 9, pp. 3746–3759, 2011.

[56] J. Zhao, H. Wang, and D. Yu, "A new approach for edge detection of noisy image based on CNN," *International Journal of Circuit Theory and Applications*, vol. 31, no. 2, pp. 119–131, 2003.

[57] T. Yoshida, J. Kawata, T. Tada, a. Ushida, and J. Morimoto, "Edge detection method with CNN," *SICE 2004 Annual Conference*, vol. 2, pp. 1721–1724, 2004.

[58] I. N. Aizenberg, "Processing of noisy and small-detailed gray-scale images using cellular neural networks," *Journal of Electronic Imaging*, vol. 6, no. July, pp. 272–285, 1997.

[59] B. E. Shi, "The effect of mismatch in current- versus voltage-mode resistive grids," *International Journal of Circuit Theory and Applications*, vol. 37, pp. 53–65, 2009.

[60] M. Hnggi and L. O. Chua, "Cellular neural networks based on resonant tunnelling diodes," *International Journal of Circuit Theory and Applications*, vol. 29, no. 5, pp. 487–504, 2001.

[61] P. Mazumder, S. R. Li, and I. E. Ebong, "Tunneling-based cellular nonlinear network architectures for image processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 4, pp. 487–495, 2009.

[62] W. H. Lee and P. Mazumder, "Motion detection by quantum-dots-based velocity-tuned filter," *IEEE Transactions on Nanotechnology*, vol. 7, no. 3, pp. 355–362, 2008.

[63] J. Schulman, H. De Los Santos, and D. Chow, "Physics-based RTD current-voltage equation," *IEEE Electron Device Letters*, vol. 17, no. 5, pp. 220–222, may 1996.

[64] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," pp. 629–639, 1990.

[65] A. B. Torralba, "Analogue Architectures for Vision Cellular Neural Networks and Neuromorphic Circuits," Ph.D. dissertation, 1999.

[66] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *Circuits and Systems Magazine, IEEE*, pp. 40–58, 2009.

[67] J. C. Hoskins and D. M. Himmelblau, "Process control via artificial neural networks and reinforcement learning," *Computers & chemical engineering*, vol. 16, no. 4, pp. 241–251, 1992.

[68] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, no. 3, pp. 283–302, 1997.

[69] M. J. Mataric, "Reinforcement Learning in the Multi-Robot Domain," in *Robot Colonies*, R. C. Arkin and G. A. Bekey, Eds. Springer US, 1997, vol. 4, pp. 73–83.

[70] J. Bagnell and J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2, 2001, pp. 1615–1620 vol.2.

[71] W. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 4, 2002, pp. 3404–3410.

[72] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin, "Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 3712–3717.

[73] N. Zheng and P. Mazumder, "Hardware-Friendly Actor-Critic Reinforcement Learning Through Modulation of Spiking-Timing Dependent Plasticity," *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–1, 2016.

[74] W. T. Miller, R. S. Sutton, and P. J. Werbos, "A menu of designs for reinforcement learning over time," in *Neural Networks for Control*. MIT Press, 1995, pp. 67–95.

[75] P. J. Werbos, "Approximate Dynamic Programming for Real-time Control and Neural Modeling," in *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Aproaches*, D. White and D. Sofge, Eds. New York: Van Nostrand Reinhold, 1992.

[76] S. N. Balakrishnan and V. Biega, "Adaptive-critic-based neural networks for aircraft optimal control," *Journal of Guidance, Control, and Dynamics*, vol. 19, no. 4, pp. 893–898, 1996.

[77] D. Liu, X. Xiong, and Y. Zhang, "Action-dependent adaptive critic designs," in *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, 2001, pp. 990–995 vol.2.

[78] R. Padhi, N. Unnikrishnan, X. Wang, and S. N. Balakrishnan, "A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear systems," *Neural Networks*, vol. 19, no. 10, pp. 1648–1660, 2006.

[79] W. S. Lin, L. H. Chang, and P. C. Yang, "Adaptive critic anti-slip control of wheeled autonomous robot," *IET Control Theory Applications*, vol. 1, no. 1, pp. 51–57, 2007.

[80] H. Zhang, Q. Wei, and Y. Luo, "A novel infinite-time optimal tracking control scheme for a class of discrete-time nonlinear systems via the greedy HDP iteration algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 4, pp. 937–942, 2008.

[81] K. Tang and G. Srikant, "Reinforcement Control via Heuristic Dynamic Programming," in *IEEE International Conference on Neural Networks*, 1997, pp. 1766–1770.

[82] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, *Direct Neural Dynamic Programming*, 2004.

[83] W. Liu, G. K. Venayagamoorthy, and D. C. Wunsch, "A heuristic-dynamic-programming-based power system stabilizer for a turbogenerator in a single-machine power system," *IEEE Transactions on Industry Applications*, vol. 41, no. 5, pp. 1377–1385, 2005.

[84] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, "Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 764–773, 2002.

[85] H. Wu and B. Luo, "Heuristic Dynamic Programming Algorithm for Optimal Control Design of Linear Continuous-Time Hyperbolic PDE Systems," *Industrial & Engineering Chemistry Research*, vol. 51, no. 27, pp. 9310–9319, 2012.

[86] D. C. Wunsch and M. S. Iyer, "Dynamic Re-optimization of a Fed-batch Fermentor Using Heuristic Dynamic Programming," *Institute of Electrical and Electronics Engineers (IEEE)*, 1999.

[87] F. M. Dias, A. Antunes, and A. M. Mota, "Artificial neural networks: A review of commercial hardware," *Engineering Applications of Artificial Intelligence*, vol. 17, no. 8, pp. 945–952, 2004.

[88] L. M. Reyneri, M. Chiaberge, and L. Zocca, "{CINTIA}: a neuro-fuzzy real time controller for low power embedded systems," in *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, 1994, pp. 392–403.

[89] P. Ienne, "Digital Hardware Architectures for Neural Networks," *SPEEDUP Journal*, vol. 9, no. 1, pp. 1–12, 1995.

[90] M. Skrbek, "Fast Neural Network Implementation," *Neural Network World*, vol. 9, no. 5, pp. 375–391, 1999.

[91] A. Schmid, Y. Leblebici, and D. Mlynek, "Mixed analogue-digital artificial-neural-network architecture with on-chip learning," *IEE Proceedings - Circuits, Devices and Systems*, vol. 146, no. 6, pp. 345–349, 1999.

[92] P. Arena, L. Fortuna, M. Frasca, and L. Patané, "A CNN-based chip for robot locomotion control," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 9, pp. 1862–1871, 2005.

[93] P. Arena, L. Fortuna, M. Frasca, L. Patane, and M. Pollino, "An autonomous mini-hexapod robot controlled through a CNN-based CPG VLSI chip," in *2006 10th International Workshop on Cellular Neural Networks and Their Applications*, 2006, pp. 1–6.

[94] B. Noory and V. Groza, "A reconfigurable approach to hardware implementation of neural networks," in *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, 2003, pp. 1861–1864.

[95] A. Muthuramalingam, S. Himavathi, and E. Srinivasan, "Neural Network Implementation Using FPGA : Issues and Application," *Journal of Information Technology*, vol. 4, no. 2, pp. 86–92, 2008.

[96] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *1990 IJCNN International Joint Conference on Neural Networks*, 1990, pp. 537–544 vol.2.

[97] L. Lewyn and N. Williams, "Is a New Paradigm for Nanoscale Analog CMOS Design Needed?" *Proceedings of the IEEE*, vol. 99, no. 1, pp. 3–6, 2011.

[98] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, 2010.

[99] L. Smith, "Implementing Neural Models in Silicon," *Handbook of Nature-Inspired and Innovative Computing*, no. May, pp. 433–475–475, 2006.

[100] F.-Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: an introduction," *... Intelligence Magazine, IEEE*, no. May, pp. 39–47, 2009.

[101] A. AlTamimi, M. Abu-Khalaf, and F. Lewis, "Heuristic dynamic programming nonlinear optimal controller," *Machine Learning*, no. February, 2009.

[102] D. Liu, X. Yang, D. Wang, and Q. Wei, "Reinforcement-Learning-Based Robust Controller Design for Continuous-Time Uncertain Nonlinear Systems Subject to Input Constraints," *Cybernetics, IEEE Transactions on*, vol. 45, no. 7, pp. 1372–1385, 2015.

[103] D. Hammerstrom, "A Survey of Bio-Inspired and Other Alternative Architectures," in *Technology*. Wiley, 2010, vol. 4, pp. 149–285.