UC Irvine
UC Irvine Electronic Theses and Dissertations

Title
Efficient Offline and Online Training of Memristive Neuromorphic Hardware

Permalink
https://escholarship.org/uc/item/4t9278vc

Author
Fouda, Mohammed

Publication Date
2020

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Efficient Offline and Online Training of Memristive Neuromorphic Hardware

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Mohammed E. Fouda

Dissertation Committee:
Professor Ahmed Eltawil, Chair
Professor Fadi Kuradhi
Professor Emre Neftci
Professor Nikil Dutt

2020

© 2020 Mohammed E. Fouda

DEDICATION

To my parents
who I owe everything.

To my dear wife, Radwa, for her continuous support.

To Adam, my son, wishing him a successful life.

TABLE OF CONTENTS

Page

LIST OF FIGURES **vii**

LIST OF TABLES **xii**

LIST OF ALGORITHMS **xiv**

ACKNOWLEDGMENTS **xv**

VITA **xvi**

ABSTRACT OF THE DISSERTATION **xviii**

1 Introduction **1**
1.1 Existing Digital Neuromorphic Hardware 2
1.2 Emerging Neuromophic Hardware 3
1.3 Contributions 6
1.4 Dissertation Organization 8

2 Memristive Neuromorphic Hardware Challenges **10**
2.1 Weight Mapping 12
2.2 Endurance and Retention 14
2.3 Sneak Path Effect 16
2.4 Delay 19
2.5 Asymmetric Non-linearity Conductance Update Model 20
2.6 Stuck-At Fault Effect 24

3 Crossbar Array Modeling and Analysis **26**
3.1 Nano-scale Interconnect Parasitics Extraction 27
3.1.1 Interconnect Wire Resistance 28
3.1.2 Interconnect Wire Inductance 29
3.1.3 Interconnect Capacitance 30
3.1.4 Interconnect Conductance 31
3.1.5 Comparison and Discussion 32
3.2 Crossbar Array Architecture 34
3.2.1 Switching Cell Structure and Circuit Model 34
3.3 Compact Mathematical Model for the crossbar array 36

3.3.1 Mathematical Model of Basic Switching Cell 38
3.3.2 Assembled Crossbar Array Model 38
3.3.3 Crossbar Modeling Equations 40
3.4 Practical Interconnect 44
3.4.1 Loading Effect On Crossbar Model 46
3.4.2 Solution of Interconnect Model 47
3.4.3 Simulation Results and Validation 48
3.5 Elmore's Delay Extension for Crossbar Arrays 49
3.5.1 Novel Mathematical Modeling of The Delay 49
3.5.2 Delay Verification and Results 51
3.6 Discussion and Comparison 52
3.7 Conclusion 58

4 Reading and Writing Techniques for Crossbar Resistive Memories 60
4.1 Introduction 60
4.1.1 Crossbar Writing Schemes 63
4.2 Proposed One Step Row Readout Technique 64
4.2.1 Proposed current sensing circuitry 68
4.2.2 Discussion and Comparison 73
4.2.3 Power Consumption Estimation 73
4.2.4 Bias Mismatch Effect 77
4.3 Modeling the write disturb problem 79
4.3.1 Optimality problem formulation and solution 80
4.3.2 Results and Discussion 81
4.4 One Step Row Write Technique 82
4.5 Combined Reading and Writing Circuitry 85
4.6 Non-Stationary Polar Codes for Resistive Memories 85
4.6.1 Non-stationary polar code construction 88
4.6.2 Polar Code Encoding and Decoding 89
4.6.3 Application to Resistive Crossbar arrays 93
4.6.4 Binary Asymmetric Channel Modeling 94
4.6.5 Punctured Polar Codes 95
4.7 Conclusion and Future Perspective 96

5 Wire Resistance-aware Training for efficient Offline Learning 100
5.1 Background and Related Work 101
5.2 Multi-bit RRAM Device Under Study 102
5.2.1 MVM using RCAs 104
5.2.2 Quantized Weight Mapping 106
5.2.3 Sensing Circuit Realization 107
5.3 Effect of Wire Resistance on Sneak Path Problem 111
5.3.1 Nano-scale Crossbar Parasitics 111
5.3.2 Source and Neuronal Resistance 112
5.3.3 Crossbar Simulation and Sneak Path Problem 114
5.4 BNNs Realization on Binary Crossbar Arrays 116

5.4.1 BNNs on RRAM crossbars . . . 116
5.4.2 Necessity of Large Weight Array Partitioning . . . 117
5.4.3 Evaluation Results . . . 120
5.4.4 DNN Framework . . . 122
5.5 Mask Technique: *Making Training Possible* . . . 125
5.5.1 Mask Method . . . 125
5.5.2 Mask Generation . . . 126
5.5.3 Architecture Solutions . . . 128
5.6 BNN Results and Discussion . . . 131
5.6.1 Impact of Retraining on Binarized MLP Network . . . 133
5.6.2 Impact of Retraining on Binarized CNNs . . . 135
5.6.3 Power Consumption in Resistive Crossbar Arrays . . . 137
5.6.4 Discussion and Comparison . . . 137
5.7 Extending Mask technique for Quantized Neural Networks . . . 140
5.7.1 Effect of Wire Resistance Problem on QNNs . . . 140
5.7.2 Proposed IR-QNN Training and Inference . . . 144
5.7.3 Batch Normalization during Inference . . . 149
5.7.4 QNN Experimental Setup . . . 152
5.8 QNN Results and Discussion . . . 153
5.8.1 Stuck-At Fault Effect . . . 154
5.8.2 Effect of Device Variability . . . 156
5.8.3 Effect of Limited Retention . . . 157
5.8.4 Power and Area Results . . . 158
5.8.5 Time Overhead Comparison . . . 159
5.9 Driver and Neuronal Circuits Requirements . . . 159
5.10 Conclusion . . . 163

6 Efficient Online Learning 165
6.1 Independent Component Analysis using RRAMs . . . 166
6.1.1 Proposed RRAM's learning method . . . 169
6.1.2 De-mixing Example and Results . . . 173
6.1.3 Power, area and speed comparison with CMOS realization . . . 174
6.2 Deep Neural Network with Local Learning . . . 176
6.2.1 Neural Network Model . . . 176
6.2.2 Local Learning with Crossbar Arrays . . . 177
6.2.3 Experimental Setup . . . 180
6.2.4 Online Training Under Asymmetric Nonlinear Updates . . . 180
6.2.5 Stochastic Ternary Update Method . . . 182
6.2.6 Results and Comparsion . . . 183
6.2.7 RRAM's Energy Update Model . . . 185
6.2.8 Limited Endurance . . . 187
6.2.9 Conductance Degradation Model . . . 187
6.3 Error-triggered Learning . . . 189
6.3.1 Local Losses and Local Errors . . . 189
6.3.2 Large-scale Simulation Experiments . . . 190

6.4 Conclusion 191

7 Prospects on Memristive Neuromorphic Hardware 193
7.1 Synaptic Plasticity and Learning in SNN 193
7.1.1 Gradient-based Learning in SNN and Three-Factor Rules 195
7.2 Stochastic Spiking Neural Networks 202
7.2.1 Learning in Stochastic Spiking Neural Networks 203
7.2.2 Three Factor Learning in Memristor Arrays 205

Bibliography 208

A KCL Formulation of the crossbar 226
A.0.1 Linear Switching Modeling Equations Derivation 229
A.0.2 Nonlinear Switching Modeling Equations Derivation 232

LIST OF FIGURES

Page

1.1 Crossbar containing switching devices (Blue). 5

2.1 Crossbar array realization of one layer neural network. 11
2.2 Mapping synaptic weight into conductances. 12
2.3 Mapping one synaptic weight into two conductances. 14
2.4 Effect of the wire resistance on the measured weights for 512×512 crossbar array at with 0.1Ω wire resistance. 3D plots of random weights distributed across the array, (a) without partitioning and (c) with partitioning into 64×64 crossbar arrays; and the measured weights with the sneak path problem versus the desired values for (b) the entire array without partitioning and (d) with partitioning. 17
2.5 Realization of the partitioned matrices. 19
2.6 RRAM's conductance update (a) long term potentiation (b) long term depression. 24
2.7 Non-idealities of the RRAM:(a) asymmetric nonlinear weight update (b) device Variations 25

3.1 ANSYS Q3D simulation of three parallel interconnect wires. 28
3.2 Capacitive model of interconnect. 30
3.3 Resistance and inductance reactance per unit cell of interconnect wire. . . . 33
3.4 Conductance and capacitive susceptance per unit cell of interconnect wire. . 33
3.5 Summarized transmission line model. 33
3.6 Basic cell model. 35
3.7 Circuit model of the crossbar array. 36
3.8 Kirchhoff's law of every cell in crossbar array. 37
3.9 Interconnect array with loading 42
3.10 Kirchhoff's law at the load. 45
3.11 Transient verification between circuit simulation (dotted line) and analytical solution (solid lines) for 4×4 interconnect with (a) linear switching devices, and (b) parabolic switching devices. 48
3.12 Comparison with the calculated Elmore delay expression and the transient simulation results for a) all HRS, and b) all LRS. 52
3.13 Comparison between the calculated delay of floating and grounded inputs for a) all HRS, and b) all LRS. 53

3.14 Effect of seeding the crossbar array with random data for different cases of (a)$F = 5nm$ structure and (b) $F = 50nm$ structure. 54
3.15 Effect of changing loading capacitance for grounded inputs of $F = 5nm$ for (a) all HRS and (b) all LRS, and floating inputs for (c)all HRS, and (d) all LRS 55
3.16 Effect of changing loading capacitance for grounded inputs of $F = 50nm$ for (a) all HRS and (b) all LRS, and floating inputs for (c)all HRS, and (d) all LRS 56
3.17 Effect of using different devices for (a) $F = 5nm$ and (b)$F = 50nm$. 57
3.18 Comparison between the calculated delay of crossbar array containing linear and nonlinear switching devices for a) all LRS, and b) all HRS. 57
3.19 Simulation time comparison between SPICE and the proposed framework. . 58

4.1 Trends in memory capacity for emerging NVMs adopted from [3]. 61
4.2 (a) Crossbar array with the sneak path problem, and (b) cumulative probability of reading 512×512 array. 62
4.3 (a) Biasing scheme and (b) row reading scheme. 62
4.4 Basic bias schemes for writing crossbar based resistive memories: (a) 1/2 bias scheme, and (b)1/3 bias scheme. 63
4.5 The set and reset switching behavior of RRAMs 64
4.6 Schematic of full single-layered crossbar array. 65
4.7 Combined plot for the 512×512 array sensed current density with histogram for (a)linear and (b) nonlinear devices. 67
4.8 (a) Current conveyor principle and (b) schematic of proposed current sensing circuit. 71
4.9 Schematic of the full proposed readout circuit. 71
4.10 Simulation results of the proposed circuit. 73
4.11 Reading power versus the array size at $V_{DD} = 1.2V$ with and without wire resistance (dashed lines and solid lines) for (a) linear switching and (b) nonlinear switching devices 74
4.12 Input comparator swing and delay versus (a) bias voltage with $V_{in} = 1.2V$, (b) applied voltage with $V_B = 0.7V$. 75
4.13 General Bias scheme of writing crossbar based resistive memories 80
4.14 Normalized optimal bias parameters versus the array size with 10% tolerance. 82
4.15 Bit Error rate for the optimal Bias scheme versus the known bias schemes. . 82
4.16 Estimated power consumption of the optimal Bias scheme versus the known bias schemes for (a) linear $I - V$ switching devices, and (b) nonlinear $I - V$ switching devices. 83
4.17 Bias scheme for one step writing technique. 83
4.18 An illustrative example of writing a) 2×2 crossbar array; b) The wordline and bitline voltages, and c) the voltage drop across each device. 84
4.19 The whole schematic the resistive memory based on one step reading and writing techniques. 86

4.20 Parallel reading of the entire row in the crossbar. The columns and rows are grounded, except the row being read. The red arrow shows the sensed current flowing through wire resistances and RRAMs. 86
4.21 (a) Measured current per cell, and its histogram for bitline number (b) 1, (c) 16, and (d) 32. 87
4.22 Polar encoding with $N = 8$ channels, with permutation $\pi = [0, 4, 2, 6, 1, 5, 3, 7]$. 91
4.23 Full system model. 91
4.24 Performance evaluation for BSCs with linearly spaced cross-over probabilities. $N = 1024, k = 512$. 92
4.25 Performance evaluation for a (32×32) crossbar array, code rate $k/n = 0.8$. . 94
4.26 BER performance under BSC and BAC modeling for a (32×32) crossbar, code rate $k/n = 0.8$. 95
4.27 Punctured polar encoding over the (32×32) crossbar array for code rate $k/n = 0.8$. 96
4.28 Stacked crossbar arrays sharing the readout circuitry. 98

5.1 Au/Al_2O_2/HfO_2/TiN-based RRAM device adopted from [4] (a) device behavior under incremental step pulse programming and (b) current-voltage characteristics. 103
5.2 Histogram and cumulative distribution function (CDF) of 100 measured samples per state. 103
5.3 Matrix Vector Multiplication using separate RCAs. 105
5.4 Possible weight mappings for the used RRAM device. 108
5.5 Schematic of proposed latched current sensing. 109
5.6 (a)Transient simulations for different Input currents, (b) output voltage versus differential input current characteristics, and (c) AC response of the transresistance gain. 110
5.7 Effect of parasitic resistances on the normalized synaptic current 113
5.8 Normalized root mean square deviation of the sensed current compared to the ideal case versus changing (a) the wire resistance and fixing the neuron resistance for different crossbar arrays and (b) the neuron resistance for different wire resistances for 128×128 array. 114
5.9 Synaptic weight realizations using RRAMS; (a) balanced and (b) unbalanced realizations. 117
5.10 Simulation time comparison between SPICE and numerical simulator, adopted from [5], for performing MVM of 256×256 array partitioned into 32×32, 64×64 and 128×128 and for different number of input samples. 119
5.11 Realization of the partitioned matrices. 121
5.12 Normalized sensed current of two crossbar arrays with a) $r = 500$ and b) $r = 10^4$. 121
5.13 Positive and negative masks of the 128x128 crossbar array 127
5.14 Positive and negative masks for 110×128 logical array using 128×128 physical array. 129
5.15 Comparison between 128×128 physical array size and 110×128 physical array size for 110×128 logical subarray mask 129

5.16 Effect of changing Wire resistance on the sneak path problem. 130
5.17 Effect of changing number of reference columns on the mask. 130
5.18 Normalized measured weight with changing the number of reference columns. 131
5.19 DNN experiment flow (Sn means Step n). 131
5.20 DNN train and test accuracy during retraining the network. 132
5.21 DNN validation accuracy before retraining (un-hashed bars) after retraining (hashed bars). 132
5.22 Conventional neural networks validation accuracy for different reference columns. 135
5.23 Power dissipation comparison between proposed masks, 110×128 logical mask (un-hashed bars) and 128×128 mask (hashed bars) in the crossbar arrays used for the MNIST network. 136
5.24 DNN validation accuracy after retraining and validation of MNIST network for different r values and for one (diamond), five (square) and nine (circle) reference columns 136
5.25 Training time comparison (per iteration). 140
5.26 (a) Measured weight per cell for 128×128 and 256×256 crossbar arrays at $R_w = 1\Omega$, (b) Histogram of the measured conductance normalized to G_{max} of the crossbar arrays shown in (a), (c) (5) Histogram of the measured conductance for 1-bit case for different wire resistance, (d) and (e) Histograms of the normalized conductance for 3-bit and 4-bit cases at $R_w = 1\Omega$. 141
5.27 Circuit model of the crossbar array. 142
5.28 The generated masks for 2-bit neural network training. 148
5.29 Effect of changing the SAF percentage on the recognition accuracy using mapping-I. Solid line refers to multiple mask trained network results and dashed line shows stochastic mask trained network results. 155
5.30 Effect of changing the variability of each conductance's state on the recognition accuracy using mapping II. Dashed lines show the validation accuracy drop with pre-trained weights with multiple mask technique. Solid lines show the accuracy drop after retraining with additive noise. 155
5.31 MNIST Recognition Accuracy against normalized retention time for a) $v_d = 10$, and b) $v_d = 0.1$. 158
5.32 Static power dissipation in RCAs per image at $0.1V$ read voltage. 159
5.33 Effect of the driver resistance on the performance of MNIST recognition; (a) and (b) for Mapping-I and (c) and (d) for mapping-II with multiple mask training for (a) and (c) and with stochastic mask training for (b) and (d). . . 160
5.34 Effect of the load resistance (the input resistance of the sensing circuit) on the performance of MNIST recognition; (a) and (b) for Mapping-I and (c) and (d) for mapping-II with multiple mask training for (a) and (c) and with stochastic mask training for (b) and (d). 161
5.35 Effect of the neuronal offset current deviation on the performance of MNIST recognition for Mapping-I; (a)multiple mask training and (b) stochastic mask training. 162

6.1 (a)Observed signals (mixtures of sources), and (b) original sources. 167
6.2 Illustration of used RRAM crossbar array to realize the ICA preceptron network. 169

6.3 (a) Evolution of the weights, and (b) Visual results of the input and the output. 171
6.4 The online training results versus training time (a) Evolution of the weights, (b) surprise Energy function, (c) Required programming pulses for each weight, and (d) Visual results of the input and the output. 175
6.5 Single memristive spiking neural network layer with online local learning capabilities. 177
6.6 RRAM's conductance change relation with number of pulses 179
6.7 Ideal validation with limited weight range and with stochastic rounding at $K = 1/20$ and $N = 100$. 180
6.8 (a) Test accuracy of the outer classifier for different scaling factors of ternary update rule and (b) total number of positive and negative update pulses for each epoch. 183
6.9 MNIST recognition accuracy for three different devices with a) $\gamma_P = \gamma_D = 1$ and b) $\gamma_P = 3\gamma_D = 3$. 185
6.10 Total Write Energy for each epoch for $\gamma_P = \gamma_D = 1$ and $\gamma_P = 3\gamma_D = 3$ . . . 186
6.11 The potentiation and depression conductance degradation for different normalized programming pulses. 188
6.12 MNIST test accuracy for different batch sizes with 10^6 endurance value. . . . 188

7.1 Network Architecture for Event-driven Random Backpropagation (eRBP). Reproduced from [6] 199
7.2 Deep continuous local learning example.Reproduced from [7]. 201
7.3 The Synaptic Sampling Machines (SSM). (a) At every occurrence of a pre-synaptic event, a pre-synaptic event is propagated to the post-synaptic neuron with probability p. (b) Synaptic stochasticity can be viewed as a continuous DropConnect method [8] where weights are masked by a binary matrix $\Theta(t)$, where $*$ denotes element-wise multiplication. (c) SSM Network architecture, consisting of a visible and a hidden layer. Reproduced from [9] 204

LIST OF TABLES

Page

1.1 Device charactersitics of mainstream and emerging memory technologies [10]. 4

2.1 Extracted Potentiation parameters of the $Mo/TiO_x/TiN$ device reported in[11]. 23
2.2 Extracted depression parameters of the $Mo/TiO_x/TiN$ device reported in[11]. 23

3.1 Extracted parasitic inductance of three parallel interconnect wires at 1GHz . 30
3.2 Extracted parasitic capacitance of three parallel interconnect wires at 1GHz 31

4.1 The proposed readout circuit parameters. 72
4.2 Performance metric of reading a complete $N \times N$ array. These results are adopted from [12, 13]. 76

5.1 Weight-conductance mapping for quantized states. 107
5.2 Understudy Binary neural networks. 124
5.3 MLP neural network validation accuracy using 128×128 crossbar arrays configuration. 124
5.4 Convolutional neural network validation accuracy using 128×128 crossbar arrays configuration. 125
5.5 MLP network validation accuracy before retraining 132
5.6 MNIST validation Accuracy for different hardware configurations 134
5.7 Comparison between 1T1R and 0T1R architectures. 139
5.8 Fitted Lognormal distributions of multiplicative noise for each state. 147
5.9 MLP network configuration (MNIST dataset) 150
5.10 CNN configuration (CIFAR10 dataset) 150
5.11 Validation accuracy without retraining. 151
5.12 MNIST dataset validation results using Mapping-I after retraining with M. Noise: Multiplicative Noise, Single mask set, Sto. Mask: Stochastic Mask and Multiple Mask set 151
5.13 MNIST dataset validation results using Mapping-II after retraining with stochastic mask and multiple mask sets. 151
5.14 CIFAR10 dataset validation results after retraining. 153

6.1 Comparison between RRAM- and transistor-based ICA realization 176
6.2 Extracted model parameters of the three devices understudy. ANLF is the asymmeteric nonlinearity factor and ASF is the asymmetery factor, ... 184

6.3 Recognition Error in Idealized Spiking Neural Network Simulations Averaged over 5 Runs. 191

LIST OF ALGORITHMS

Page

5.1 Modified BinaryNet algorithm 123
5.2 Proposed IR-QNN Training Algorithm 145
6.1 Proposed Training Algorithm. 173

ACKNOWLEDGMENTS

Firstly, I would like to express my deepest gratitude to my adviser, Professor Ahmed Eltawil, for his support and motivation throughout my PhD journey at UCI. I am thankful to him for giving me the opportunity to expand my knowledge to new and diverse research directions which are reflected in this thesis.

Secondly, I would also like to thank Prof. Fadi Kurdahi for the thoughtful discussions which always opened new doors to me and helped me overcome several challenges in my research. He always promoted my work and encouraged me to go further. I would also like to gratefully thank Prof. Emre Neftci for teaching me two courses on neuromorphic engineering in which I learned a lot about the field. The conversations with him were always exciting and motivating to explore new directions in this field.

I would also like to thank Prof. Nikil Dutt for dedicating some of his time to serve in my qualification and defense committees.

I would also like to thank our collaborators Dr. Marwen Zorgui and Prof. Zhiying Wang at UCI and Prof. Jongeun Lee and Sugil Lee at UNIST, Korea who have been actively co-worked with me on several successful projects with fruitful discussions.

I would like to thank my friends for their support and motivation especially, Mahmoud Taha, Omar Abdallah, Mohammed Eletriby, Mohammed Abdelghany, Wael Elsharkasy.

Most importantly, I would like to thank my dear parents, my twin brother, and my sisters who were always supporting me and encouraging me with their best wishes.

Last but not least, this work would not have been possible without the constant support of my beloved wife, Radwa Hanafy who sacrificed a lot of things and taking care of our son, Adam, to give me a chance to focus and finish my PhD.

Thank you all.

Vita

Mohammed E. Fouda

EDUCATION

Doctor of Philosophy in Electrical Engineering and Computer Science **2020**
University of California-Irvine *Irvine, California*

Master of Science in Engineering Mathematics **2014**
Cairo University, *Giza, Egypt*

Bachelor of Science in Electronics and Communications Engineering **2008**
Cairo University, *Giza, Egypt*

RESEARCH EXPERIENCE

Graduate Research Assistant **2015 – 2020**
University of California, *Irvine, California*

Graduate Research Assistant **2011 – 2014**
Cairo University, *Giza, Egypt*

PUBLICATIONS

1. **M. E. Fouda**, S. Lee, J. Lee, A. Eltawil, and F. Kurdahi, ' ' IR-QNN Framework: An IR Drop-Aware Offline Training Of Quantized Crossbar Arrays' ', under review .
2. M. Payvand, **M. E. Fouda**, F. Kurdahi, A. Eltawil and E. Neftci "Error-triggered Three-Factor Learning Dynamics for Crossbar Arrays", AICAS, 2020.
3. S. Lee, **M. E. Fouda**, J. Lee, A. Eltawil, and F. Kurdahi, ' 'Efficient Sneak Path-aware Training of Binarized Neural Networks' ', DAC, Under review.
4. **M. E. Fouda**, E. Neftci, A. Eltawil and F. Kurdahi, ' ' Effect of Asymmetric Nonlinearity Dynamics in RRAMs on Spiking Neural Network Performance ' ',Asilomar Signal Processing, 2019.
5. **M. E. Fouda**, F. Kurdahi, A. Eltawil, E. Neftci, " Spiking Neural Networks for Inference and Learning: A Memristor-based Design Perspective", Memristive Devices for Brain-Inspired Computing, 2020.
6. M. Zorgui, **M. E. Fouda**, Z. Wang, A. Eltawil, and F. Kurdahi, ' 'Non-Stationary Polar Codes for Resistive Memories' ', Globecom, 2019.
7. **M. E. Fouda**,S. Lee, J. Lee, A. Eltawil, and F. Kurdahi, "Mask Technique for Fast and Efficient Training of Binary Resistive Crossbar Arrays", IEEE TNANO, 2019.

8. **M. E. Fouda**, A. Allagui, A. Elwakil, A. Eltawil, and F. Kurdahi ' 'Supercapacitor Discharge under Constant Resistance, Constant Current and Constant Power Loads" Journal of Power Sources, 2019

9. **M. E. Fouda**, E. Neftci, A. Eltawil, and F. Kurdahi, "Independent Component Analysis using RRAMs", IEEE Transactions on Nanotechnology, 2018.

10. **M. E. Fouda**, A. Eltawil, and F. Kurdahi, ' 'Activated Current Sensing Circuit for Memristive Neuromorphic Networks' ', NEWCAS, 2019.

11. **M. E. Fouda**, J. Lee, A. Eltawil, and F. Kurdahi, "Overcoming Crossbar Nonidealities in Binary Neural Networks Through Learning", NANOARCH, 2018.

12. **M. E. Fouda**, A. Eltawil, and F. Kurdahi, ' 'One Step Row Readout Technique for High-Density Resistive Memories' ', Arxiv.

13. **M. E. Fouda**, A. Eltawil and F. Kurdahi. "Modeling and Analysis of Passive Switching Crossbar Arrays", IEEE Transactions of Circuits and systems I, 2018.

14. **M. E. Fouda**, A. E. Khorshid, I. Alquaydheb, A. Eltawil, and F. Kurdahi "Extracting the Cole-Cole Model Parameters of Tissue-mimic Materials" BIOCAS, 2018

15. **M. E. Fouda**, A. Eltawil, and F. Kurdahi, "Minimal Disturbed Bits in Writing Resistive Crossbar Memories", NANOARCH, 2018.

16. **M. E. Fouda**, A. Eltawil and F. Kurdahi. "On One Step Row Readout Technique for Gateless Resistive Arrays", MWSCAS 2017 (Selected as one of the best 10 papers).

17. M. A. Bahloul, **M. E. Fouda**, R. Naous, M. A. Zidan, A. M. Eltawil, F. Kurdahi and K. N. Salama. "Design and Analysis of 2T-2M Ternary Content Addressable Memories", MWSCAS 2017.

18. H. Yantir,**M. E. Fouda**, A. Eltawil and F. Kurdahi, ' 'Process Variations-Aware Resistive Associative Processor Design", ICCD2016.

ABSTRACT OF THE DISSERTATION

Efficient Offline and Online Training of Memristive Neuromorphic Hardware

By

Mohammed E. Fouda

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2020

Professor Ahmed Eltawil, Chair

Brain-inspired neuromorphic systems have witnessed rapid development over the last decade from both algorithmic and hardware perspectives. Neuromorphic hardware promises to be more energy- and speed- efficient as compared to traditional Von-Neumann architectures. Thanks to the recent progress in solid-state devices, different nanoscale-nonvolatile memory devices, such as RRAMs (memristors), STT-RAM and PCM, support computations based on mimicking biological synaptic response. The most important advantage of these devices is their ability to be sandwiched between interconnect wires creating crossbar array structures that are inherently able to perform matrix-vector multiplication (MVM) in one step. Despite the great potential of RRAMs, they suffer from numerous nonidealities limiting the performance, including, high variability, asymmetric and nonlinear weight update, endurance, retention and stuck at fault (SAF) defects in addition to the interconnect wire resistance that creates sneak paths. This thesis will focus on the application of RRAMs for neuromorphic computation while accounting for the impact of device nonidealities on neuromorphic hardware.

In this thesis, we first develop a compact SPICE-like framework for the resistive crossbar array that incorporates the RRAM device model and interconnect parasitics such as wire resistance, inductance, capacitance, and conductance. This framework is the corner-stone of

the simulation infrastructure developed in this work, allowing for $\geq 1000\times$ faster simulation results as compared to SPICE. Second, we propose novel reading and writing techniques to read and write the entire word in one clock cycle with an optimized bias scheme to minimize the write errors. To complete the memory design, the required reading and writing CMOS peripheral circuits are designed as well. Due to the inevitable existence of the sneak path problem in crossbar arrays, nonstationary polar codes are designed to mitigate the effect of this problem for crossbar-based memory applications showing a significant improvement in bit-error-rate performance.

In the third part, we propose software-level solutions to mitigate the impact of nonidealities, that highly affect the offline (*ex-situ*) training, without incorporating expensive SPICE or numerical simulations. We propose two techniques to incorporate the effect of sneak path problem during training, in addition to the device's variability, with negligible overhead. The first technique is inspired by the impact of the sneak path problem on the stored weights (devices' conductances) which we referred to as the mask technique. This mask is element-wise multiplied by the weights during the training. This mask can be obtained from measured weights of fabricated hardware. The other solution is a neural network estimator which is trained by our SPICE-like simulator. The test validation results, done through our SPICE-like framework, show significant improvement in performance, close to the baseline BNNs and QNNs, which demonstrates the efficiency of the proposed methods. Both techniques show the high ability to capture the problem for multilayer perceptron networks for MNIST dataset with negligible runtime overhead. In addition, the proposed neural estimator outperforms the mask technique for challenging datasets such as CIFAR10. Furthermore, other nonidealities such as SAF defects and retention are evaluated.

Fourth, we develop a model to incorporate the stochastic asymmetric nonlinear weight update in online (*in-situ*) training. We propose two solutions for this problem; 1) a compensation technique which is tested on a small scale problem to separate two Laplacian mixed

sources using online independent component analysis. 2) stochastic rounding and is tested on a spiking neural network with deep local learning dynamics showing only a $1 \sim 2\%$ drop in the baseline accuracy for three different RRAM devices. We also propose Error-triggered learning to overcome the limited endurance problem with only 0.3% and 3% drop in the accuracy for N-MNIST and DVSGesture datasets with around $33\times$ and $100\times$ reduction in the number of writes, respectively.

Finally, the prospects of this neuromorphic hardware are discussed to develop new algorithms with the existing resistive crossbar hardware including its nonidealities.

Chapter 1

Introduction

The Internet of Things (IoT) market is growing exponentially and will reach 1.7 trillion dollars in 2020 according to IDC Insights Research [14]. It is expected that IOT will contain over 26 billion devices excluding PCs, tablets, and smartphones. These devices include sensor-based medical devices, automobiles, manufacturing plants, power systems, and smart homes. IOT enable the data exchange between all these devices where it is expected that IOT will generate 500 zettabytes of data per year by 2019, coming from 50 billion connected devices, according to a report from Cisco. As a result, almost all IOT applications need a system to analyze patterns in this data, detect certain types of events and take decisions. It is thus imperative to build systems that are able to deal with massive data sets efficiently. Machine learning systems are the best candidate to perform these tasks and satisfy the requirements.

Machine Learning and particularly deep learning have become the *de facto* choice in solving a wide range of problems when adequate data is available. So far, machine learning has been mainly concerned more by the “learning” rather than the “machine”. This focus is natural given that von Neumann computers and GPUs capable of general-purpose processing offer excellent performance per unit of monetary cost. As the scalability of such processors hit difficult scalability and energy efficiency challenges, interest in dedicated, multicore and multiprocessor systems is increasing. This calls for increased efforts on improving the physical

instantiations of "machines" for machine learning. Physical instantiation of computations are challenging because the structure and nature of the physical substrate severely restrict the basic computational operations it can carry out. However, if the computations can be formulated in a way that they exploit the physics of the devices, then the efficiency and scalability can be drastically improved. In this line of thought, the field of neuromorphic engineering is arguably the one that has attracted the most attention and effort. The field's core ideas, communicated by R. Feynmann, C. Mead and other researchers in a series of lectures called physics of computation, elaborate on the analogies between the physics of ionic channels in the brain and those of CMOS transistors [15]. By building synapses, neurons, and circuits modeled after the brain and driven by similar physical laws, neuromorphic engineers would "understand by building" and help engineering novel computing technologies equipped with the robustness and efficiency of the brain.

1.1 Existing Digital Neuromorphic Hardware

In the last decade, there have been enormous advances in building and scaling neuromorphic hardware using mixed-signal [16, 17, 18, 19] and digital [20, 21, 22] technologies, including embedded learning capabilities and scales achieving 1M neurons per chip. A number of novel hardware based architectures for large scale neuromorphic systems have been demonstrated with many millions of neurons [1]. There are large scale systems with complementary approaches and divergent goals such as IBM TrueNorth chip which is built based on distributed digital neural models to enable real-time cognitive applications, Stanford Neurogrid which uses real-time sub-threshold analog neural circuits to be used in robotic control, Heidelberg BrainScaleS system that uses analog neural circuits running 10000 times faster than biological real time to be used in simulating very long biological models, and Manchester SpiNNaker machine which consists of a real-time digital many core system that implements neural and synapse models in software running on small embedded processors. The recent

fabricated neuromphic hardware with learning capabilities is Intel's Loihi chip in 2018 which is developped using by Intel's 14-nm FinFET technology [21]. This chip contains 128 neural cores with 1024 neuron per core and supports around 130,000 neurons and 130 million synapses. The chip supports different learning rules that are based on spiketiming and reward modulation.

A major limitation in these technologies is the memory required to store the state and the parameters of the system. For example, in both mixed-signal and digital technologies, synaptic weights are typically stored in SRAM, the densest, fastest and most energy-efficient memory we can currently locate next to the computing substrate [23, 20, 21]. Unfortunately, SRAMs are expensive in terms of area and energy, making on-chip memory small given the computational requirements. In fact, learning often requires higher precision parameters to average out noise and ambiguities in real-world data, especially in the case of gradient-based learning in neural networks [24].

1.2 Emerging Neuromophic Hardware

Many emerging switching devices can be used to build crossbars, such as memristors, phase change memory (PCM), and spin transfer torque (STT). Each one of these devices has its own characteristics and properties that make one device more suitable for certain applications as compared to other devices. Memristors have shown great promise in crossbar applications due to their nanometer level dimensions, high switching speed, long retention time, low programming power and non-volatile characteristics [25, 26]. A Memristor switches between two resistance states, low resistance state (LRS) which is called R_{on}, and high resistance state (HRS) which is called R_{off}. The switching ratio for typical memristosr is around $10^3 \sim 10^6$ [27, 28, 29]. Memristors have been proposed theoretically for the first time by Chua in 1971[30, 31]. In 2008, HP team announced the realization of a memristor based

Table 1.1: Device charactersitics of mainstream and emerging memory technologies [10].

	Mainstream Memories				Emerging Memories		
			Flash				
	SRAM	DRAM	NOR	NAND	STT-MRAM	PCRAM	RRAM
Cell area	$> 100F^2$	$6F^2$	$10F^2$	$< 4F^2$	$6\ 50F^2$	$4\ 30F^2$	$4\ 12F^2$
Multibit	1	1	2	3	1	2	4
Voltage	$< 1V$	$< 1V$	$> 10V$	$> 10V$	$< 1.5V$	$< 3V$	$< 3V$
Read time	$\sim 1ns$	$\sim 10ns$	$\sim 50ns$	$\sim 10\mu s$	$< 10ns$	$< 10ns$	$< 10ns$
Write time	$\sim 1ns$	$\sim 10ns$	$10\mu s - 1ms$	$100\mu s - 1ms$	$< 10ns$	$\sim 50ns$	$< 10ns$
Retention	N/A	$\sim 64ms$	$> 10y$	$> 10y$	$> 10y$	$> 10y$	$> 10y$
Endurance	$> 10^{16}$	$> 10^{16}$	$> 10^5$	$> 10^4$	$> 10^{15}$	$> 10^9$	$> 10^6 \sim 10^{12}$
Write Energy (J/bit)	$\sim 1f$	$\sim 10f$	$\sim 100p$	$\sim 10f$	$\sim 100f$	$\sim 10p$	$\sim 100f$

on TiO_2 [32]. Since that date, many studies have been introduced to realize memristors based on different materials. For instance, TaO_x memristors have demonstrated improved write/erase endurance by orders of magnitude while having nanosecond switching speeds, however the linear current-voltage (I-V) characteristic in the low resistance state limits their applications in large passive crossbar arrays[33, 34]. In this paper, without loss of generality, we will focus on memristors, as the switching devices. However. the analyses derived can be generalized to other switching devices.

Spin Transfer torque devices (STT) are a magnetic device and are built based on magnetic tunnel junctions (MTJ), where switching occurs due to tunnel magneto resistance effect [35] since MTJs are composed from three thin films: two ferromagnetic layers and an oxide layer. STT devices offer stable non-volatility, fast write/read access speed and excellent endurance [36]. MTJ switches between parallel resistance, RP, and antiparallel resistance, RAP, depends on the relative orientation. STT device can be used for storage memory and for computing[37, 38]. The switching ratio of STT is around $10^8 \sim 10^{10}$ due to its magnetic characteristics. Table 1.1 summerizes a typical comparsion between the mainstream and emerging memory technologies.

Recently, programmable crossbar architectures have been considered as a key enabler for integrated nanoscale electronics using emerging memory technologies [25]. Crossbar arrays

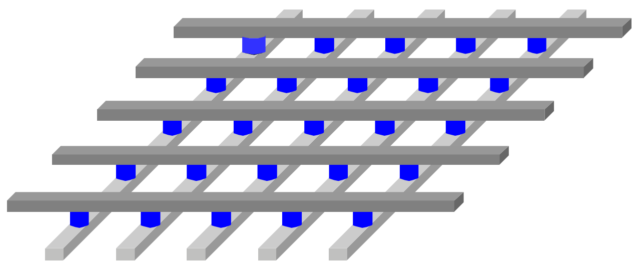


Figure 1.1: Crossbar containing switching devices (Blue).

have a small area footprint which enables high density structures. Moreover, crossbars can be stacked, creating 3D crossbars [39, 40], or used in FPGA structures such as in [41, 42]. Crossbar arrays that utilize resistive switching devices are considered the best candidate to replace traditional memories. These architectures are typically referred to as Resistive-switching random access memory (RRAM). An ideal RRAM has low programming voltage, high speed, and high density [43, 44]. However, passive RRAM arrays suffer from drawbacks, such as sneak path current [45], and interconnect parasitic resistance which limit the practical size of the RRAM array, and its access speed. Some techniques have been introduced to manage or reduce these effects. [46, 47, 48]. For instance, [46, 47] introduced different techniques to create sneak path free crossbar arrays using techniques such as multistage reading, multiport reading or transistor gating. In addition to RRAMs, crossbars have been used to perform computing operations [49, 50, 51]. A memory structure with computing capabilities using crossbars is referred to as in-memory computing or crossbar nanocomputer [52, 49, 53].

Crossbars can be treated as multi input multi output programmable neural networks. Thus, crossbars can be used in neural network applications such as pattern classification and recognition where the switching devices in the crossbar are trained and configured to enable classification of unknown patterns [54, 55, 56], as well as, bio-inspired/neuromorphic computing [57, 58]. Furthermore, crossbars are used in the field of physical cryptography to build physical unclonable functions (PUFs) using the randomness and disorder in the switching devices of the array [59, 60] .

1.3 Contributions

The contributions can be summarized as follows:

- A compact RC framework to include the effects of parasitics is proposed where Memristors are used as an exemplar device. Interconnect parasitics (resistance, inductance, capacitance and conductance) are extracted using ANSYS Q3D extractor for $5nm$ and $50nm$ feature sizes. A model for the crossbar is presented taking into consideration the stray and coupling capacitive parasitics of the crossbar. The derived model is based on state-space representation and provides more insight into the behavior of crossbar arrays containing either linear or nonlinear switching devices. The framework provides a closed form solution to evaluate Elmore delay, as well as the steady state response of the system. Signal delay is evaluated and compared for both grounded and floating interconnect inputs and verified against HSPICE, showing a perfect match.

- A row readout and writing techniques with circuitry is proposed that can be used to read/ write selector-less resistive crossbar based memories. High throughput reading and writing techniques are needed to overcome the memory-wall bottleneck problem and to enable near memory computing paradigm. The proposed technique can read the entire row of dense crossbar arrays in one cycle, unlike previously published techniques. The requirements for the readout circuitry are discussed and satisfied in the proposed circuit. Additionally, an approximated expression for the power consumed while reading the array is derived. A figure of merit is defined and used to compare the proposed approach with existing reading techniques. Finally, a quantitative analysis of the effect of biasing mismatch on the array size is discussed.

- The write disturb problem is mathematically formulated in terms of the bias parameters and optimized analytically. A closed form solution for the optimal bias parameters

is calculated. Results are compared with the 1/2 and 1/3 bias schemes showing a significant improvement.

- Polar coding technqiue is proposed to improve the bit error rate (BER) performance over channels with different reliability levels caused by sneakpath problem. We then apply the framework of non-stationary polar codes to the crossbar array and evaluate its BER performance under two modeling approaches, namely binary symmetric channels (BSCs) and binary asymmetric channels (BSCs). Finally, we propose a technique for biasing the proportion of high-resistance states in the crossbar array and show its advantage in reducing further the BER. Several simulations are carried out using a SPICE-like simulator, exhibiting significant reduction in BER.

- Some software techniques to capture the sneak path problem are proposed to enable off-chip binarized neural networks learning and weight transfer. The proposed technique can be easily integrated into any neural network training framework. Performance results show a significant improvement after retraining the network with the proposed mask technique. Two mask solutions have been proposed and studied to capture the sneak path problem in resistive crossbar arrays. Both mask solutions were successfully able to achieve classification accuracy close to the baseline accuracy. The trade-offs between the two solutions are discussed and compared in terms of accuracy, power and area.

- In addition, a fast and efficient training and validation framework is proposed to incorporate the wire resistance in Quantized DNNs without the need for computationally extensive SPICE simulations during the training. A fabricated four-bit $Au/Al_2O_3/HfO_2/TiN$ device is extensively modeled and integrated in the framework with two-mapping methods to realize the quantized weights. Simple and efficient system-level IR-drop estimation methods are integrated to the framework to accelerate the training. The SPICE validation results show the effectiveness of the proposed

method to capture the sneak path problem and can achieve the baseline accuracies with 2% and 4% drop in the worst-case scenario for MNIST and CIFAR 10 datasets, respectively. Finally, other nonidealities effects, such as stuck-at fault defects, variability and aging, are studied. Furthermore, the design considerations of the neuronal and the driver circuits are discussed.

- TiO_2 RRAMs are used to solve blind source separation problem through Independent Component Analysis (ICA). A local, unsupervised learning algorithm (error-gated Hebbian rule) to extract the independent components is deployed. The online evaluation of the weights during the training is studied taking into consideration the asymmetric nonlinear weight update behavior. The effects of the device variability are considered in the results. Finally, an example of de-mixing two Laplacian signals is given to demonstrate the efficacy of the approach.

- A memristive Spiking Neural Network (SNN) with local learning is introduced. Then, we study the effect of this asymmetric and nonlinear behavior on the spiking neural network performance and propose a method to overcome the performance degradation without extra nonlinearity cancellation hardware and read cycles. The performance of the proposed method approaches the baseline performance with $1 \sim 2\%$ drop in recognition accuracy.

- Finally, local error-triggered learning dynamics compatible with crossbar arrays and the temporal dynamics of spiking neural networks is proposed .

1.4 Dissertation Organization

This dissertation is organized as follows:

Chapter II discusses the challenges facing the memristive neuromorphic deployment such

interconnect parasitics, endurance, retention, asymmetric nonlinearity conductance update, and stuck-at fault.

Chapter III proposes a mathematical framework for simulating crossbar array structure with RC parasitics. The framework provides a closed form solution to evaluate Elmore delay, as well as the steady state response of the system.

Chapter IV proposes one step reading and writing techniques of memristive crossbar arrays for memory applications. In addition, we adapt error-correcting technique using polar codes for varying channels to treat the sneak path problem existing due to wire parasitics.

Chapter V proposes a different techniques to capture the sneak path problem to enable off-chip quanitized neural networks learning and weight transfer. The proposed techniques are validated on binarized and quantized neural networks on MNIST, CIFAR10 and SVHN datasets. The proposed mask solution was successfully able to achieve classification accuracy close to the baseline accuracy. Other nonidealities effects, such as stuck-at fault defects, variability and aging, are studied.

Chapter VI discusses the effect of asymmetric nonlinearity conductance update on the performance of simple and deep neural networks to solve source separation problem and MNIST recognition. In addition, the error-triggered learning is proposed to efficient online learning with RRAMs.

Chapter VII discusses the prospects of the memristive neuromorphic hardware.

Chapter 2

Memristive Neuromorphic Hardware Challenges

The basic building block in both spiking and artificial (non-spiking) neural network is the matrix-vector multiplication (MVM) which can be performed in a single step using crossbar structure unlike the conventional computing methods that required $N \times M$ steps or clock cycles. Figure 2.1 shows a single-layer crossbar based resistive neural network with M inputs and N outputs representing N perceptrons with M inputs each and the weights are stored in the memristors. The inputs to the perceptrons (presynaptic signals) are encoded in the input voltages, and the output of each perceptron is the sum of the currents passing through each memristor. This way, all currents within the same column can be linearly summed to obtain the postsynaptic currents. The total postsynaptic currents need sensing and shaping circuits to convert them into voltages and passed to subsequent neurons. In an artificial neural network, the postsynaptic current is summed up through the sensing circuit and passed through another shaping circuit to create the required neural activity such as sigmoid, tanh or rectified linear. With spiking neurons, the output of current sensing circuit is instead passed through a LIF circuit.

In neural networks, both positive (excitatory) and negative (inhibitory) connections are required. However, the RRAM conductance is positive by definition which only supports

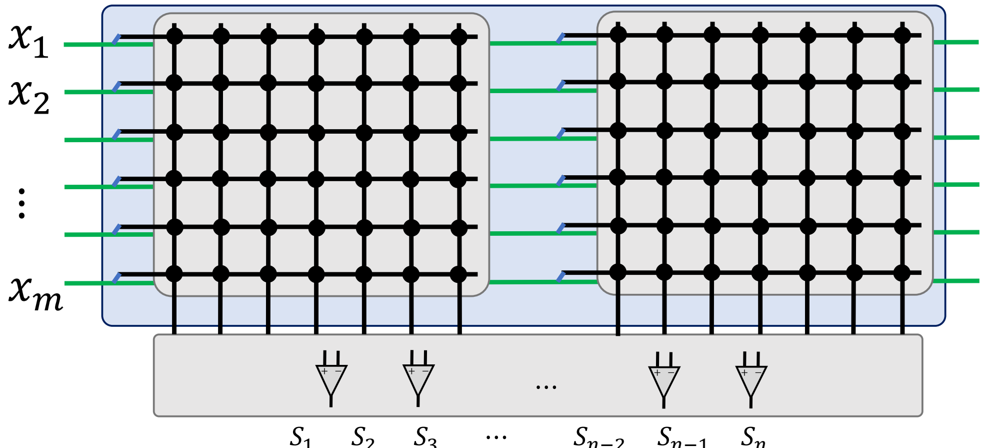


Figure 2.1: Crossbar array realization of one layer neural network.

excitatory or inhibitory connections. Two weight realization techniques are possible to create both excitatory and the inhibitory connections;

1) using two RRAMs per weight [61, 62] or

2) using one RRAM as weight in addition to one reference RRAM having a conductance set to $G_r = (G_{max} + G_{min})/2$ [63, 64].

The first realization has double the dynamic range $w \in [-\Delta G, \Delta G]$, where $\Delta G = G_{max} - G_{min}$, making it more immune to variability at a cost of double area, double power consumption during reading and additional programming operations. The second technique has one RRAM device, meaning that $w \in [-\Delta G/2, \Delta G/2]$ making it more prone to variability but the overall area is smaller, requires less power, and i

s easier to program (programming only one RRAM per weight). Due to the high variations in the existing devices, the first approach is commonly used with either one big crossbar or two crossbars (one for positive weights and the other one for negative weights as shown in 2.1). The output of the memristive neural network can be written as

$$S_j = \sum_{i=1}^{m} G_{ij} V_i \tag{2.1}$$

where S_j is the output of the jth neuron and $G_{ij} = G^+_{ij} - G^-_{ij}$, is the synaptic weight, and V_i is the ith input. The crossbar array forms the majority of the research in using RRAMs for neuromorphic computation [65]. In practice, RRAMs and crossbar structures suffer from many problems and do not behave ideally for computational purposes. These non-idealities can severely undermine the overall performance of applications unless they are taken into consideration while the training operation. After defining the mapping of synaptic weights to RRAM conductances, the following sections will overview these non-idealities in the light of neuromorphic computation and learning.

2.1 Weight Mapping

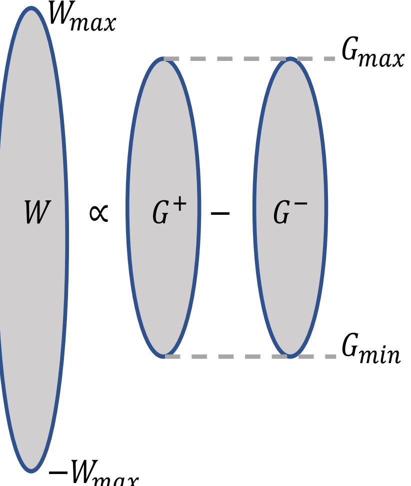


Figure 2.2: Mapping synaptic weight into conductances.

As discussed above, each weight is translated into two conductances which is one-to-two mapping which can be mathematically formulated as

$$G = G^+ - G^- = \frac{W}{W_{max}} \Delta G, \tag{2.2}$$

where W_{max} is the maximum value of the weight. If it is required to realize W_{max}, G^+ and G^- are set to G_{max} and G_{min}, respectively. The difference between the two conductances is

constant and proportional to the required weight value and each conductance is constrained to be between G_{min} and G_{max} as shown in Fig. 2.2. Thus, there are many possible realizations for each weight; for example the zero weight can be realized with any equal values of G^+ and G^-. Therefore, a criterion on selecting the weight mapping should be defined. This can be formulated as an optimization problem as follows:

$$\begin{aligned} &\text{minimize} \quad \mathcal{L}(\mathrm{G}^+, \mathrm{G}^-) \\ &\text{subject to} \\ &\qquad G_{min} \leq \mathrm{G}^+, \quad \mathrm{G}^- \leq G_{max}, \text{and} \\ &\qquad \mathrm{G}^+ - \mathrm{G}^- = \frac{\mathrm{W}}{W_{max}} \Delta G. \end{aligned} \tag{2.3}$$

where $\mathcal{L}$ is the objective function (objective function are discussed in further detail later in this chapter). G^+ and G^- are the positive and negative conductance matrices.

Since many conductance configurations are possible to obtain the same effective weight, additional criteria such as power consumption while reading (important for inference) or the write energy (important in online training) can be introduced. These constraints can be taken in consideration while training the network with a regularization term in the loss function. We note that the mapping is more important for offline training where the optimization is completed on software and the final weight values are transferred to the hardware.

One of the important metrics is the power consumption due to the high power consumption of RRAMs which is directly proportional to the device conductance ($P = GV^2$). Thus, in order to minimize the power, high resistances(low conductances) should be used. So, it is required to minimize both conductances. Thus, the objective function can be defined as $f = \sum_i \sum_j (G^+_{ij} - G^-_{ij})$. Figure 2.3 shows a simple example of realizing one weight using two coductances where the objective function is defined to minimize the power consumption example as $F = G^+ + G^-$. By applying the difference constraint, the objective function can

be written as $F = 2G^- + W_{ij}\Delta G/W_{max}$. Clearly, with the minimum value is achieved when one of the conducances is set to G_{min} and the other one is calculated based on that which will guarantee using the lowest possible conductances. Therefore, generalizing the case to realize NM weights, the optimal value of the objective function can be intuitively solved and equal $MNG_{min} + G_{max} \sum_i \sum_j |W_{ij}/W_{max}|$.

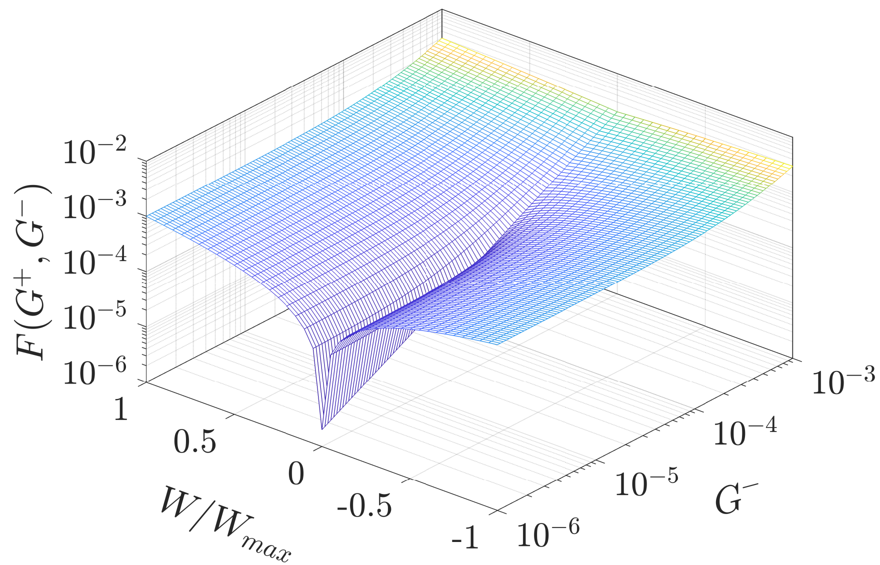


Figure 2.3: Mapping one synaptic weight into two conductances.

2.2 Endurance and Retention

An attractive purpose of RRAMs is to accelerate the training operation, especially the inference (feedforward) part of the training. However, one of the main issues facing RRAM deployment in neuromorphic hardware with online learning is endurance. In online learning, the devices are frequently updated, and especially so during gradient-based learning such as in artificial neural networks. However, each device has a limited number of potentiation and depression cycles [66, 67]. The endurance depends on the device's switching and electrode materials. For example HfO_x devices can achieve endurance up to 10^{10} cycles, but Al_2O_3 devices achieve endurance up to 10^4 [68]. With limited endurance, it is necessary to

complete the training before the devices degrade. The endurance requirement for learning is application-dependent. In standard deep learning, weight updates are usually performed every batch. Classification benchmarks such as MNIST handwritten digit recognition require writing around 10^4 cycles. However, real-world applications that involve reinforcement learning can easily scale to 10^8 cycles [69].

Furthermore, because neuromorphic hardware can be multi-purpose (i.e. the same device and be used to perform many different tasks), where a complete training of the network is performed every task. Consequently, the device endurance should be high enough to cover its lifetime use. There are some solutions to mitigate the endurance problem in machine learning scenarios:

- Full offline training: training is completed off the device and the final weights are transferred to the RRAM-based hardware. This requires an accurate modeling of the used devices, crossbar array and the sensing circuitry to be included into the training. This would require to verify the response of each part of the network to make sure that the response matches the simulated one [70].
- Semi-online training: A complete training cycle is performed offline, then the new weights are transferred to the devices. Then, an online retraining cycle is performed to improve the performance due to the existing impairments. Due to the smaller number of writing cycle, this solution would relax the endurance requirements. In [64], it was noticed that the network was able to recover after $10 \sim 20\%$ of the training epochs.

Once the online or the offline training is performed, the network can operate in the inference mode where only reading cycles are performed. In this case, the retention of the stored values becomes an important issue. As with endurance, RRAM retention is also dependent on the device materials and temperature. For example the HfO_x devices have around 10^4 seconds (2.78 hours) retention [71]. Although this might be sufficient for certain single-use

scenarios, such as biomedical applications, it is inadequate for IoT and autonomous control applications. There, retention values need to be more than 10^6 seconds across different temperature values (since retention degrades with increasing the temperature).

While both endurance and retention are important for machine inference and learning tasks, the learning approach may require one more over the other. I.e. full online learning requires high endurance and moderate retention, but, semi-online requires moderate endurance and retention.

2.3 Sneak Path Effect

The sneak path problem arises from the existence of the wire resistance which is inevitable in nanostructure crossbar arrays. The wire resistance creates many paths to the signal from each the input port to the output port. These multiple paths create undesired currents which perturb the reading of the weight. It is expected that the wire residence would reach around 92Ω for $5nm$ feature size [5], which is the expected feature size for crossbar technology according to International Technology Roadmap of Semiconductors (ITRS) [25]. Fig. 2.4a and Fig. 2.4b show an example of the sneak path problem in 512×512 with random weights. A linear switching device having a $10^6\Omega$ high resistance state and $10^3\Omega$ low resistance state is used while the wire resistance is 0.1Ω. Ideally, the measured weights should be similar to the measured weights, as shown in Fig. 2.4c. Despite the small value of the wire resistance, it has a very high effect on the weights stored in the crossbar arrays (Fig. 2.4a). The weights are exponentially decaying across the diagonal of the array where the cell (1,1) has the least sneak path effect and the cell (N,M) has the worst sneak path effect.

Some devices have a voltage-dependent conductance where the conductance is exponentially or quadratically function of the applied voltage [72]. This conductance non-linearity can help reduce the sneak path problem in resistive memories on crossbar or Xpoint structures

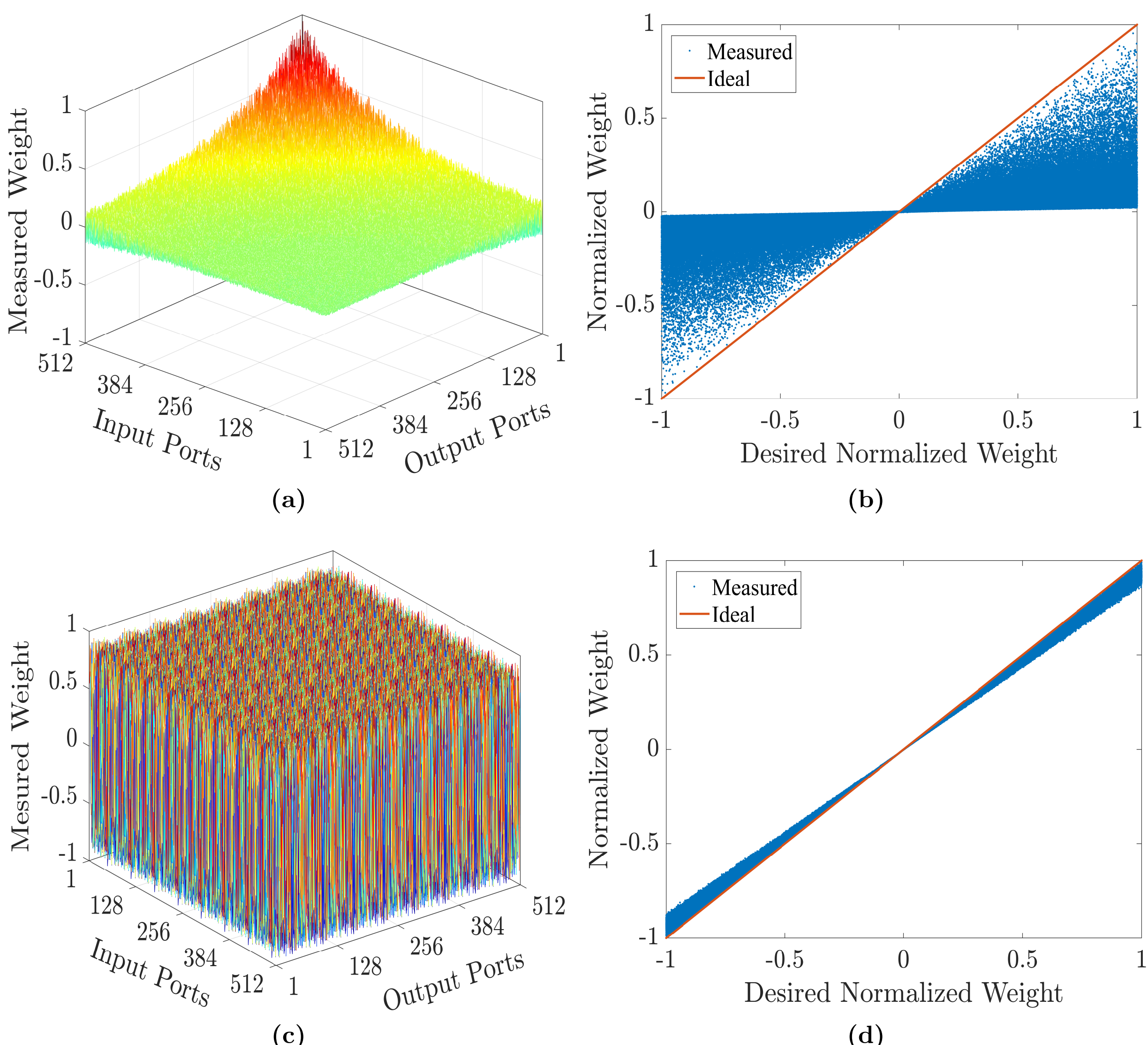


Figure 2.4: Effect of the wire resistance on the measured weights for 512 × 512 crossbar array at with 0.1Ω wire resistance. 3D plots of random weights distributed across the array, (a) without partitioning and (c) with partitioning into 64 × 64 crossbar arrays; and the measured weights with the sneak path problem versus the desired values for (b) the entire array without partitioning and (d) with partitioning.

[73] due to single cell reading. But, in neuromorphic applications, this adds an exponential behavior to MVM which becomes

$$S_j = \sum_{i=1}^{m} G_{ij} \sinh(aV_i). \tag{2.4}$$

This exponential non-linearity makes the MVM operation inaccurate which deteriorates the

training performance [74]. Some algorithms were developed to take the affect of the device's voltage-dependency into consideration while training non-spiking neural networks such as [74]. The same algorithm idea can be extended to the spiking neural networks.

Partitioning of Large Layer Matrices

The sneak path problem prohibits the implementation of large matrices using a single large crossbar array. One possible solution is to partition the large layer matrices into small matrices that can be implemented using realizable crossbar arrays. Figure 2.5 shows the partitioned crossbar arrays and the interconnect fabric between them to realize the complete MVMs where the large crossbar array, having $N \times M$ RRAMs, is partitioned into $n \times m$ crossbar arrays. In order to have the same structure of a large crossbar array, vertical and horizontal interconnects are placed under the crossbar arrays. This horizontal interconnect is used to connect the inputs between the crossbar arrays within the same array rows. The vertical interconnect is used to connect the outputs of the vertical crossbar arrays. The vertical interconnects are grounded through the sensing circuit to absorb the currents within the same vertical wire. The sensed currents are connected then to the neuronal activity. It is worth highlighting that each crossbar array may require input drivers (buffers) to reduce the loading effect of the vertical interconnect and crossbar arrays. These buffers are not shown in Fig. 2.5 for clarity. Moreover, they can be placed under the crossbar arrays to save the wafer area where the crossbar arrays are usually fabricated between higher metal layers. Fig. 2.4c shows the measured random synaptic weights with the same aforementioned parameters after partitioning the 512×512 crossbar arrays into 64 of 64×64 crossbar arrays. The weight variations due to their locations in the crossbar array became much smaller as shown in Fig. 2.4d and can be considered be modelled with the device variation.

Although, partitioning the array mitigates the sneak path problem, it might cause routing problems where the non-idealities (e.g parasitics) of the routing fabric will affect the perfor-

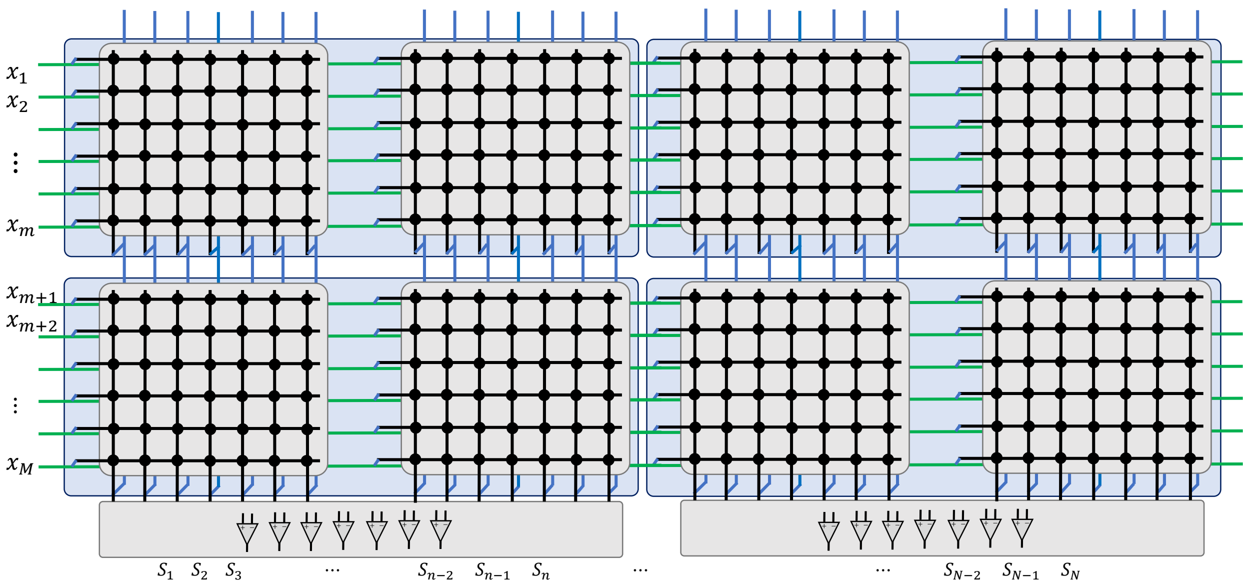


Figure 2.5: Realization of the partitioned matrices.

mance. Thus, routing's non-idealities must be simulated in the case of full-offline learning. Also, additional algorithmic work is needed to overcome the residual sneak path problem after partitioning (especially with the aforementioned high wire resistance expected to be 10Ω per cell), such as the mask technique proposed in binary neural networks [64]. In the mask technique, the exponentially decaying profile is used to capture the effect of the sneak path problem during learning by multiplying element-wise the trained weights.

2.4 Delay

Signal delay determines the speed at which computations can be run on hardware. While delays are not an issue for neuromorphic hardware designed to run with real-work time constants [16, 23], other models are accelerated [75]. Due to the parallel MVM operation, the memristive hardware would be dedicated for an accelerated regime. For this, it is necessary to reduce delay is caused by the device and structure parasitics and circuits.

In [72], a complete mathematical model for the crossbar delay is discussed. The model showed that the delay is a function of the weights stored in the crossbar arrays. The higher the device

resistance, the more delay to the signal. For $1M\Omega$ switching resistance, the maximum delay of the crossbar arrays is expected to be in range of nanoseconds. In addition, there is another delay resulting from the sensing circuit which is expected to be around 10ns.

The partitioning and the drivers add extra delay factors can be caused by the wire resistance of the interconnect fabric and the input capacitance of the drivers. The delay can be simply calculated using the Elmore delay model [72]. The wire resistance of the interconnect per array is nR_w where n is the number of columns per array and R_w is the wire resistance per cell. The Elmore delay of such an interconnect wire is $0.67nR_wC_d$, where C_d is the input capacitance of the buffer. Thus the total input delay is $0.67(N-n)R_wC_d + (N/n)\tau_d$, where N/n is the number of horizontal crossbar arrays and τ_d is the driver delay. The delay resulting from the partitioning and drivers is expected to be in range of nanoseconds. Thus, the total delay of the entire layer would be in range of $20n \sim 100n$ seconds. It is worth mentioning that the effect of the capacitive parasitics of the crossbar array is often ignored because the feature size of the fabricated devices is in the range of sub micrometers, i.e. $F = 200nm$, [76]. However, for nano-scale structures, i.e. $F = 10nm$, the capacitive parasitics may cause leakage paths at high frequency, where the impedance between the interconnects would be comparable to or less than the switching device's impedance, which would affect the performance. Thus, a more detailed analysis of the capacitive parasitics of the crossbar array must be considered on a case-to-case basis.

2.5 Asymmetric Non-linearity Conductance Update Model

Several RRAM devices demonstrating promising synaptic behaviors are characterized by nonlinear and asymmetric update dynamics, which is a major obstacle for large-scale deployment in neural networks [65], especially for learning tasks. Applying the vanilla backpropagation algorithms without taking into the consideration the device non-idealities does

not guarantee the convergence of the network. Thus, a closed form model for the device non-linearity must to be derived based on the device dynamics and added to the neural network training algorithm to guarantee the convergence to the optimal point (minimal error).

Most of the potentiation and depression behaviors have exponential dynamics versus the programming time or number of pulses. In practice, the depression curve has higher slope compared to the potentiation curve, which causes the asymmetric programming. The asymmetric non-linearity of the RRAM's conductance update can be fitted to the following model

$$G(t) = \begin{cases} G_{max} - \beta_P e^{-\alpha_1 \phi(t)} & v(t) > 0 \\ G_{min} + \beta_D e^{-\alpha_1 \phi(t)} & v(t) < 0 \end{cases} \tag{2.5}$$

where G_{max} and G_{min} are the maximum and minimum conductances respectively, $\alpha_1, \alpha_2, \beta_P$ and β_D are fitting coefficients. β_P and β_D are related to the difference between G_{max} and G_{min} and $\phi(t)$ is the time integral of the applied voltage.

Updating the RRAM conductance is commonly performed through positive/negative programming pulses for potentiation/depression with pulse width T and constant programming voltage V_p. As a result, the discrete values of the flux are $\phi(t = nT) = V_p nT$ where n is the number of applied pulses. This technique provides precise and accurate weight updates. For $t = n\Delta T$, and substituting back into (2.5), the potentiation and depression conductances become:

$$G_{LTP} = G_{max} - \beta_P e^{-\alpha_P n}, \text{ and} \tag{2.6}$$

$$G_{LTD} = G_{min} + \beta_D e^{-\alpha_D n}, \tag{2.7}$$

respectively, where n is the pulse number, $\alpha_P = |V_p|\alpha_1 T$ and $\alpha_D = |V_p|\alpha_2 T$. The rate of

change in conductance with respect to n becomes

$$\frac{dG}{dn} = \begin{cases} \beta_P \alpha_P e^{\alpha_P n}, \text{ for LTP} \\ -\beta_D \alpha_D e^{\alpha_D n}, \text{ for LTD} \end{cases}. \tag{2.8}$$

One way to quantify the device potentiation and depression asymmetry and linearity is the asymmetric non-linearity factors [77]. The effect of these factors are reflected in the coefficients $\alpha_P, \alpha_D, \beta_P$ and β_D which are used for the training. The potentiation asymmetric non-linearity (PANL) factor and depression asymmetric non-linearity (DANL) are defined as $PANL = G_{LTP}(N/2)/\Delta G - 0.5$ and $DANL = 0.5 - G_{LTD}(N/2)/\Delta G$, respectively, where N is the total number of pulses to fully potentiate the device. $PANL$ and $DANL$ are between $[0, 0.5]$. The sum of both potentiation and depression asymmetric non-linearities represents the total asymmetric non-linearity (ANL) which can be written as follows for the proposed RRAM model:

$$ANL = 1 - \frac{\beta_P e^{-0.5\alpha_P N} + \beta_D e^{-0.5\alpha_D N}}{\Delta G}. \tag{2.9}$$

Asymmetric Non-linearity Behavior Example

An example of a synaptic device is a non-filamentary (oxide switching) TiO_x based RRAM with a precision measured to 6 bits [11]. The $Mo/TiO_x/TiN$ device was fabricated based on a redox reaction at Mo/Tio_x interface which forms conducting MoO_x. This type of interface based switching devices exhibits good switching variability across the entire wafer and guarantees reproducibility [11]. The asymmetric nonlinear behavior of this device is shown in Fig. 2.7a.

The proposed model was fitted and parameters were extracted for the three programming cases $\{\pm 2V$, $\pm 2.5V$, and $\pm 3V\}$. Tables 2.1 and 2.2 show the extracted model identification

parameters of the device for the three reported voltages with negligible root mean square errors (RMSE). According to the results, the higher the applied voltage, the higher switching range. Clearly, the model parameters are function of the applied voltage. Thus, each parameter can be modeled as function of the applied voltage which would help to interpolate potentiation and depression curves if non-reported responses are required to be tested. The interpolated models are reported in the tables as function of the applied voltage.

Practically, $V_p = \pm 3V$ cases would be considered since it has the widest switching range. Figure 2.6 shows the curve fitted model on the top of the reported conductance for both potentiation and depression scenarios. This device has $PANL = 0.32$ and $DANL = 0.45$ with $ANL = 0.77$.

$V_p(V)$	G_{max} (nS)	$\alpha_P \times 10^{-3}$	$\beta_P \times 10^{-9}$	RMSE
3	674	30.58	626.8	9.07
2.5	252.7	18.23	220.22	0.6416
2	83.38	19.19	71.7	0.2276
V_p	$2.968e^{1.823V_p} - 30.4$	$2.019 \times 10^{-9}e^{7.51V_p} + 18.28$	$1.522e^{2.014V_p} - 13.78$	–

Table 2.1: Extracted Potentiation parameters of the $Mo/TiO_x/TiN$ device reported in[11].

$V_p(V)$	G_{max} (nS)	$\alpha_D \times 10^{-3}$	$\beta_D \times 10^{-9}$	RMSE
-3	32.95	353.4	921.9	23.696
-2.5	186.3	35.29	410.9	10.3215
-2	340.5	20.55	330.8	6.12
V_p	$307.6V_p + 955.5$	$8.14 \times 10^{-6}e^{-5.48V_p} + 20.5$	$0.009e^{-3.706V_p} + 315.9$	–

Table 2.2: Extracted depression parameters of the $Mo/TiO_x/TiN$ device reported in[11].

Device variations are an important issue to be taken into consideration during training. In RRAMs where there are two type of variations: (1) The variation during the write operation where a slightly different value is written in the device because of the randomness in the voltage variation and switching materials. This randomness can be mitigated with write-verify techniques where the written value is read to verify the value and corrected until the desired value is obtained [78]. (2) Independent device-to-device due to fabrication mismatch and material inhomogeneity. These variations can be included in the model by treating

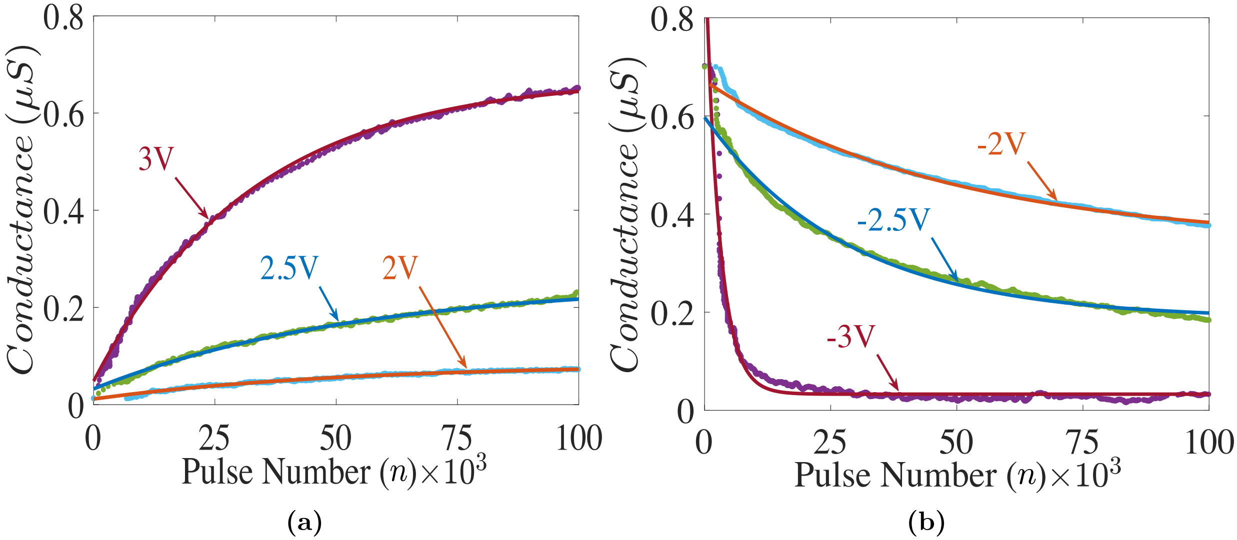


Figure 2.6: RRAM's conductance update (a) long term potentiation (b) long term depression.

each parameter in the model as a independent random variable. Figure 2.7b shows the conductance variations of multiple devices during the potentiation and depression cycles with $\pm 3V$ programming pulses. The model parameters are sampled from Gaussian sources with 25% tolerance (Variance/mean) for α, and 1% and 5% tolerances for the maximum and minimum conductances, respectively.

The effect of the variation in the parameter β is considered inside the variations of α. β is modeled as a lognormal variable to have a monotonic increasing or decreasing conductance update. Thus, the second term of the conductance update has log Gaussian variable, which is e^z, multiplied by $e^{\alpha n}$ where z and α are Gaussian variables. Since the sum of two Gaussian random variables is a Gaussian random variable, the variation of β and α can be included in either one of them.

2.6 Stuck-At Fault Effect

Stuck-At Fault (SAF) defects cause another inevitable problem that affects the accuracy results of the MVM which is the main operation in DNNs. SAF defects vary based on the

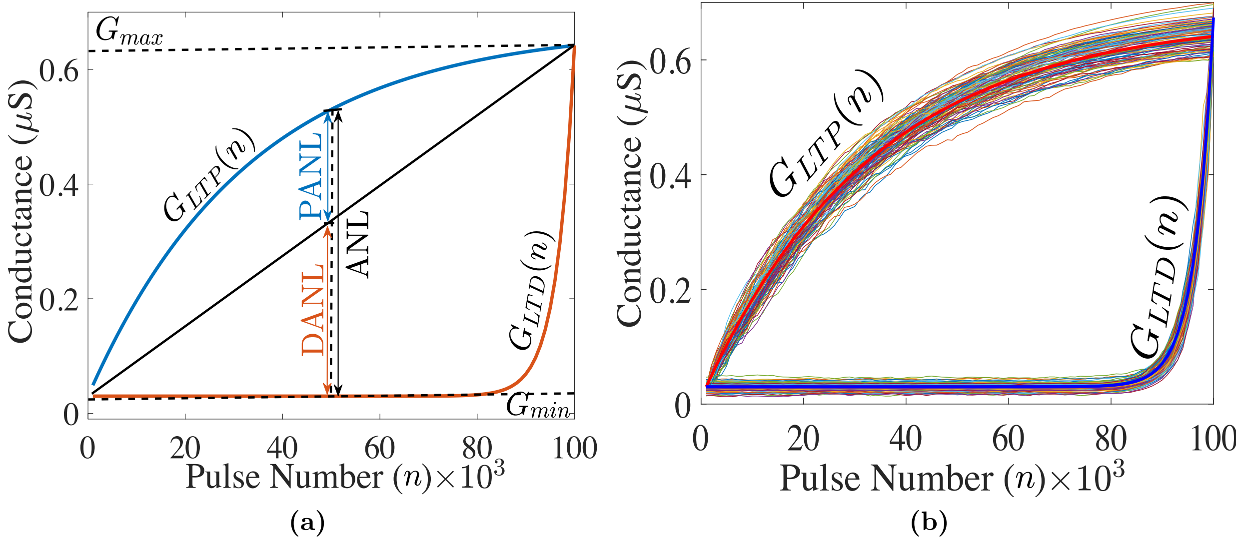


Figure 2.7: Non-idealities of the RRAM:(a) asymmetric nonlinear weight update (b) device Variations

fabrication technology and RRAM switching materials. In some recent works, the percentage of SAF fabricated crossbar arrays is about 10% for 1024 × 1024 for an in-house test array [79, 80] and is about 0.2% for 128 × 64 array (with only 15 devices stuck off) [81]. With the knowledge of the exact locations of the SAF devices, the network can be trained to isolate the SAF devices or at least mitigate their effect, however, this is not practical for DNNs where the trained weights are not designed for specific hardware. It should run without any knowledge of the location of SAF defects. Thus, in this work, we explore the effect of different SAF percentages on the recognition accuracy without retraining the network assuming that the SAF devices are randomly distributed in each crossbar array.

Chapter 3

Crossbar Array Modeling and Analysis

The chapter discusses the parasitics modeling of the crossbar structure to be included in the numerical simulators to have efficient and accurate calculations.

With the nanometer scale of memristors, interconnect modeling becomes very important due to the dominant effect that parasitics have in determining the overall performance of the system. Numerous research efforts have focused on means to analyze and model the effects of interconnect including voltage degradation, time delay, overshoot, crosstalk and coupling [82, 83, 84]. Prior work included only the effect of line resistance when analyzing crossbars [85, 86]. In this work, a closed form solution is proposed to provide designers with insights in the dynamic behavior of the system. The chapter focuses on creating a general framework that is simulator agnostic, while yielding closed form solutions for delay, as well as steady state solutions of the array. Thus, the main contribution of this work versus SPICE can be summarized in two points: a) the solution is closed form and not time stepped as in SPICE, yielding delay and steady state solution and b) the proposed framework provides a simulator agnostic approach to solving crossbars using any scripting language. While describing complex crossbars and their associated parasitics is possible in SPICE, it is highly time-consuming both in formulation and simulation time. Thus, the proposed model is very beneficial for many applications such as neural networks and pattern recognition, to get fast, accurate, and practical results. In addition, the derived Elmore delay

expression can be used to estimate delay of digital applications such as RRAMs.

3.1 Nano-scale Interconnect Parasitics Extraction

According to the ITRS, switching devices are expected to be in range of a few nanometers [25]. A crossbar is considered a nanostructure since the dimension of the wires and devices is less than $100nm$. As a result, nano-scale effects, such as grain boundaries, are inevitable and should be taken into consideration as they highly degrade system performance. In this section, we calculate wire parameters using the ANSYS Q3D extractor design tool to extract Resistance, Inductance, Capacitance and Conductance, (RLCG) circuit model. The extracted parameters discussed in this section will be used in our simulation to generate practical results based on realistic values. Two different feature sizes are considered; $F = 50nm$ cell which is a recently published switching device and $F = 5nm$ cell which is the projection of ITRS for resistive crossbar arrays [25]. Thus, the cell size is $4F^2$. In parasitics simulations, wire RLCG values are extracted for each feature size. Simulating a network of three parallel interconnect wires is enough to extract all parasitics including mutual and coupling parameters. In reality, an interconnect network will have more than three wires with mutual parasitics between all the wires, however, the mutual parasitics decreases significantly with increasing the distance between them. In our simulations, the three parallel wires are assumed to be constructed above a ground substrate and separated by silicon dioxide ($\epsilon_r = 3.9$ and electrical resistivity is $10^{16}\Omega.m$ [87] as shown in Fig. 4.1. Wire length is selected to be ten times greater than the cell length, $L = 20F$ to get practical results for a 10×10 interconnect array. Finally, we consider an operational frequency of 1 GHz, at which parameters are extracted. This frequency should be sufficient to cover most applications.

3.1.1 Interconnect Wire Resistance

Wire resistivity increases exponentially with technology scaling not only because of the reduction in minimum wire dimensions but also due to the increase in electron scattering at grain boundaries, surfaces, and interfaces as well. Conductivity is size dependent, especially with line width less than $100nm$ [88]. In our simulation, we used practical values for all parameters based on current literature [25, 89]. For $F = 50nm$, the wire resistivity is taken as $4.77\mu\Omega.cm$ which was used in fabrication. For the $F = 5nm$ cell, $45\mu\Omega.cm$ is used, which is calculated for copper nanocrystalline structure with the corresponding cross-sectional area $(50(nm)^2)$ [89]. It is worth noting that this value is much larger than the ITRS projection, which is expecting resistivity to be $11.41\mu\Omega.cm$ [25]. The wire resistance is given by

$$R = \frac{\rho l}{wh} \tag{3.1}$$

where ρ, l, w and h are the resistivity, wire length, wire width and wire height respectively. The width and the height of the wire are usually comparable and has aspect ratio, AR, which is $1 - 2$.

By performing 3D simulations for a unit cell, the wire resistance per cell was found to be

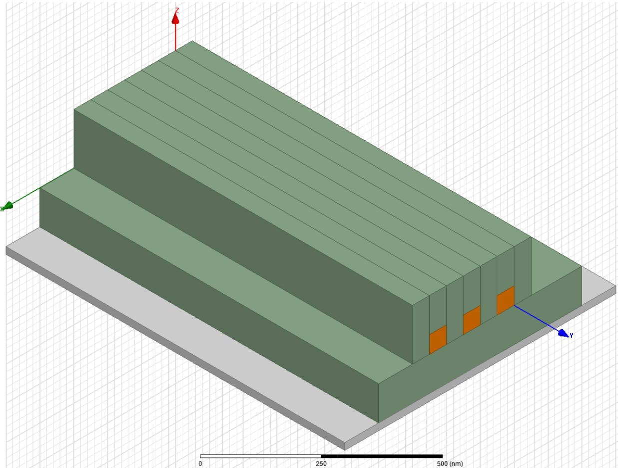


Figure 3.1: ANSYS Q3D simulation of three parallel interconnect wires.

1.908Ω for $F = 50nm$ case, where $w = h = 50nm$ and $l = 100nm$. On the other hand, the wire resistance per cell is 91.2Ω for $F = 5nm$ where $h = 2w = l = 10nm$. The cell resistance increased more than $47X$ despite of using double wire aspect ratio in the second case to decrease resistivity.

Skin effect causes an increase in the wire resistance with increasing the operating frequency which appears when the skin depth δ is smaller than the wire dimensions. The skin depth $\delta = \sqrt{\frac{2\rho}{\omega\mu}}$, where ω and μ are the operating frequency (rad/s) and the magnetic permeability of the wire, respectively. The skin effect appears after certain frequency. Estimating this cut off frequency can be done based on the fact that skin effect happens at the skin depth is less than the wire dimension. Thus, the skin effect cut off frequency is $f_{sk} \approx \frac{\rho}{\pi\mu h^2}$. The skin effect cut off frequencies are $1140THz$ and $4.833THz$ for $F = 5nm$ and $F = 50nm$, respectively. These values are very high and out of our range of interest. Thus, the skin effect can be ignored.

3.1.2 Interconnect Wire Inductance

Generally, any wire has self-inductance and mutual inductance with the surrounding wires. The self and mutual inductances of of a wire can be calculated using the following equation for $l \geq 10(h+w)$.

$$L_s = \frac{\mu_o l}{2\pi}\left[ln\left(\frac{2l}{h+w}\right) + 0.5 + 0.2235\left(\frac{h+w}{l}\right)\right],\ and \tag{3.2a}$$

$$L_m = \frac{\mu_o l}{2\pi}\left[ln\left(\frac{2l}{d}\right) - 1 + \left(\frac{d}{l}\right)\right], \tag{3.2b}$$

Table 3.1: Extracted parasitic inductance of three parallel interconnect wires at 1GHz

	Inductance/cell (fF)	
	$F = 50nm$	$F = 5nm$
L_{11}	70.3087	6.246
$L_{12} = L_{21} = L_{23} = L_{32}$	41.79	4.08
$L_{13} = L_{31}$	29.8	2.956

respectively, where l, h and w are the wire length, height and width, respectively. And, d is the distance between two wires. The parasitic extraction results of the inductance for three lines are summarized in table I for both structures. It worth noting that inductance decreased with decreasing the cell size.

3.1.3 Interconnect Capacitance

Within an interconnect network, there are two types of capacitances; stray capacitance and coupling capacitance as shown in Fig.3.2. The stray capacitance represents the capacitance between the interconnect wire and the reference plane. The position of the wire in the interconnect affects the value of the stray capacitance. For instance, for the wire in the middle of other two wires, it has two mutual coupling capacitances, one with each neighbor wire, C_c, and stray capacitance, C_s . The interconnect wires are so close to each other which in turn increases the coupling capacitance significantly. There are many coupling capacitance

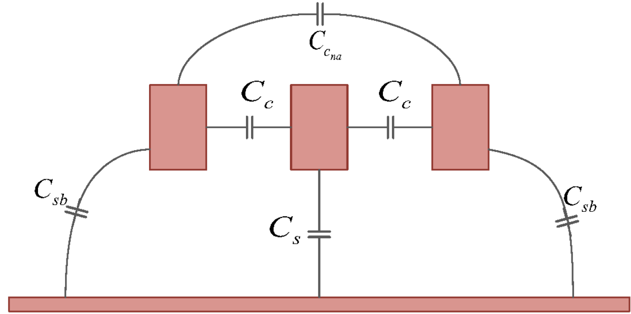


Figure 3.2: Capacitive model of interconnect.

Table 3.2: Extracted parasitic capacitance of three parallel interconnect wires at 1GHz

	Capacitance/cell (aF)	
	$F = 50nm$	$F = 5nm$
C_s	1.34	0.0624
C_{sb}	2.012	0.118
C_c	1.6	0.276
C_{na}	0.157	0.0376

sources such as parallel plate capacitance, fringing capacitance, and face-face capacitance. It is worth noting that there is also a coupling capacitance between non-adjacent wire as well and can be referred as $C_{c_{na}}$. These capacitance sources are analytically derived in details in [90]. All these factor, increases the coupling capacitance exponentially. On the other hand, if the wire is located at the boundary, it has one coupling capacitance and stray capacitance, C_{sb}. The boundary stray capacitance is greater than the stray capacitance of the wires located in the middle. This difference is referred as C_{wb} . In our simulation, silicon dioxide is used as an isolation between interconnect wires with 3.9 relative permittivity. The parasitic simulation results of the capacitance for three lines are summarized in table II for both structures. Practically, there are more than three wires in the interconnect. But as shown in the table, non-adjacent coupling capacitance is $7.2X$ less than the adjacent coupling between first and second wires. Thus, the coupling capacitance between first and fourth or higher positioned wires is much less than adjacent coupling capacitance.

3.1.4 Interconnect Conductance

For the sake of completeness, the conductance between two interconnect wires is calculated using parasitics extraction simulator. The conductance between the wire is a result of the conductivity of the isolation material between wires. High-k materials are widely used in MOS fabrication to reduce leakage. In our simulation, we used $10^{16}\Omega.m$ as the resistivity of

the isolation material. The conductance per unit cell is $1.77 \times 10^{-23}\Omega^{-1}$ and $3.03 \times 10^{-24}\Omega^{-1}$ for $F = 50$, and 5 feature size, respectively. In case of using a more conductive isolator, the conductance will increase.

3.1.5 Comparison and Discussion

Figure 3.3 shows the wire resistance and the reactance. For a large-size interconnect network, reactance cannot be ignored where the reactance becomes comparable to wire resistance. So, both wire resistance and inductance should be taken into consideration as shown in case of F=50nm. However, in nanostructure case, the wire resistance dominates the wire inductance and becomes the most important parasitic parameter to be taken into consideration. Generally, with decreasing wire dimensions, resistance increases and inductance increases. Thus, there is a need to define a cutoff frequency, below which, wire resistance dominates and above which, the wire inductance is comparable to wire resistance and should be considered in the analysis. For the purposes of this chapter, the cutoff frequency, f_{crl}, is defined as the frequency where the wire resistance is 100 times more the wire inductive reactance. Then, $f_{crl} = R/20\pi L$. The cutoff frequency was found to be $43.2GHz$ and $229THz$ for $F = 50nm$ and $5nm$, respectively. The cutoff frequency increased $5300X$. An analytic expression can be obtained by substituting by wire and inductance equations, the cutoff frequency can be given by (3.3) for $l = 10(h + w)$.

$$f_{crl} = \frac{279.96}{\sigma wh} \quad (THz) \tag{3.3}$$

where $\sigma(S/nm), w(nm)$ and $h(nm)$. The cutoff frequency decreases quadratically with increasing wire width. Also, as previously discussed, the line conductivity is function of the wire width as well. The cutoff frequency should be calculated for each wire width. On the other hand, as shown in Fig. 3.3, the skin effect appears at Tera Hertz frequency range which is as previously expected.

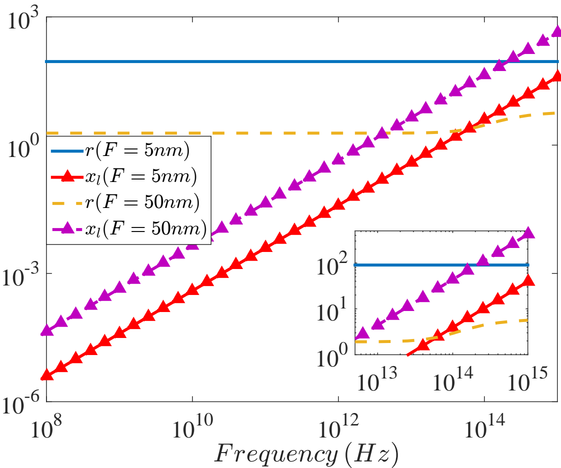


Figure 3.3: Resistance and inductance reactance per unit cell of interconnect wire.

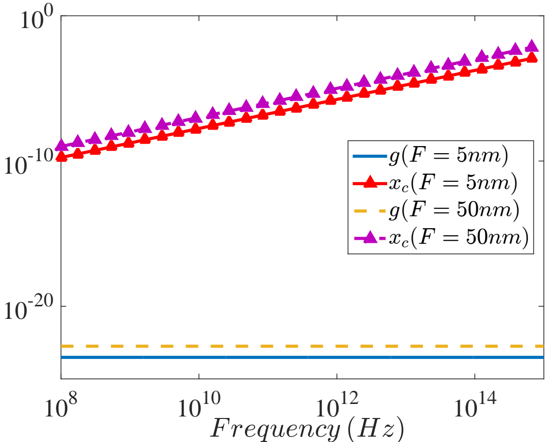


Figure 3.4: Conductance and capacitive susceptance per unit cell of interconnect wire.

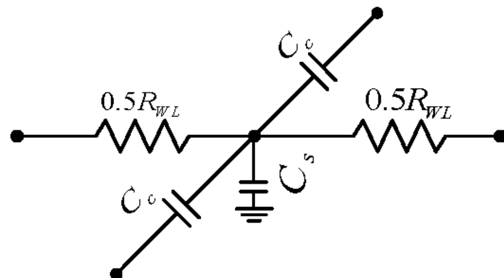


Figure 3.5: Summarized transmission line model.

Figure 3.4 shows the conductance and susceptance between two interconnect wires with changing frequency. As shown, the susceptance is much greater the conductance. Consequently, the conductance is neglected compared to susceptance.

Most modern memory and signal processing applications operate at a few GHz range for clock speed. At this frequency range, and as shown in Figs. 3.3 and 3.4, inductance can be neglected compared to resistance and conductance can be neglected compared to susceptance as well. In our analysis and simulations, wire resistance and capacitance are included. Also, the stray and coupling capacitance of adjacent lines are greater than coupling capacitance of nonadjacent wires. Thus, stray and adjacent coupling capacitance are included. In order to simply the model , the T model shown in Fig.3.5 is used so that the basic cell is symmetric and reciprocal. This model is used for both wire lines and bit lines of the crossbar arrays, in addition to the resistive switching device model.

3.2 Crossbar Array Architecture

A crossbar consists of perpendicularly intersected lines where the switching device is sandwiched at every intersection, as shown in Fig.1.1.

The crossbar array can be divided into cells where each cell contains a switching device. In order to get a full model for the crossbar, each individual switching cell should be modeled and then integrated into the architecture to get the full model. In the next section, the model of the switching cell is developed.

3.2.1 Switching Cell Structure and Circuit Model

At each intersection in the array, a switching device cell exists. This switching device has a certain impedance which is usually resistive which will be referred to as R_m. This resistance switches between two values: low-resistance state, LRS, and high-resistance state, HRS, typically referred to as R_{on} and R_{off} for some devices, such as memristors. Additionally, as shown in fig. 4.20, due to the electrodes of the device, the parasitic capacitance has

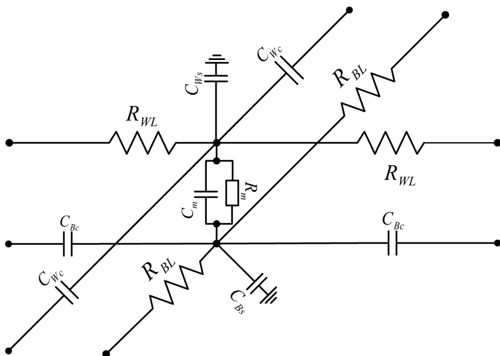


Figure 3.6: Basic cell model.

two components C_{m1} andC_{m2}. C_{m1} is adequately modeled as $C_{m1} = \epsilon_o \epsilon_r W^2/d$ where ϵ_o is permittivity of the free space, ϵ_r is the relative permittivity, W is the device cross section area, and d is the device width. In [91], a prototype memristor is presented where the parasitic capacitance of the memristor is approximately $100fF$ for TaO_2 memristor with $w = 10\mu m$ and $d = 7nm$. However, the recent fabricated memristors have nano dimensions so the switching device capacitance should scale accordingly. Thus, in our simulations, we will use $1fF$ and $0.1fF$ for $50nm$ and $5nm$ feature sizes as typical values. Furthermore, this device is sandwiched between two lines. The width of these lines is larger than the device width, and consequently, another capacitance across the device, C_{m2}, should be included into calculations. Then, the total impedance, which models the switching device, is $R_m//C_m$ where $C_m = C_{m1} + C_{m2}$.

In order to get full circuit model of a cell inside the crossbar, the wire model of word line and bit line are combined with the switching device model. Figure 4.20 shows the proposed cell model for the crossbar array where R_{WL}, and R_{BL} are the word and bit lines resistance, C_{Ws} and C_{Bs} are the stray capacitance of the word and bit lines, C_{Wc} and C_{Bc} are the coupling capacitance of the word and bit lines, and C_{Wsb} and C_{Bsb} are the boundary stray capacitance of the word and bit lines respectively. It is clear that the model is symmetric and reciprocal. In the next section, the mathematical model for the crossbar array is introduced.

3.3 Compact Mathematical Model for the crossbar array

In the previous section, the model of each node was discussed. By connecting all the nodes together, we form the full crossbar. This crossbar array has m rows (word lines (WLs)) and n columns (bit lines (BLs)) representing a 4 port network, as shown in Fig.4.21 where a port is composed of n or m lines which can be inputs or outputs. Consequently, we indicate the port voltages as $V_{app_WL1}$and V_{app_WL2} for word lines and V_{app_BL1} and V_{app_BL2} for the bit lines, as shown in Fig.3. The general case is considered in this model where the external voltages can be applied from one side or both sides of word lines and/or bit lines. the input nodes have an access resistance R_{s_WL} and R_{s_BL} for a word line and bit line, respectively.

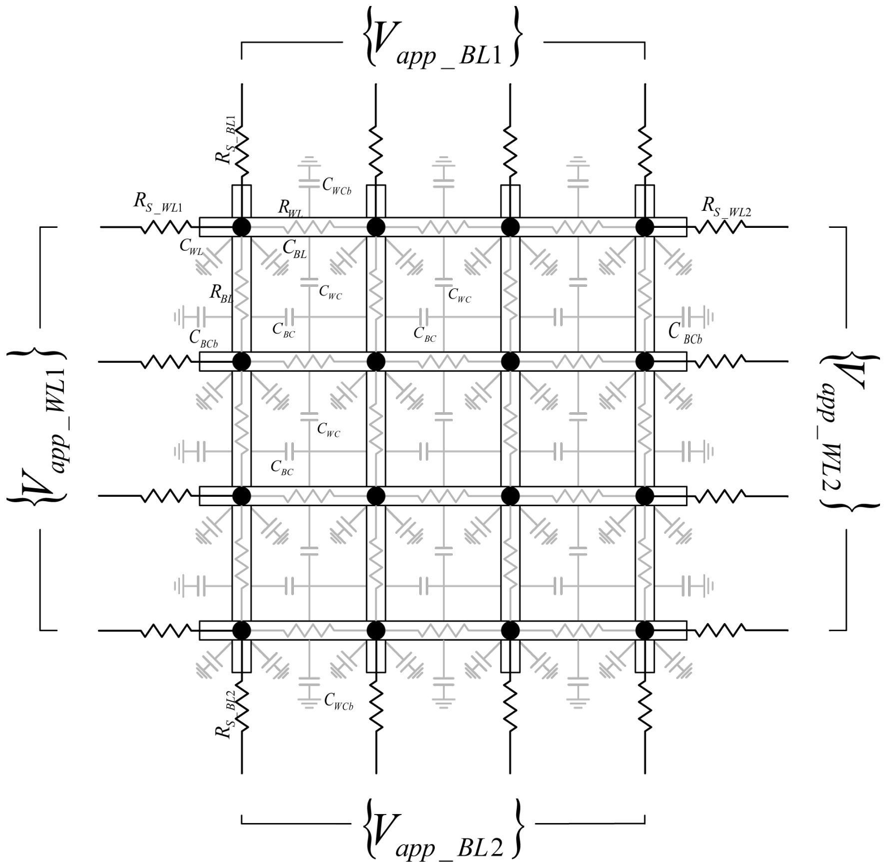


Figure 3.7: Circuit model of the crossbar array.

The access resistors are connected to the ports of the crossbar. In order to disconnect certain applied voltages, the corresponding access resistors can be set to ∞.

The electric model of the crossbar array should be described by the current flowing through the nodes or by the nodal voltages or both. The crossbar array has mn nodes so there are $2mn$ nodal voltages because each node has two voltages; one on the word line plane, $V_{WL}(i,j)$, and the other one is on the bit line plane, $V_{BL}(i,j)$, where $1 \leq i \leq m$, and $1 \leq j \leq n$. Thus, this model has $2mn$ unknown variables. In order to get a unique solution for this model, it is required to derive a system of $2mn$ independent equations.

Initially, in order to develop a complete model for the crossbar array, it is required to obtain the electrical model of the basic cell. Then, by repeating this model for the entire array and adding the model of the boundaries (ports), a complete and compact model for the crossbar array can be generated.

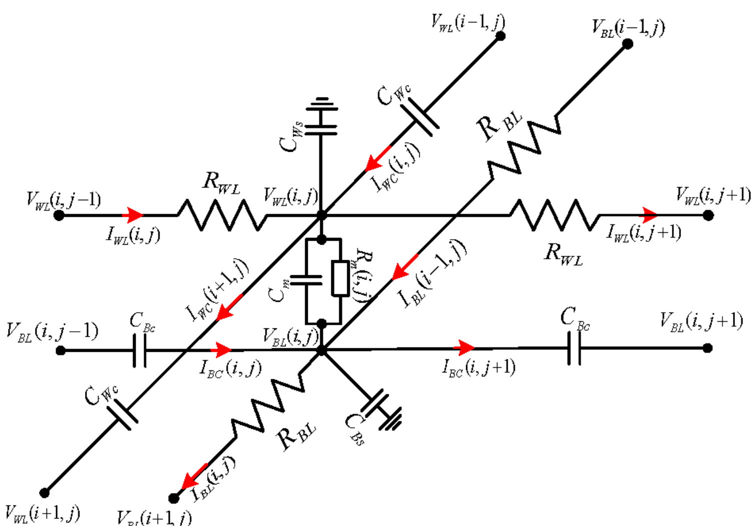


Figure 3.8: Kirchhoff's law of every cell in crossbar array.

3.3.1 Mathematical Model of Basic Switching Cell

By using Kirchhoff's Current Law at every node, two current equations exist for word line and bit line planes, as shown in Fig. 4.6. At node (i, j), the word line current flows from node $(i, j-1)$ then is divided to three paths; switching device (R_m// C_m), stray capacitance (C_{WL}) and next cell path. KCL of the WL is given as follows:

$$\begin{aligned} &I_{WL}(i,j) + I_{WC}(i,j) = I(i,j) + \\ &\quad I_{WL}(i,j+1) + C_{Ws}\frac{dV_{WL}(i,j)}{dt} + I_{WC}(i+1,j), \end{aligned} \tag{3.4}$$

where $I_{WL}(i,j)$, $I(i,j)$, $I_{WC}(i,j)$ and $V_{WL}(i,j)$ are the word line current entering the node (i, j), the current passing through the switching device at node (i, j), the coupling word line current entering the node (i, j), and word line voltage of the node (i, j), respectively. A similar equation can be derived for the bit line current by applying KCL for the bit line node. This equation is

$$I(i,j) + I_{BC}(i,j) + I_{BL}(i-1,j) = C_{Bs}\frac{dV_{BL}(i,j)}{dt} + I_{BC}(i,j+1) + I_{BL}(i,j) \tag{3.5}$$

For simplicity, we refer to $I(i, j)$ and $V(i, j)$ as $I_{i,j}$ and $V_{i,j}$, respectively. These KCL equations are describing the current passing through each branch. In this crossbar array, there are $2mn$ nodes where m and n are the crossbar dimensions. The nodal voltage analysis is shown in the supplementary materials.

3.3.2 Assembled Crossbar Array Model

The derived $2mn$ equations can be written in matrix form where there are parts in the equations; the nodal voltages, the derivative of the nodal voltages and the applied voltages.

The nodal voltages of the word lines and bit lines can be written as $2mn \times 1$ vector as follows:

$$\mathbf{V} = [V_{WL,1,1}, \ldots, V_{WL,1,n}, \ldots, V_{WL,m,1}, \ldots, V_{WL,m,n},$$
$$V_{BL,1,1}, \ \ldots, \ V_{BL,1,n}, \ldots, V_{BL,m,1}, \ldots, V_{BL,m,n}]^T \tag{3.6}$$

Also, the applied signals can be written as $2(m+n) \times 1$ vector $\mathbf{u}$ as follows:

$$\mathbf{u} = [V_{app\text{-}WL1}(1), \ldots, V_{app\text{-}WL1}(m), V_{app\text{-}WL2}(1), \ldots,$$
$$V_{app\text{-}WL2}(m), V_{app\text{-}BL1}(1), \ldots, V_{app\text{-}BL1}(n),$$
$$V_{app_BL2}(1), \ldots, V_{app_BL2}(n)]^T \tag{3.7}$$

The derived equations are 1st order differential equations which can be organized in a matrix form as follows:

$$\mathcal{M}\mathbf{V} + \mathcal{N}\dot{\mathbf{V}} = G\mathbf{u} \tag{3.8}$$

where $\mathcal{M}$ and $\mathcal{N}$ are $2mn \times 2mn$ matrices, $\dot{\mathbf{V}}$ is $2mn \times 1$ vector, G is $2mn \times 2(m+n)$ matrix. $\mathbf{V}$ and $\mathbf{u}$ are the nodal voltage and excitation input vectors. $\dot{\mathbf{V}}$ is the time derivative of $\mathbf{V}$.

Equation (5.10) can be further described by introducing a number of intermediate coefficients. $\mathcal{M}$, $\mathcal{N}$ and G represent the coefficients of the nodal voltages and are segmented as given in (3.9) as each segment is subjected to different nodal voltages. For instance $\mathcal{M}$ is segmented where $\mathcal{A}$ and $\mathcal{C}$ are the coefficients of the WLs nodal voltages, however, $\mathcal{B}$ and $\mathcal{D}$ are the coefficients of the BLs nodal voltages. The elements of each matrix is derived and is

shown in the supplementary materials.

$$\mathcal{M}= \begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{bmatrix}, \ \mathcal{N}= \begin{bmatrix} \mathcal{P} & \mathcal{Q} \\ \mathcal{R} & \mathcal{S} \end{bmatrix}, \ and G= \begin{bmatrix} G_{\mathbf{WL1}} G_{\mathbf{WL2}} & \mathbf{0} \\ \mathbf{0} & G_{\mathbf{BL1}} \quad G_{\mathbf{BL2}} \end{bmatrix} \tag{3.9}$$

In the case of applying low frequency inputs, the capacitive effect can be ignored in this model. Thus, the derivative term can be ignored by putting $\mathcal{N}=\mathbf{0}$, where only the resistive model for the crossbar can be obtained as previously derived in [85].

3.3.3 Crossbar Modeling Equations

By rearranging (5.10), the equation can be written as follows:

$$\dot{\mathbf{V}} = \mathcal{N}^{\mathbf{-1}} \left(-\mathcal{M}\mathbf{V}+G\mathbf{u}\right) \tag{3.10}$$

This equation represents the state equation of state-space representation for the crossbar array where $\mathbf{V}$ and $\mathbf{u}$ are state vector and input vector, respectively. $-\mathcal{N}^{-1}\mathcal{M}$ and $\mathcal{N}^{-1}G$ are state and input matrices, respectively. The output equation can be defined as

$$\mathbf{V_o} = \mathbf{\Psi V} + \mathbf{Wu} \tag{3.11}$$

where $\mathbf{V_o}$ is the output vector and is defined according to desired outputs. $\mathbf{\Psi}$ and $\mathbf{W}$ are output matrix and feedthrough matrix. In our case, it is required to calculate the nodal voltages then the output equation is $\mathbf{V_o}=\mathbf{V}$ where $\mathbf{\Psi}=\mathbf{I}_{2mn\times 2mn}$ and $\mathbf{W} = \mathbf{0}_{2mn\times 2mn}$ (no feedthrough path from inputs to outputs). Consequently, (5.11) and (5.12) represent the full state-space representation of the crossbar array.

Equations 5.11 and 5.12 model the crossbar array regardless of the nonlinearity of the switching devices. In a linear switching devices, the low resistance state and high resistance states are constant (or have minor non-linearity) with respect to the applied voltage [34, 92]. In this case, $\mathcal{M}$ is a constant matrix. Thus, it is possible to get a closed form solution as discussed in the next subsection. On the other hand, nonlinear devices, such as [93], have nonlinear low and high resistance states that are a function of the voltage. Within the crossbar, the resistance of the device is a function of the nodal voltages which vary depending on the position of the device in the crossbar. The proposed model can accommodate the nonlinearity as well, as will be discussed later in the chapter.

Linear Switching Devices based Crossbar Model

In this case, the proposed model of the crossbar array is mathematically well defined since it represents linear time invariant (LTI) control equations [94]. There are two methods to get the solution of the state-space representation: The first method is s-domain solution using Laplace transform where the state vector, $\mathbf{V}$, and the output vector, $\mathbf{V_o}$, can be written as follows:

$$\mathbf{V}\left(s\right)=\left(s\mathbf{I}+\mathcal{N}^{\mathbf{-1}}\mathcal{M}\right)^{\mathbf{-1}}\mathcal{N}^{\mathbf{-1}}\mathbf{Gu}\ \textit{and} \tag{3.12a}$$

$$\mathbf{V_o}\left(t\right)=\mathscr{L}^{-1}\left\{\mathbf{\Psi}\left(s\mathbf{I}+\mathcal{N}^{\mathbf{-1}}\mathcal{M}\right)^{\mathbf{-1}}\mathcal{N}^{\mathbf{-1}}G\mathbf{u}\right\}, \tag{3.12b}$$

respectively. The other method is the time domain method where the state vector can be written as:

$$\mathbf{V}\left(t,\mathbf{V}(0)\right)=\mathbf{e}^{-\mathcal{N}^{\mathbf{-1}}\mathcal{N}t}\mathbf{V}\left(0\right)+\int_0^t e^{-\mathcal{N}^{-1}\mathcal{M}(t-t)}\mathcal{N}^{-1}G\mathbf{u}\left(t\right)dt \tag{3.13}$$

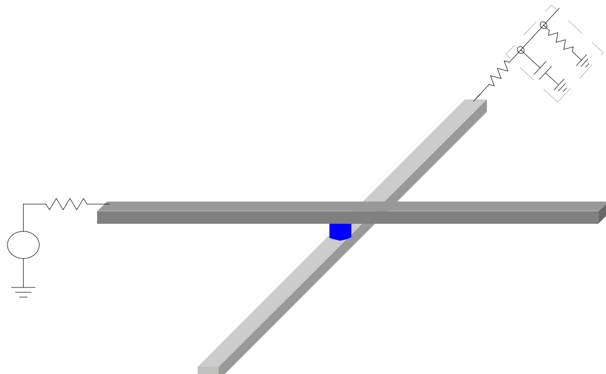


Figure 3.9: Interconnect array with loading

where $\mathbf{V}(0)$ represents the initial voltage vector. Consequently, the output vector is given by

$$\mathbf{V_o}=\mathbf{\Psi e}^{-\mathcal{N}^{-1}\mathcal{M}t}\mathbf{V}(\mathbf{0})+\mathbf{\Psi}\int_0^t e^{-\mathcal{N}^{-1}\mathcal{M}(t-t)}\mathcal{N}^{-1}G\mathbf{u}(t)\,dt \tag{3.14}$$

Alternatively, it can be written as in (3.15), with zero initial conditions,

$$\mathbf{V_o}(\mathbf{t})=\mathbf{\Psi}e^{-\mathcal{N}^{-1}\mathcal{M}(t)}\bigoplus\mathcal{N}^{-1}G\mathbf{u}(t) \tag{3.15}$$

where $\bigoplus$ is the convolution operator.

In reality, the most common example for the crossbar array is the interconnect where one of the WLs is the input and one of BLs is the output. In the next section, we discuss this scenario as a special case of the crossbar array.

Nonlinear switching based Crossbar model

The resistance of the nonlinear switching device is a function of the voltage across it. Parabolic and exponential shapes I-V behavior are typical in space-charge-limited-conduction and tunneling-based transport mechanisms, respectively [85]. Since there are many nonlinear

devices, each device has its own characteristics and behavior. A general model for nonlinearity of I-V characteristics can be defined as $I(V) = G_m(V) \times V$ where G_m is a nonlinear function of the voltage across the switching device and can be generally written as

$$G_m(V) = G_{m_o} + \sum_{i=1}^{\infty} G_{m_i} V^i \tag{3.16}$$

Substituting using the previous equation in the derived model, a system of of nonlinear equations is obtained and can be written as follows (see the supplementary materials for derivation details):

$$\mathcal{N}\dot{\mathbf{V}} = -\mathcal{M}_a\mathbf{V} + G\mathbf{u} - \sum_{i=1}^{\infty} \mathcal{M}_{b_i}(\mathbf{V} - T\mathbf{V})^{i+1} \tag{3.17}$$

$\mathcal{M}_a$ is linear coefficient matrix containing the transconductance of devices at $V = 0$ which is the same as the linear case. $\mathcal{M}_{b_i}$ is a nonlinear coefficient matrix containing coefficients of nonlinear terms of the switching devices. And, T is transformation rotational matrix of V.

In [85], the switching device nonlinearity is defined by $\eta = R(0)/R(V_{dd})$ where η is the nonlinearity ratio, $R(0)$ and $R(Vdd)$ are the resistances corresponding to zero and V_{dd} voltages across device. Substituting by (3.16), thus

$$\eta = \frac{G_m(V_{dd})}{G_m(0)} = 1 + \sum_{i=1}^{\infty} \frac{G_{m_i}}{G_{m_o}} V_{dd}^i \tag{3.18}$$

As an example, consider that the transconductance of switching devices has a parabolic nonlinearity where $G_m = a + b \times V^2$ for both LRS and HRS. consequently, the matrix form of the system can be written as follows:

$$\mathcal{N}\dot{\mathbf{V}} = -\mathcal{M}_{\mathbf{a}}\mathbf{V} - \mathcal{M}_{\mathbf{b}}(\mathbf{V} - T\mathbf{V})^3 + G\mathbf{u} \tag{3.19}$$

Then, the nonlinearity ratio is $\eta = 1 + bV_{dd}^2/a$. By specifying the nonlinearity ratio and resistance at V_{dd}, the transconductance can be obtained where $a = 1/(\eta R(V_{dd}))$ and $b = (\eta - 1)/(aV_{dd}^2)$.

Alternatively, consider that the switching devices has exponential I-V behavior where $I = I_o \times sinh(V/V_t)$ [95, 93], then the transconductance of the device is $G_m = dI/dV = (I_o/V_t) \times cosh(V/V_t)$ which can be expanded to

$$G_m = (I_o/V_t) \sum_{i=0}^{\infty} (V//V_t)^{2i}/(2n)! = (I_o/V_t) \times (1 + (V//V_t)^2/2! + (V//V_t)^4/4! + ...).$$

The transconductance can be approximated to $G_m \approx (I_o/V_t)(1 + (V//V_t)^2/2)$ where higher order terms are negligible for $V \leq V_t$. Thus, it is a special case of parabolic nonlinearity where $G_{m_o} = (I_o/V_t)$, $G_{m_2} = 0.5I_o/V_t^3)$ and the nonlinearity ratio is $\eta = 1 + 0.5(V_{dd}/V_t)^2$ which is 1.5 for $V_{dd} = V_t$.

3.4 Practical Interconnect

An interconnect network usually has one port input in the WL plane and one port output in the BL plane. Thus, we consider $\mathbf{V}_{app_WL1}$ as the input port and $\mathbf{V}_{app_BL1}$ as the output port. Moreover, $\mathbf{V}_{app_WL2}$ and $\mathbf{V}_{app_BL2}$ are disconnected so R_{s_WL2} and R_{s_BL2} should be open circuit i.e ∞ . Generally, the output port is connected to loading capacitor C_L and R_L and in this case, the access resistance R_{s_BL1} represents the wire resistance from the outputs of the interconnect to the load. The access resistor R_{s_WL1} represents the source resistance and the wire resistance from the source to the interconnect, as shown in Fig. 4.8.

The interconnect is a special case of the crossbar array so it has the same model but with taking into consideration disconnected ports and loading effects. Consequently, by substituting by R_{s_WL2}=$R_{s_BL2} = \infty$, the model of the interconnect is obtained. However, the loading effect is not included which will be discussed in the next subsection. Moreover, the

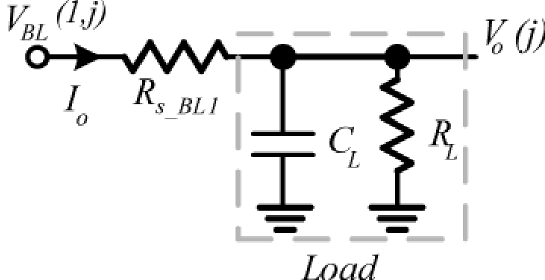


Figure 3.10: Kirchhoff's law at the load.

output voltage nodes, $\mathbf{V}_o$ which were $\mathbf{V}_{app_BL1}$, should be added to the system. thus, the nodal voltage vector can be written as

$$\begin{aligned}\mathbf{V} = [&V_{WL}(1,1)\,,\ \ldots,\ V_{WL}(1,n)\,,\ \ldots,\ V_{WL}(m,1)\,,\\ &\ldots,\ V_{WL}(m,n)\,,\ \ V_{BL}(1,1)\,,\ \ldots,\ V_{BL}(1,n)\,,\ldots,\\ &V_{BL}(m,1)\,,\ \ldots,\ V_{BL}(m,n)\,,\ V_o(1)\,,\ldots,V_o(n)]^T\end{aligned} \tag{3.20}$$

where $\mathbf{V}$ is the nodal voltage vector with size $(2m+1)n \times 1$. It is worth to note that, the output vector is included in (3.20) where the solution of the nodal voltages gives the output vector solution. Also, there is only one input port thus the input vector $\mathbf{u}$ is reduced to $m \times 1$ vector.

$$\mathbf{u} = \left[V_{app\text{-}WL1}(1)\,,\ \ldots,\ V_{app\text{-}WL1}(m)\right]^T \tag{3.21}$$

According to these changes, the size of matrices in the state-space representation is changed, for instance the system matrix size will be $(2m+1)n \times (2m+1)n$, and the size of input matrix will be $(2m+1)\,n \times m$. These changes are due to the loading effect.

3.4.1 Loading Effect On Crossbar Model

Practically the interconnect is connected to some load in order to read the data stored in the interconnect when used as memory [43, 44]. However, in some applications, such as neural networks [54, 55, 56], the outputs of the interconnect, where the input signals are scaled and mixed together with a certain pattern, are connected to the load to obtain the result of this mixing. This load may be resistive or capacitive depending on the application. In our analysis, we consider a general load which is a parallel RC circuit (R_L // C_L).

As previously mentioned, the load will affect the model so it should be included to get an accurate solution for the output signals of the interconnect. It is worth noting that the loading effect is added to the crossbar model regardless of the linearity of the devices and without affecting the nonlinear coefficients matrix, $\mathcal{M}_b$. Thus, we consider only the linear switching devices case. Figure 3.10 shows the load model at an arbitrary output node i.e. node j. Consequently, the KCL at the output node is $I_o = I_{R_L} + I_{C_L}$. Hence, the voltage form of the load can be written as follows:

$$V_o\left(j\right)\left(\frac{1}{R_{s_BL1}}+\frac{1}{R_L}\right)+C_L\frac{dV_o(j)}{dt}-\frac{V_{BL,1,j}}{R_{s_{BL1}}}=0 \tag{3.22}$$

where $V_o(j)$ is the output voltage at node j where $1 \leq j \leq n$.

The matrix form for the system is even by (5.10) where now M and N are square matrix of size $(2m+1)\,n \times (2m+1)\,n$ where n equations have been added to model the load. The modified M and N are given as follows:

$$\mathcal{M}=\begin{bmatrix}\mathbf{A} & \mathbf{B} & \mathbf{0}\\ \mathbf{C} & \mathbf{D} & \mathbf{H}\\ \mathbf{0} & \mathbf{a}\ \ \mathbf{0} & \mathbf{b}\end{bmatrix},\ \mathcal{N}=\begin{bmatrix}\mathbf{P} & \mathbf{Q} & \\ \mathbf{R} & \mathbf{S} & \mathbf{0}\\ & \mathbf{0} & \mathbf{C_L I}_{n\times n}\end{bmatrix} \tag{3.23}$$

where **A**, **B**, **C**, **D**, **P**, **Q**, **R** and **S** have the same values mentioned in the previous section. The added terms **a** and **b** are $n \times n$ matrices which are given by (3.24a)), but **H** is $mn \times n$ matrix and is given by (3.24c). Besides, the input matrix, G, is simplified to $(2m+1)n \times m$ matrix which is G=$[G_{\mathbf{WL1}} \ \ \mathbf{0}]^T$ where $G_{\mathbf{WL1}}$ is $nm \times m$ matrix and is given by (3.24c).

$$\mathbf{a} = -\frac{1}{R_{s_{BL1}}}\ I_{nxn}, \quad \mathbf{b} = \left(\frac{1}{R_{s_{BL1}}} + \frac{1}{R_L}\right)\ I_{nxn} \tag{3.24a}$$

$$\mathbf{H}(j,j) = -\frac{1}{R_{sBL1}} \quad 1 \leq j \leq n \tag{3.24b}$$

$$\mathbf{G}_{WL1}((i-1)n+1, i) = \frac{1}{R_{sWL1}\,(i)} \quad 1 \leq i \leq m \tag{3.24c}$$

3.4.2 Solution of Interconnect Model

The interconnect is a special case of the crossbar array so it is governed by the same modeling equations. However, in the case of the interconnect, the output matrix can be defined since the solution of the output voltages is included in the state vector, **V**, and also there is no direct connection to the output. As a result, the feedthrough matrix $\mathbf{W}_{n\times m} = \mathbf{0}$. Consequently, the output voltage is $\mathbf{V_o}$=$\mathbf{\Psi V}$ where $\mathbf{\Psi}$ is $n \times (2m+1)n$ matrix to select the output voltage from the state vector where $\mathbf{\Psi}$= $[\mathbf{0_{n\times 2m}}{}_n \ \mathbf{I_{n\times n}}\ ]$. Thus, the output voltage is given by (3.14).

In case of nonlinear switching devices, the obtained system is first order nonlinear differential equations and can be solved using numerical techniques such Euler and Runge Kutta methods. In this work, we used ordinary differential equations toolbox in MATLAB to solve the system.

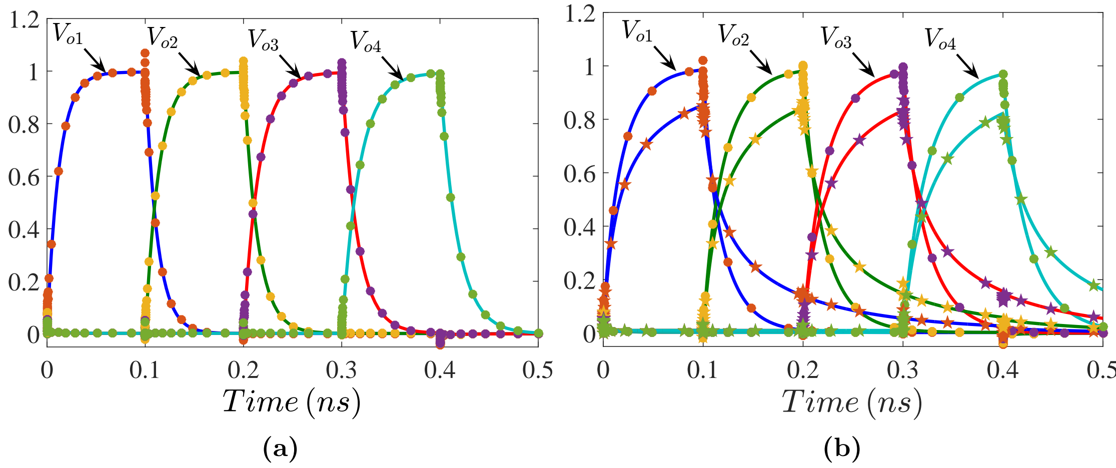


Figure 3.11: Transient verification between circuit simulation (dotted line) and analytical solution (solid lines) for 4×4 interconnect with (a) linear switching devices, and (b) parabolic switching devices.

3.4.3 Simulation Results and Validation

Simulations were performed using the parasitic values which are extracted for $5nm$ feature sizes. We chose this feature size case, which has very small values, to prove that the model perfectly matches the HSPICE simulations. In our simulation, the switching device has resistance, R_m, which switches between $1K\Omega$ and $1M\Omega$ at $1V$ reading voltage as mentioned in [27, 91]. Practically, the reading crossbar sensing circuits have capacitive input impedance, such as sensing circuit of the interconnect [96]. Thus, capacitive loading is considered as $C_L = 10fF$ and $R_L = 10^{18}\Omega$.

In order to confirm the validity of the model solution, we simulated 4×4 crossbar array using HSPICE and compared the results with MATLAB numerical simulation results based on the proposed model for both linear switching devices case (Fig. 3.11a) and nonlinear switching devices case (Fig. 3.11b). In the nonlinear case, we simulated two scenarios for nonlinearity of LRS and HRS (η_{LRS}, η_{HRS}), which are case I $(2, 10)$ and case II $(10, 100)$. Figure 3.11 shows the transient verification between circuit simulation and the proposed analytical solution showing perfect matching where 1V pulse signal is applied to each input while the other

inputs are grounded. Figure 3.11b shows the output response for two nonlinearity cases, where the curves with stars/bullets have higher/lower nonlinearity (case II/case I). It is clear that the delay increases with increasing nonlinearity.

3.5 Elmore's Delay Extension for Crossbar Arrays

In this section, we are introducing a novel closed form expression for the delay in the crossbar arrays for the first time. Obtaining closed from expression is beneficial in modeling the transient behavior of the crossbar. The delay from the input to the output determines the maximum frequency of operation which is important to determine assuming the crossbar used either as a memory block or for connecting processing blocks as a crossbar switch. Thus, it is essential to find closed form expressions for delay. In order to do that we consider the linear switching devices case in the analysis.

3.5.1 Novel Mathematical Modeling of The Delay

One of the most efficient techniques to calculate delay in RC networks is Elmore's delay [97, 98]. The importance of Elmore's delay comes from its mathematical robust definition which ca n be easily transformed to s-domain. The delay time, T_D, is defined as follows:

$$T_D = \int_0^\infty t y'(t)\, dt \tag{3.25}$$

where $y'(t)$ is the derivative of the transient response due to the step input signal. Elmore delay is defined based on the observation of the monotonic increase of the response of the signal, $y(t)$. The delay time, T_D, is the time of maximum point of $y'(t)$ representing the first-order moment. It is worth to note that Elmore delay is defined for zero initial conditions which is suitable to our case. In [98], expressions for signal delay in general RC networks

are developed but these expressions are not valid in our case because they are derived based on the assumption that output responses reach unity as time tends to infinity which does not happen in our case. In order to get the final value for the output signals, the final value theorem is applied to the output voltage in s-domain, given by (3.12), such that

$$\mathbf{V_o}\left(\infty\right)=\lim_{s\to 0} s\mathbf{V_o}=\lim_{s\to 0} s\mathbf{\Psi}\left(s\mathbf{I}+\mathcal{N}^{\mathbf{-1}}\mathcal{M}\right)^{\mathbf{-1}}\mathcal{N}^{\mathbf{-1}}G\mathbf{u}(s) \tag{3.26}$$

By using power series expansion, the inverse bracket can be given by

$$\left(s\mathbf{I}+\mathcal{N}^{\mathbf{-1}}\mathcal{M}\right)^{\mathbf{-1}}=\left[\mathbf{I}-s\mathcal{M}^{\mathbf{-1}}\mathcal{N}+s^2\mathcal{M}^{\mathbf{-2}}\mathcal{N}^{\mathbf{2}}-\ldots\right]\mathcal{M}^{\mathbf{-1}}\mathcal{N} \tag{3.27}$$

It is worth noting that this series is expanded by this way in order to guarantee convergence where the eigenvalues of $\mathcal{N}^{\mathbf{-1}}\mathcal{M}$ are greater than unity. By substituting into (3.26), and taking the limit $s \to 0$, the final output voltages are

$$\mathbf{V_o}\left(\infty\right) = \mathbf{\Psi}\mathcal{M}^{\mathbf{-1}}G\boldsymbol{\delta} \tag{3.28}$$

where $\boldsymbol{\delta}$ is $m \times 1$ coefficients vector of input vector, $\mathbf{u}$. Thus, the final value of each output voltage is a function of the crossbar parameters.

By applying Laplace transform to (3.25), the s-domain version of Elmore's delay is given by

$$T_D = \lim_{s\to 0} -\frac{d}{ds}sY(s) \tag{3.29}$$

where $Y\left(s\right)$ is the Laplace transform of $y(t)$ which represents V_o in our case. Consequently,

the delay time for the output vector can be obtained by

$$\mathbf{T_D}=\lim_{s\to 0} -\frac{d}{ds}s\mathbf{V_o}=\lim_{s\to 0}-\frac{d}{ds}\left[s\mathbf{\Psi}\left(s\mathbf{I}+\mathcal{N}^{-\mathbf{1}}\mathcal{M}\right)^{-\mathbf{1}}\mathcal{N}^{-\mathbf{1}}G\mathbf{u}\right] \tag{3.30}$$

Substituting by (3.27) and taking the limit, the delay can be reduced to be given by

$$\mathbf{T_D} = \mathbf{\Psi}\mathcal{M}^{-\mathbf{1}}\mathcal{N}\mathcal{M}^{-\mathbf{1}}G\boldsymbol{\delta} \tag{3.31}$$

3.5.2 Delay Verification and Results

In order to verify the derived expression for the Elmore delay in the crossbar arrays, the maximum delay obtained from Elmore expression is compared with numerical values (using the previous derived model in section IV) and HSPICE simulations. Fig. 3.12 shows this comparison where two extreme cases were selected to get the maximum possible delay. These cases are when all the array has low resistance state (LRS) as shown in Fig. 3.12a. The second case is when all the array has high resistance state as shown in Fig. 3.12b. Two cases were chosen to compare with Elmore expression; the first one when the signal reaches 63% from its final value which corresponds to one time constant (τ) for the basic series RC circuit which shows perfect matching with the derived Elmore expression. On the other hand, we also compared with the case when the signal reaches 90% of its final value representing the rising time.

This comparison, shown in Fig. 3.12, is performed for $5nm$ feature size architecture with $HRS = 1M\Omega$, LRS= $1K\Omega$ and $C_m = 0.1fF$. A $10fF$ loading capacitor was assumed to model the input capacitance of the sensing circuit [96]. It is clear from this comparison that HSPICE simulation and the MATLAB simulation based on the derived model are matching, thus validating the model.

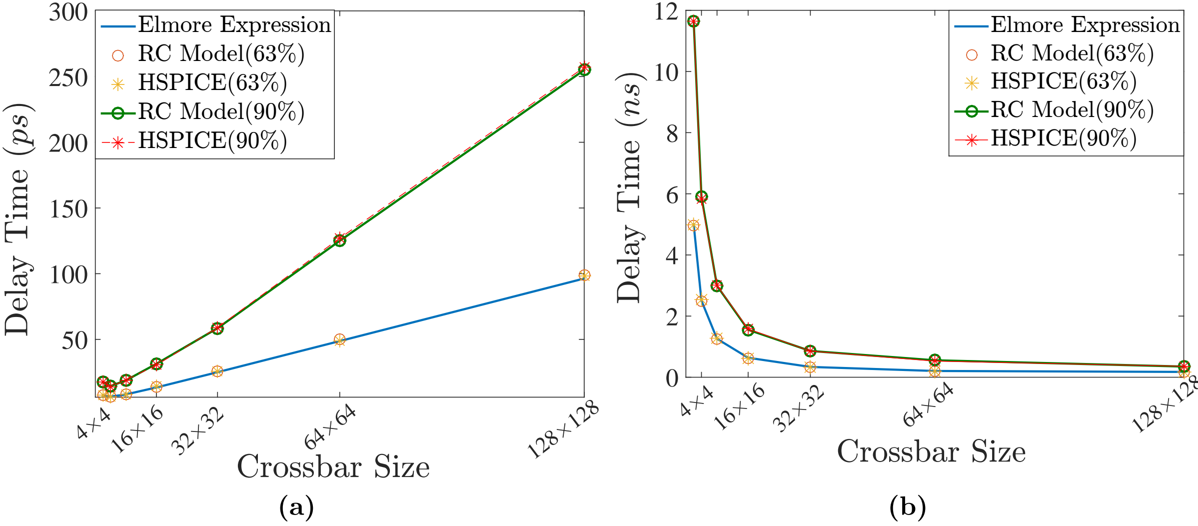


Figure 3.12: Comparison with the calculated Elmore delay expression and the transient simulation results for a) all HRS, and b) all LRS.

3.6 Discussion and Comparison

In some applications especially RRAMs, the inputs of the crossbar arrays may be disconnected which means that the inputs terminals are floating. As a result, the structure of the crossbar is changed and the delay can be calculated using the same derived expression by putting the access resistors to ∞ (effectively disconnecting some inputs). As shown in Fig. 3.13, the delay of the grounded inputs decreases with increasing the size of the crossbar, due to the increase in the number of parallel branches, to a point where the line resistance dominates and starts to increase. However, in the floating case, the delay monotonically increases with increasing the crossbar size.

In case of grounded input configuration, shown in Fig.3.13, the delay curve exhibits a valley curve since for the low size crossbar, the switching device resistance is the dominant resistance, thus the delay decreases with increasing the crossbar size. However, with increasing the crossbar size, the line resistance increases and dominates the total resistance of switching devices (For instance, at 128, the total switching device resistance is LRS/128=7.8Ω but total line resistance is around $128R_{WL} = 244.4\Omega$ and $11.67k\Omega$ for $50nm$ and $5nm$ feature

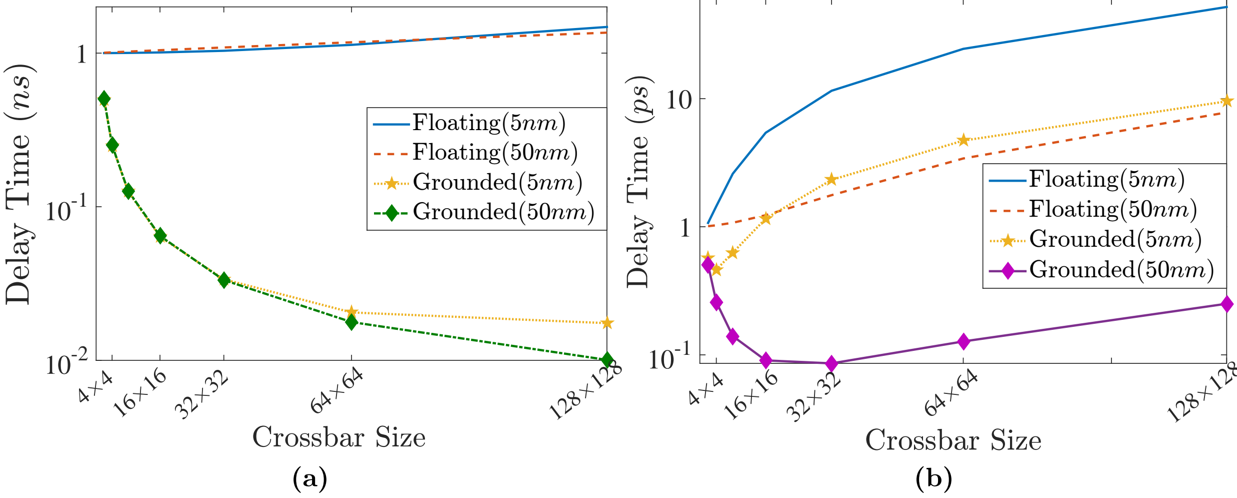


Figure 3.13: Comparison between the calculated delay of floating and grounded inputs for a) all HRS, and b) all LRS.

size). And as a result, the delay increases.

The minimal delay point of a square crossbar array can be approximated as $N_{minD} \approx \sqrt{\frac{R_m}{r_l}}$, where N_{minD}, R_m and r_l are the length of the crossbar corresponding to minimum delay, switching device resistance, and line resistance, respectively. For example, in case of $Rm = 1k\Omega$ and $r_l = 1.92\Omega$ (50nm case), $N_{minD} \approx 22.8$ but at $r_l = 91.2$ (5nm case), then $N_{minD} \approx 3.3$ which matches the obtained results from the simulations (Fig.3.12b). On other hand, at $R_m = 1M\Omega$, and $r_l = 1.92\Omega$ (50nm case), $N_{minD} \approx 722$ but at $r_l = 91.2$ (5nm case), then $N_{minD} \approx 105$. This is clear from Fig. 3.12a where the delay curve started to increase for 5nm case and still decreasing for 50nm case. It is worth noting that the switching device capacitance does not affect the delay since the delay is dominated by the grounded capacitances which is dominated by the loading capacitance.

In all applications, the crossbar array contains randomly distributed LRS and HRS. Thus, delay is a function of the stored data. Figure 3.14 shows the effect of randomly distributed data in the crossbar with 50%, 75% and 90% HRS for the grounded inputs case. Three cases lie between the maximum and minimum delay representing all HRS and LRS, respectively. By increasing the size of the crossbar, the delay tends to the minimum delay curve. But, for

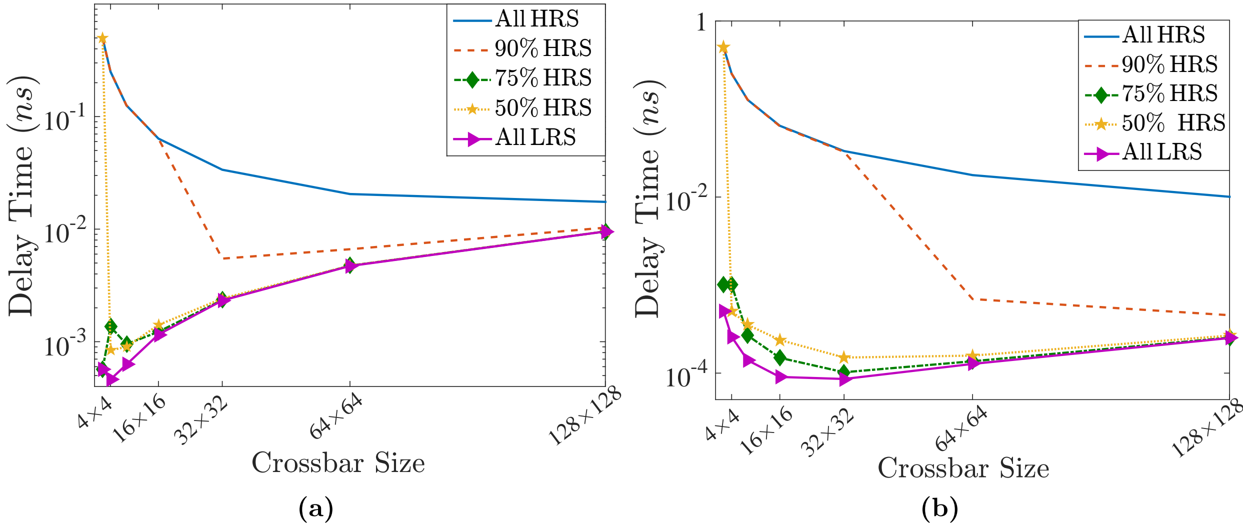


Figure 3.14: Effect of seeding the crossbar array with random data for different cases of (a)$F = 5nm$ structure and (b) $F = 50nm$ structure.

the case of the floating input, there is no path to ground so the delay depends on the data stored in the crossbar array which is random.

Figures 3.15 and 3.16 show the effect of loading capacitance for floating and grounded input crossbars for $5nm$ and $50nm$, respectively. Clearly, the delay increases linearly with increasing the loading capacitance. In case of no loading, the crossbar array is self loaded by the parasitics which is the limit for the minimum delay. This limit increases with increasing the size of the crossbar array. In case the loading capacitance is much larger than parasitic capacitances, the delay is dominated by this load as clear from Figs. 3.15 and 3.16.

Moreover, the position of the applied signal has a minor effect on delay for the floating case. The first and last inputs have the same delay approximately, since there is no path to the ground except through the output ports, thus the whole crossbar is loaded to the output. But, in the grounded case, each output has its own delay because each output has different attached resistance and capacitance.

Lastly, we consider the effect of the switching resistances as there are many types of switching devices. Each device has its own LRS and HRS. As noticed earlier, the delay increases

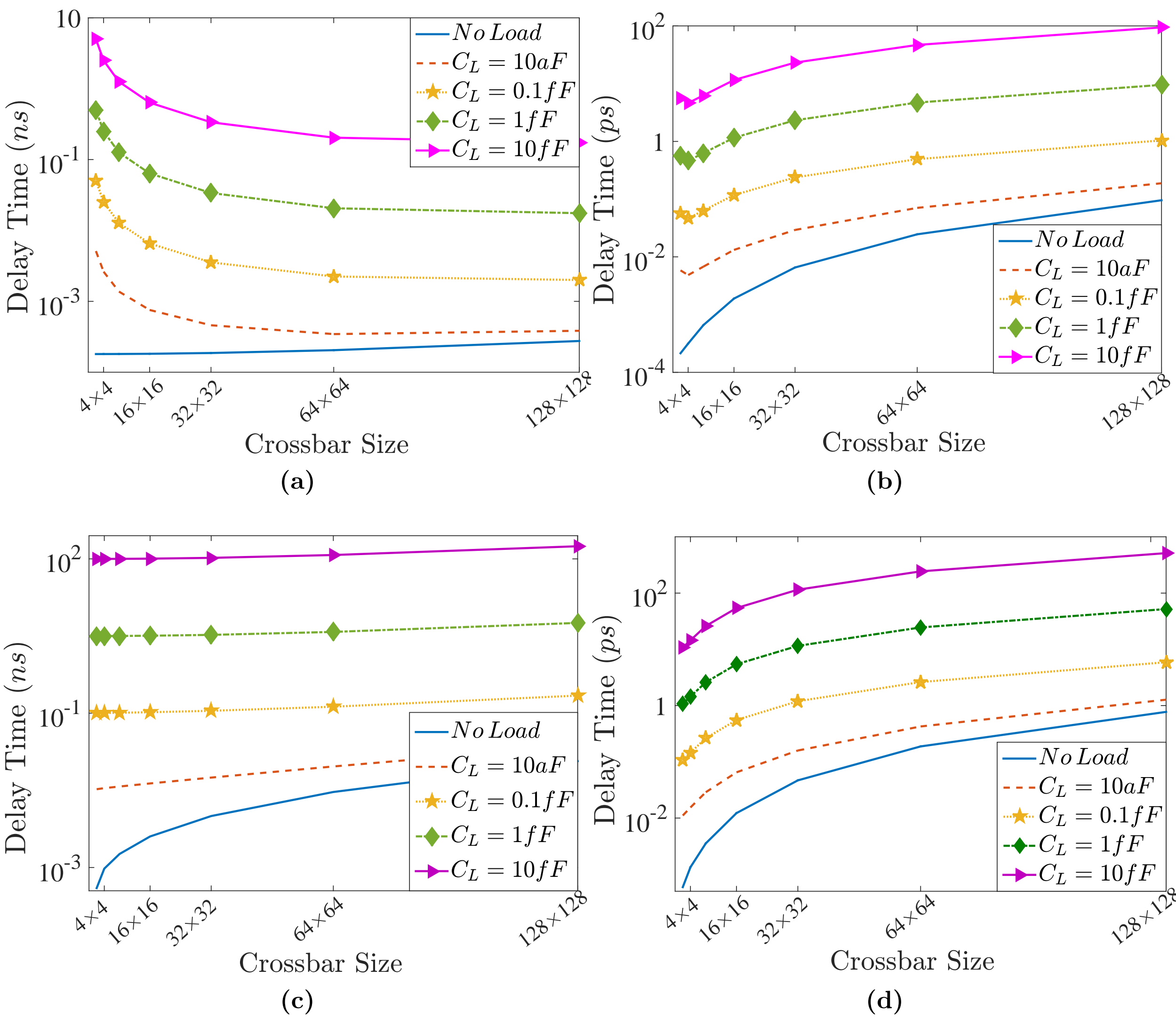


Figure 3.15: Effect of changing loading capacitance for grounded inputs of $F = 5nm$ for (a) all HRS and (b) all LRS, and floating inputs for (c)all HRS, and (d) all LRS

with increasing the line resistance after N_{minD} for the grounded input crossbar. Thus, selecting a device with a different LRS, gives another N_{Dmin}. Figure 3.17 shows the effect of changing LRS representing different devices. By increasing LRS, N_{minD} increases but the delay increases as well.

As previously shown in the transient simulation, the nonlinearity of the switching devices affects the delay. Figure. 3.18 shows the effect of the nonlinearity of the switching devices on the delay for different crossbar arrays compared to the linear case (Elmore delay). The

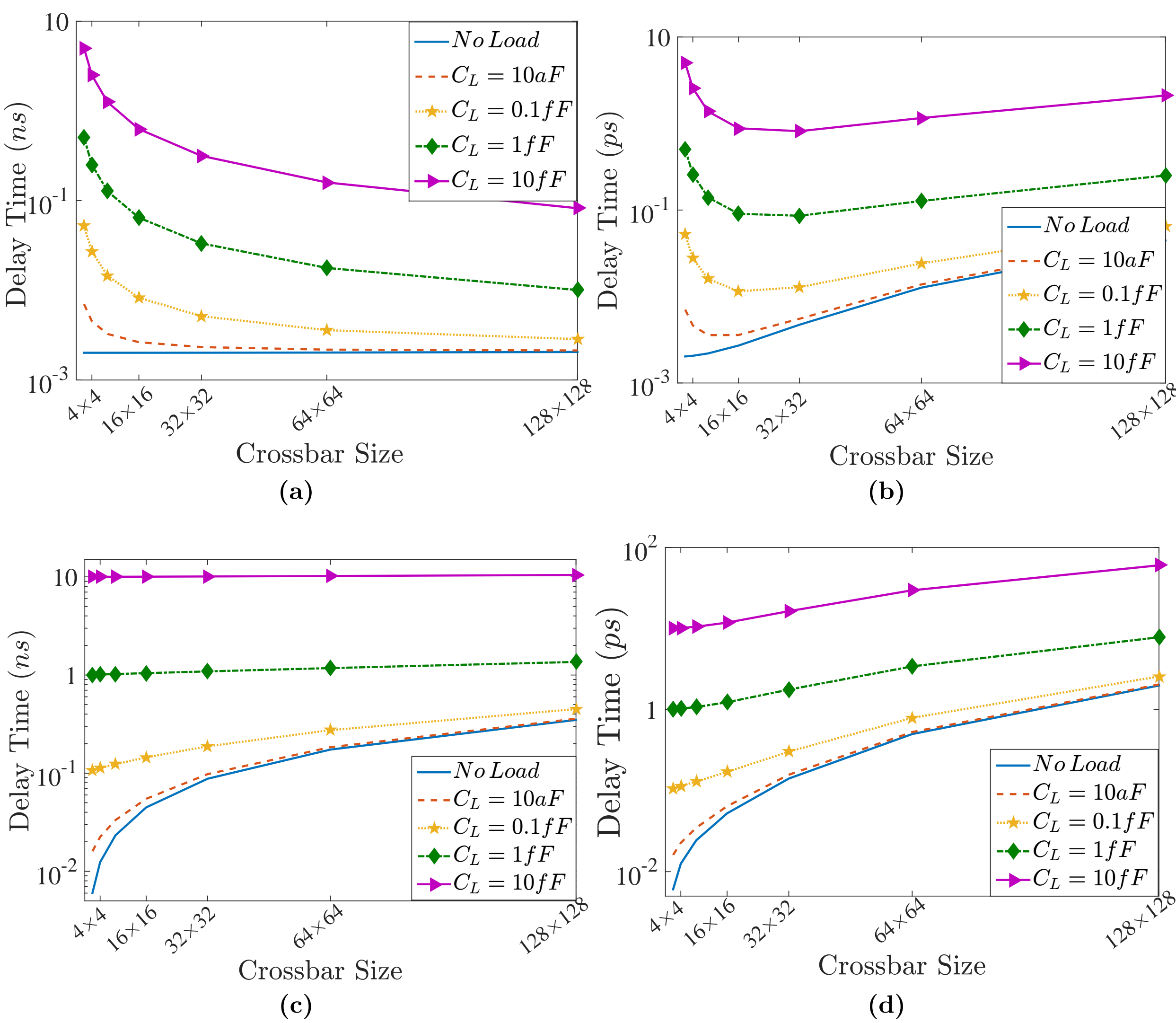


Figure 3.16: Effect of changing loading capacitance for grounded inputs of $F = 50nm$ for (a) all HRS and (b) all LRS, and floating inputs for (c)all HRS, and (d) all LRS

dashed lines are HSPICE simulations, and diamonds and stars represent the results of the proposed model (3.17) which match HSPICE simulations. For all HRS case (Fig.3.18b), the more nonlinearity, the higher the delay, as well as a reduction in the minimum point N_{minD}. However, in all LRS case (Fig.3.18b), the delay curves are around Elmore delay since LRS nonlinearity is not high, unlike the HRS nonlinearity. It worth noting that due to the nonlinear behavior of the devices, the delay will be totally random and may be more than the high resistance state case.

As an exmple for a comparsion with SPICE, if it is required to train a single layer neural

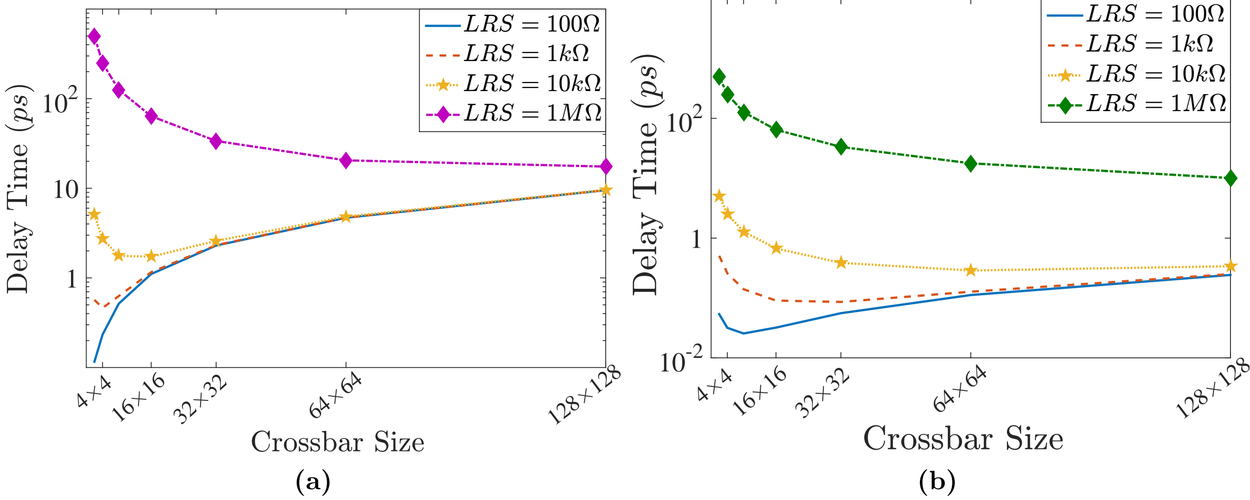


Figure 3.17: Effect of using different devices for (a) $F = 5nm$ and (b)$F = 50nm$.

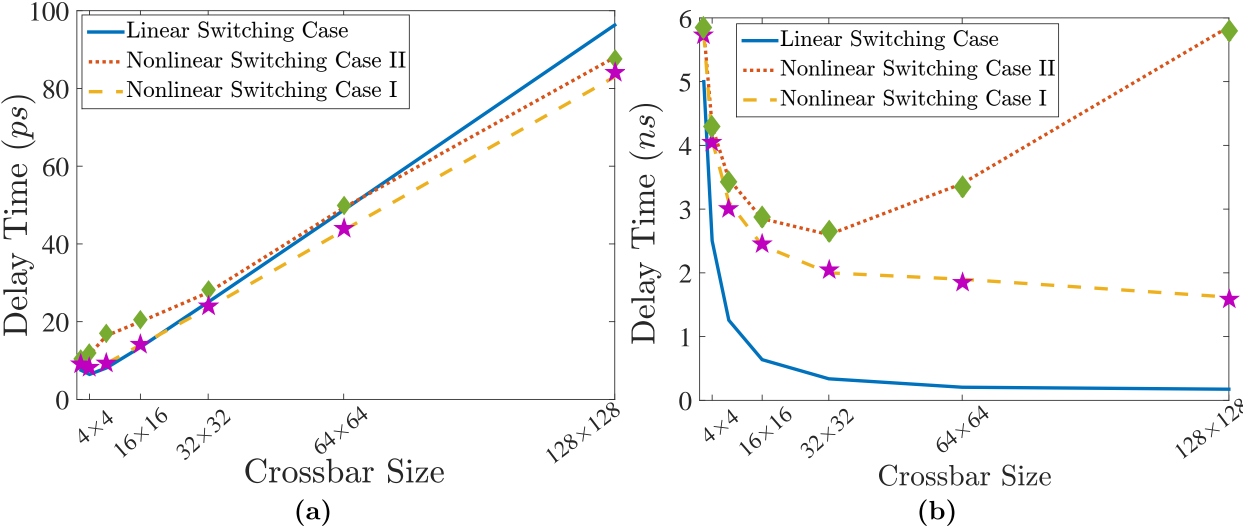


Figure 3.18: Comparison between the calculated delay of crossbar array containing linear and nonlinear switching devices for a) all LRS, and b) all HRS.

network with N samples and input sample size is 64 and output size is 64. In order to train this network, $N * t_s$ is needed to train the network where t_s is the SPICE simulation time. On the other hand, using the proposed model, the designer needs to simulate one time to get the maximum delay to determine the maximum speed of the network which takes t_d seconds. And run N simulations to get the final value for each sample which takes tf for each sample. Then the total simulation time is $t_d + N * t_s$. Then, the performance improvement between SPICE and our model is $\eta = N * t_s/(t_d + N * t_s)$ representing the time

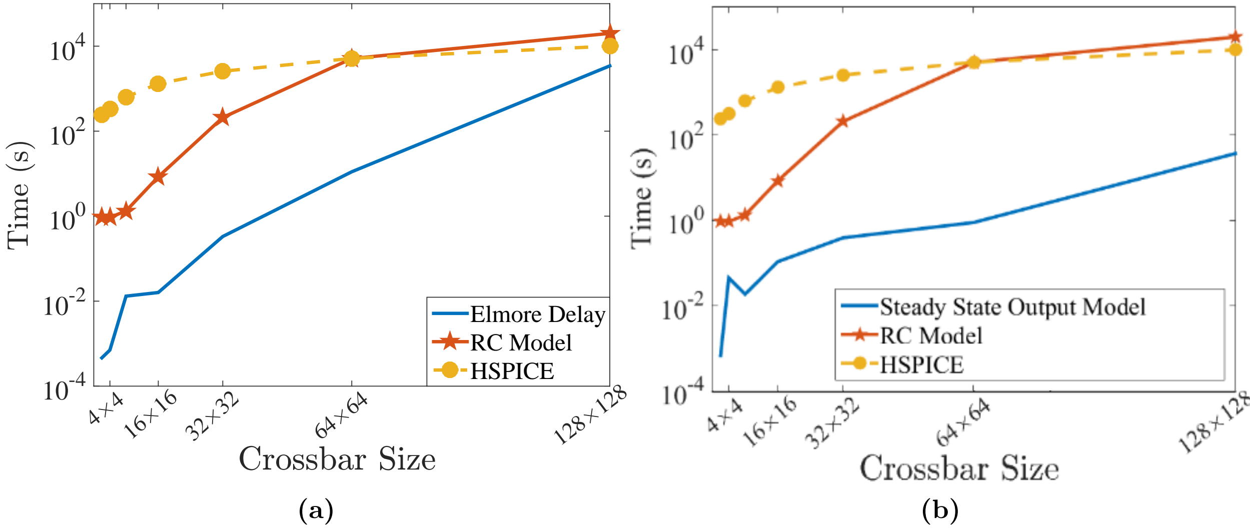


Figure 3.19: Simulation time comparison between SPICE and the proposed framework.

saving. Figure 3.19a shows a comparison for the simulation time between proposed model and HSPICE and for 64x64 array, Elmore delay calculation takes $t_d = 11.09$ s and SPICE simulation takes $t_s = 5160$s. Figure 3.19b shows a comparison for obtaining final outputs values using proposed model and HSPICE and the proposed model takes $t_f = 0.89$ s. Then, the performance improvement is 5798x which is very big improvement for one layer.

3.7 Conclusion

This chapter introduced a closed RC model for crossbar arrays taking into consideration the effect of parasitics and nonlinearity of the switching device in addition to its analytical solution. This solution has been verified against HSPICE simulation showing perfect matching. This model is used to determine the behavior of the crossbar due to the parasitics. Thus, this model is essential for the applications that need extensive simulations such as neural networks and pattern recognition. Also, a novel analysis for the delay of the crossbar has been derived for the first time. In general, the bigger the crossbar array, the more signal delay for the floating case in contrast with the grounded case. Moreover, the delay has a minimum limit which comes from the self loading of the crossbar. The more the line resis-

tance, the more delay is experienced for the floating case. For the grounded input crossbar, delay is dominated by the LRS so it should be low to get low delay, However, if it very low, the line resistance of the crossbar dominates. Thus, one can select the device depending on the required size of the crossbar by $LRS = N^2_{minD} * r_l$. such that delay is minimized. On the other hand, the delay in the floating case increase with increasing the crossbar size. As a result, the device, having minimum LRS, should be used. Finally, delay is dominated by the loading capacitance, where by choosing small loading capacitor, the capacitive parasitics in the crossbar affect the delay. As a result, suitable biasing for the crossbar is chosen based on the application.

Chapter 4

Reading and Writing Techniques for Crossbar Resistive Memories

4.1 Introduction

Over the last decade, emerging nonvolatile memories (NVMs), such as phase change memory (PCRAM), ferroelectric memory (FeRAM), spin transfer torque magnetic memory (STT-MRAM), and resistive memory (RRAM), have shown high potential as alternatives for floating-gate-based nonvolatile memories [3]. Figure 4.1 shows the emerging NVMs' capacities in the recent years. RRAMs are considered the best candidate for the next generation nonvolatile memory due to their high reliability, fast access speed, multilevel capabilities and stack-ability creating 3D memory architectures [99, 100].To achieve higher density memories, switching devices alone are sandwiched between the crossbar metal layers without using access devices such as transistors, diodes and selectors[100]. In some cases, the switching devices might have exponential behavior (the selector is embedded inside the switching device) such as FAST selector [39]. The main drawback of selector-less (gate-less) crossbar-based memories is the sneak path effects which limit the readability of the array. Conventional reading approaches for selector-less crossbars suffer from the sneak path loading which makes reading the data very difficult, and even impossible at times.

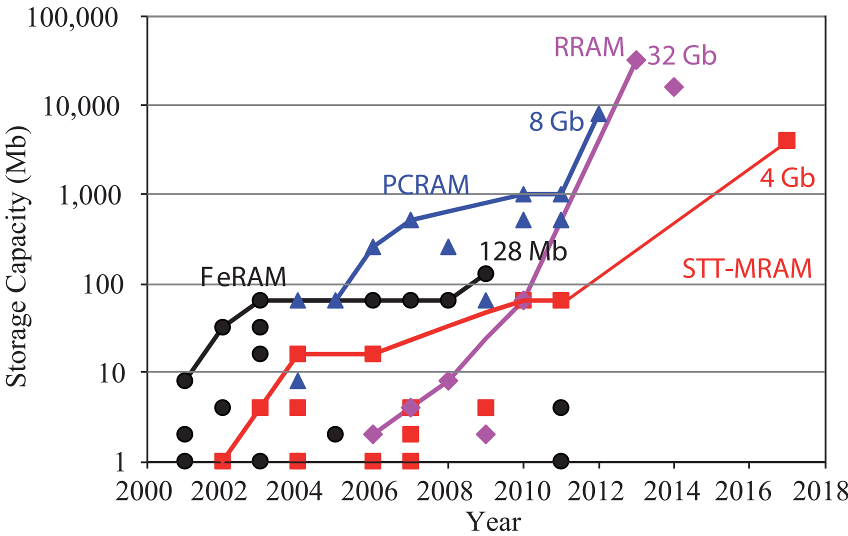


Figure 4.1: Trends in memory capacity for emerging NVMs adopted from [3].

The sneak paths problem arises because there are many paths from the inputs to the outputs. Figure 6.3b shows the sneak path in 2×2 crossbar array. The sneak path current is added to the main path current which disturb the cell reading. Figure 6.3a shows the cumulative probability of the readout current for 512×512 crossbar array with $LRS = 1M\Omega$, $HRS = 1G\Omega$ and 10Ω line resistance. The readout currents corresponding to LRS and HRS are totally overlapped. As a result, it is impossible to find a threshold to differentiate between the two states even with very large switching resistance values.

Recently, different readout techniques were proposed to address this problem in high-density arrays using two main procedures. (a) Reading and writing the stored data multiple times such as [28, 46] which require an Analog to Digital Converter (ADC), as well as registers and comparators. (b) Dispersing predefined dummy cells in the array to assist in reading the data such as [12, 101]. These dummy cells should be initialized first, which requires more than one cycle to read a certain cell and requires locality property, ADC and comparators, all of which limit the applicability of the technique. Other techniques have been proposed to read the data in parallel by adding sensing resistors to the bitlines and sense the voltage across the resistors [102] which loads the crossbar and does not mitigate the sneak path effects. Yet

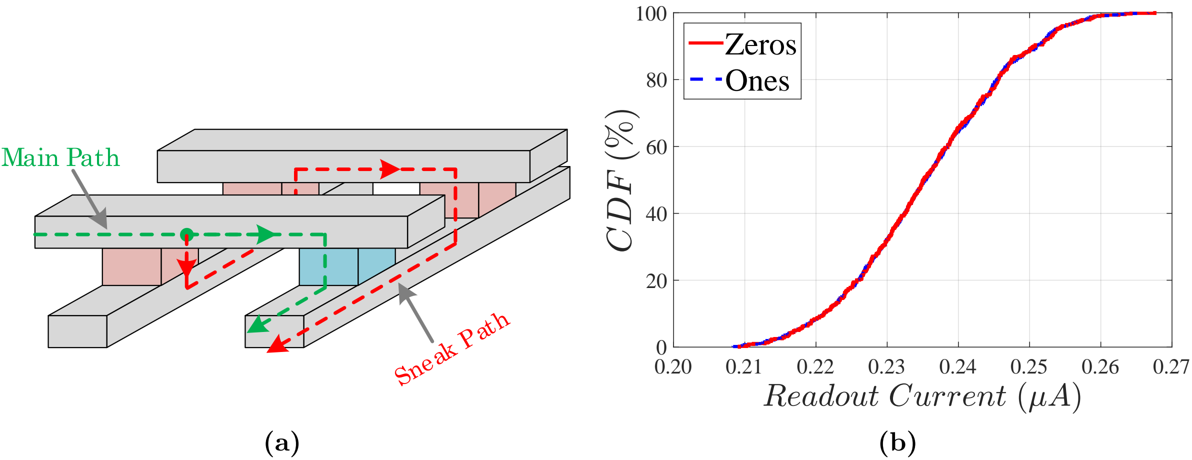


Figure 4.2: (a) Crossbar array with the sneak path problem, and (b) cumulative probability of reading 512×512 array.

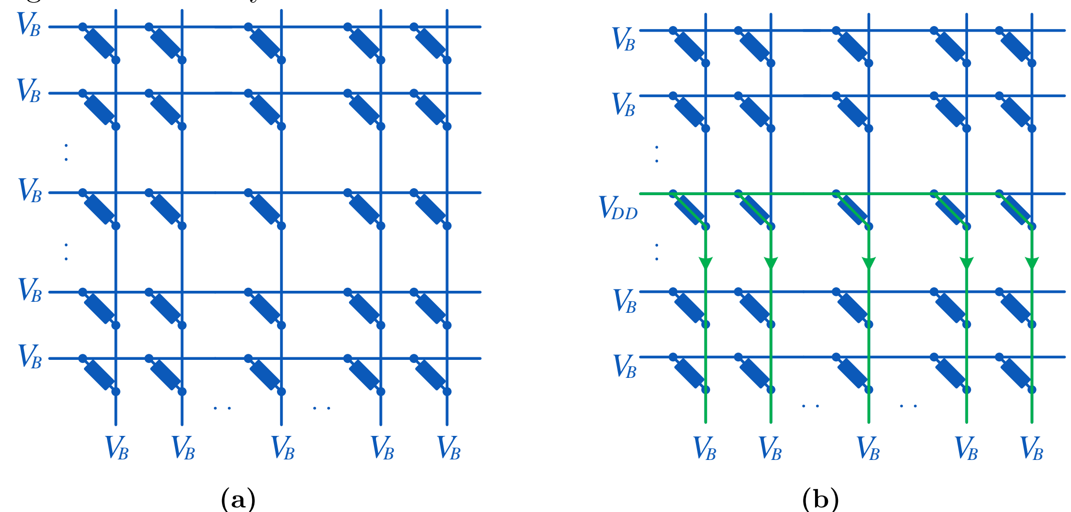


Figure 4.3: (a) Biasing scheme and (b) row reading scheme.

another approach is to keep the bitlines floating and sense the bitlines voltages as discussed in [103]. The floating bitlines schemes can read array sizes up 128×128 without errors. Both resistive load and floating reading schemes are suitable only for small size arrays. In this brief, a sneak path mitigation readout technique for high density 3D resistive memories is proposed in addition to the required peripheral readout circuitry.

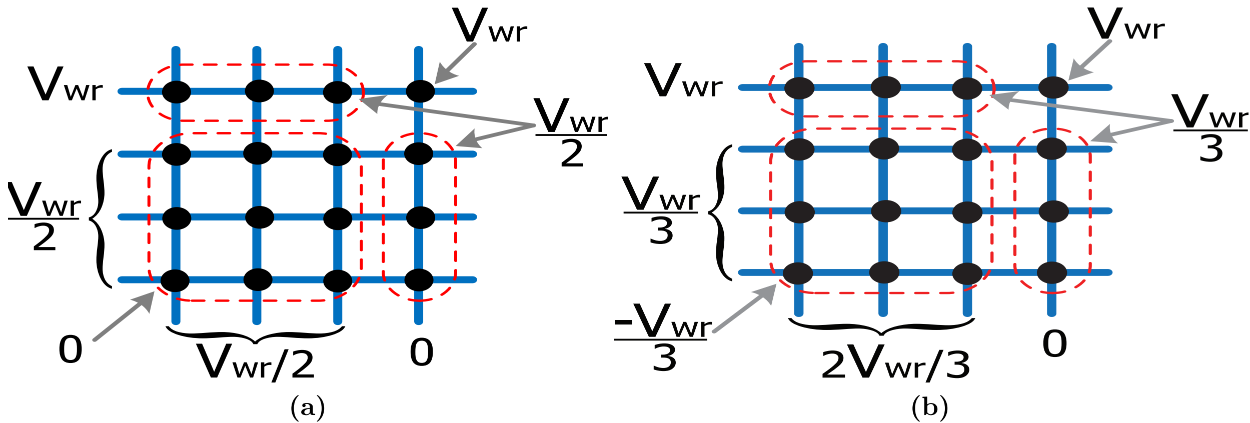


Figure 4.4: Basic bias schemes for writing crossbar based resistive memories: (a) 1/2 bias scheme, and (b)1/3 bias scheme.

4.1.1 Crossbar Writing Schemes

In literature, there are two basic writing schemes; 1/2 bias and 1/3 bias schemes. In the 1/2 bias scheme, the unselected columns and rows are biased to half of the write voltage $V_{wr}/2$, while the selected row and column are biased to V_{wr} and $0V$, respectively as shown in Fig. 4.4a. As a result, the voltage drops over the unselected devices is 0, and $V_{wr}/2$ for half-unselected devices and V_{wr} for selected devices, ignoring wire resistance. On the other hand, in the 1/3 bias scheme, the selected rows and columns are biased to $V_{wr}/3$ and $2V_{wr}/3$, respectively while the selected row and column are biased to V_{wr} and $0V$, respectively as shown in Fig. 4.4b. As a result, the voltage drop over the unselected and half-selected devices are $V_{wr}/3$, and V_{wr} for selected devices.

Figure 4.5 shows the switching behavior and the histogram of the set and reset switching behavior of the switching devices [104]. Practically, a significant voltage margin, ΔV, should be added to the average of set and reset voltages to reduce the probability of error while writing the selected cell.

The 1/3 bias scheme was introduced to reduce the write disturb problem where the voltage drop over the unselected and half-selected cells is $\pm V_{wr}/3$ compared to $V_{wr}/2$ for the half-

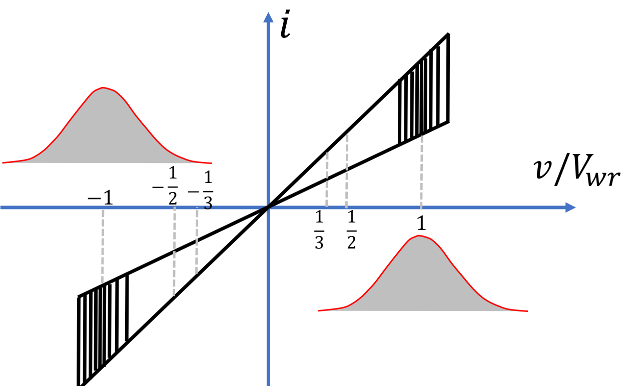


Figure 4.5: The set and reset switching behavior of RRAMs

selected cells in 1/2 bias scheme. However, this results in switching errors in the unselected cells, as well as a significant amount of power consumption.

4.2 Proposed One Step Row Readout Technique

Our target is to design circuitry that can be used for reading stacked resistive crossbar arrays. As is clear from previously published techniques, the sneak path problem appears due to the existence of many paths between the input and the outputs. To avoid this problem, we present a simple solution which mitigates the sneak path problem and maximizes throughput. Consider the case, where all the crossbar array input and output ports are biased to a certain bias voltage, V_B, as shown in Fig. 4.21a. For simplicity, assume that the input and output ports are grounded. Consequently, no current flows across the crossbar array. However, when a V_{DD} signal is applied to one of the input rows, current flows from this input to all the outputs and no current flows across the other rows where the voltage drop across the other rows is zero as shown in Fig. 4.21b. The current is absorbed by the sensing circuit. This current is proportional to the resistance of the switching device. In this way, the sneak current paths are eliminated. By reading this current, it is easy to distinguish between the low resistance and high resistance states. The architecture is inherently parallel, where all the row data can be accessed in parallel, enabling high throughput applications.

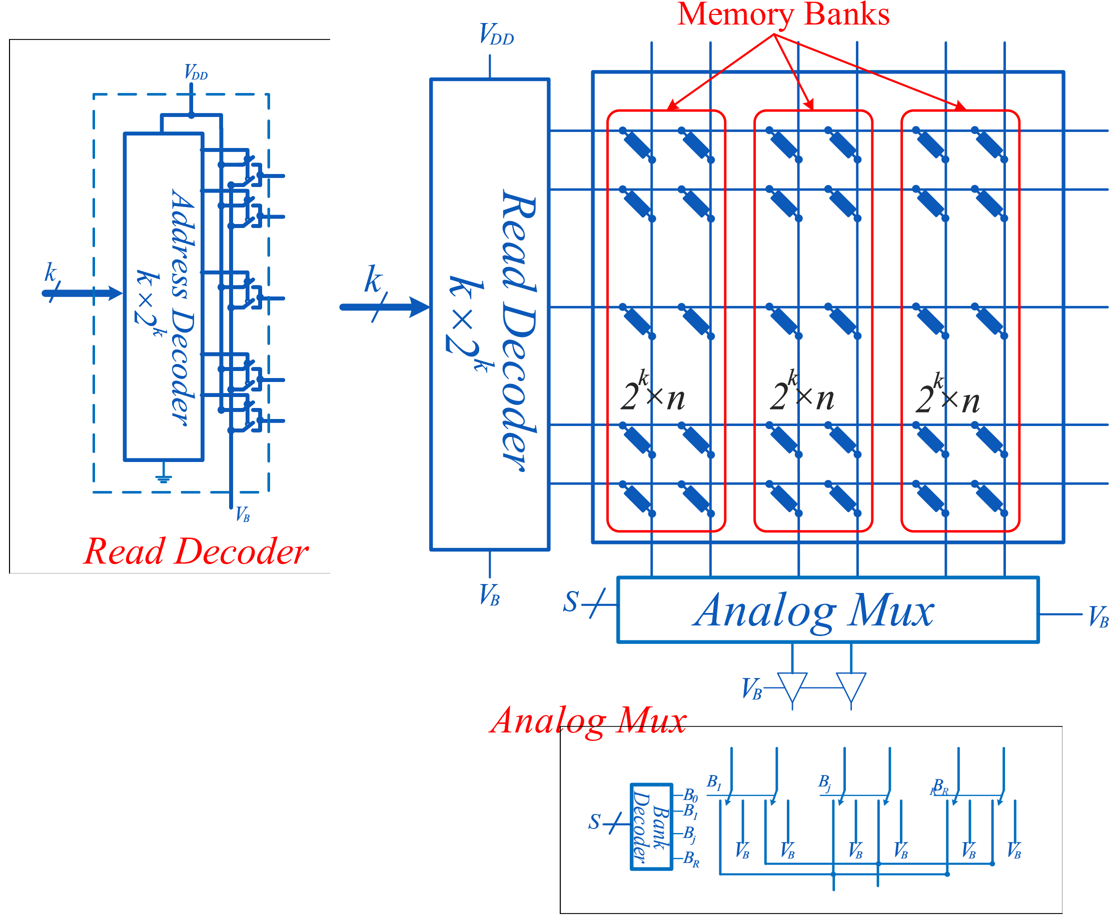


Figure 4.6: Schematic of full single-layered crossbar array.

Memory Architecture: To read the (i, j) cell, a V_{DD} signal is applied to the ith row and the current of output port number j is sensed where $1 \leq i \leq M$ and $1 \leq j \leq N$. This path can be modeled as resistor, R_m, between input and output ports which is either LRS or HRS. The maximum and minimum resistance can be used to determine the sensitivity of sensing circuit as will be discussed later. It is worth noting that this technique can be generalized to any crossbar size since the sneak current that is resultant from multi-paths is mitigated by this reading technique. Figure 4.6 shows the full schematic of the single layered crossbar array. The crossbar inputs are connected to the read decoder which selects only one input according to the input address. The selected line is biased by V_{DD} and unselected lines are biased to V_B. Grounding the lines is a special case of the general bias voltage V_B. Furthermore, if smaller word-lengths are needed, banking can be applied as shown in the figure, where each bank has size of $M \times n$ and the total number of banks is $R = N/n$. Consequently, one row per bank is read each clock cycle which means n readout circuits are needed. In order to choose between the banks, an analog mux is necessary which can easily be constructed using switches. Note that the bitlines of all unselected banks should be biased to V_B to guarantee the aforementioned scenario. The outputs of the analog mux are connected directly to the reading circuits which bias the selected bank to V_B and sense the current.

Effect of Wire Resistance: Wire resistance is inevitable in such crossbar arrays which effects the reading technique making the voltage across the selected devices less than $V_{DD} - V_B$ by a factor which is a function of the stored data and cell location. Due to the random nature of the data, it is hard to estimate this factor analytically. Figure 4.7 shows the sensed current density of each cell in 512×512 array, in addition to the histogram of the sensed current for both linear and nonlinear devices with 10Ω wire resistance and 10% resistance variations in both states. Generally, the wire resistance creates leakage paths from the selected wordline to the unselected wordlines. However, with these input to input leakage paths, the technique is still able to distinguish between LRS and HRS with wide current range for linear devices.

On the other hand, the nonlinear devices have exponential voltage-dependency modeled as $I = k \times \sinh(aV)$ where V is the voltage across the resistive device, k and α are the fitting parameters[12, 5]. Using such devices improves the sensing current range due to the high resistance facing the leakage currents in the input ports.

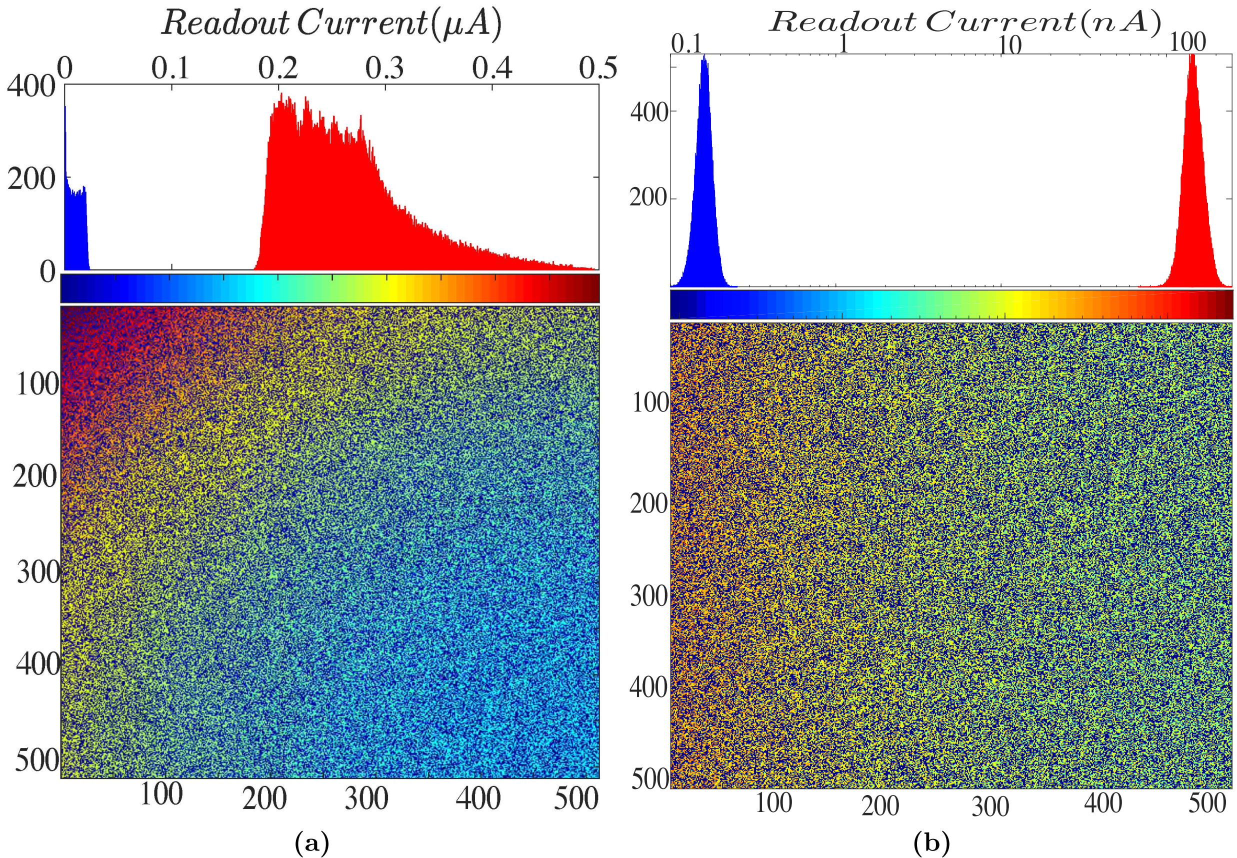


Figure 4.7: Combined plot for the 512 × 512 array sensed current density with histogram for (a)linear and (b) nonlinear devices.

Practically, the wordline is about 64-128 bits at most. Thus, there is no need to read the entire row for high dense wordlines more than 128 bits. The memory array can be divided into banks, as shown in Fig. 4.6,

4.2.1 Proposed current sensing circuitry

The readout circuit to sense the output current of the crossbar should satisfy the following specifications:

1) the sensing terminal has a fixed bias voltage, V_B, and

2) the range of sensed current needs to be identified.

One way to satisfy these requirements is to use the current conveyor concept where the applied voltage of *port* 1 is mirrored to *port* 2, while the input current to *port* 2 is mirrored to *port* 3 as shown in Fig. 4.8a[105]. The characteristics equations of the second generation current conveyor (CCII) can be summarized as follows:

$$\begin{bmatrix} i_1 \\ V_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ i_2 \\ v_3 \end{bmatrix} \tag{4.1}$$

where i_1, i_2 and i_3 are the current at *ports* 1, 2 and 3, respectively. From the previous equation, it is clear that the ideal CCII has zero current at *port* 1 and the current at *port* 2 is mirrored to *port* 3. In addition, the voltage at *port* 1 is mirrored to *port* 2. In our design, the complete CCII is not need for the sensing circuitry because the requirements can be satisfied with a simpler version of CCII. *Port* 1 can be biased to V_B which is mirrored to *port* 2 and the input current to *port* 2 is around $(V_{DD} - V_B)/R_{m,i,j}$ where $R_{m,i,j}$ is the (i, j) cell resistance which is either LRS or HRS. At this point, it is easy to distinguish between the LRS and HRS by designing suitable readout circuit.

The continuous behavior of RRAMs enables the ability of storing multi-level data. The proposed reading technique can be used to read multi-level resistive memories as well since it reads the selected cell resistance only. Thus, a suitable readout circuitry that differentiates

between the states is needed. In this work, we focus only on reading binary resistive memories. The proposed circuit is divided into two parts; 1) current sensing circuit which should have the aforementioned specifications which works as a transimpedance amplifier and 2) a latched comparator to distinguish between the two states and also to latch the data.

The Proposed Current Sensing Circuit

The proposed circuit is based on the current conveyor concept [105, 106] shown in Fig. 4.8b. The voltage V_B is the biasing voltage and can take any value as long as M_1 and M_2 are kept in the saturation region. M_2 and M_4 are designed so that the current passing through M_2 is mirrored to M_4. Consequently, the current passing through M_1 is equal to the current passing through M_3. Assuming that all transistors are in the saturation region, then $V_{GS_1} - V_{thp_1} = V_{GS_3} - V_{thp_3}$. Hence,

$$V_{in} = V_B - \Delta V_{thp} \tag{4.2}$$

where $\Delta V_{thp} = V_{thp_3} - V_{thp_1} = \gamma(\sqrt{V_{SB_3} + 2\phi_F}$-$\sqrt{V_{SB_1} + 2\phi_F})$. By connecting these transistors' bodies to their sources, $\Delta V_{thp} = 0$, and $V_{in} = V_B$. Now, the first condition in the reading circuit is satisfied.

The current passing through M_3, I_3, is constant and its value equals to I_1 because of the current mirror between M_2 and M_4. The value of the current ,I_1, can be obtained as follows: M_1 and M_2 are diode connected transistors and the current passing through them is the same, thus

$$k_{p1}(V_B - V_{G1} - |V_{thp1}|)^2 = k_{n2}(V_{G2} - V_{ss} - V_{thn2})^2 \tag{4.3}$$

The gates of M_1 and M_2 are connected ($V_{G1} = V_{G2} = V_G$), thus,

$$V_G = \frac{\sqrt{\theta}(V_{ss} + V_{thn2}) + V_B - |V_{thp1}|}{\sqrt{\theta} + 1} \tag{4.4}$$

where $\theta = k_{n2}/k_{p1}$. By substituting into the current equation of M_2, the current passing though the two transistors is given by

$$I_1 = \frac{k_n}{2} \left(\frac{V_B - V_{ss} - V_{thno} - |V_{thpo}|}{\sqrt{\theta} + 1} \right)^2 \tag{4.5}$$

From this equation, the bias voltage V_B should be greater than $V_{ss} + V_{thno} + |V_{thp0}|$. Thus, I_1 is a constant current that depends on the biasing voltage and aspect ratios of the transistors.

By applying KCL at the input node, the current passing through M_5 is $I_5 = I_{11} + I_{in} - I_3$. This current is mirrored through M_6 to the output node and imposed into the load resistance creating the output voltage, $V_o = V_{DD} - (I_{11} + I_{in} - I_3)R_L$ where I_{11} is mirrored from I_{10} and its value is αI_{ref} where α is the ratio between the aspect ratios. I_3 is a constant current and equals I_1 due to the current mirror effect.

Consequently, the output voltage is $V_o = V_{ref} - I_{in}R_L$, where $V_{ref} = V_{DD} - (\alpha I_{ref} - I_1)R_L$. The input current I_{in} is either $(V_{DD} - V_B)/LRS$ or $(V_{DD} - V_B)/HRS$. Thus, we have two outputs, V_{oh} and V_{ol} corresponding to HRS and LRS, respectively. It is necessary to widen the difference between V_{ol} and V_{oh} to easily distinguish between the states, which is possible by controlling the values of R_L, α and I_1. This circuit can be followed by either a buffer or a latch circuit so that the output swings between V_{DD} and V_{ss}. To avoid having a large loading resistor occupying a large area, with limited output voltage range, a latched comparator is necessary.

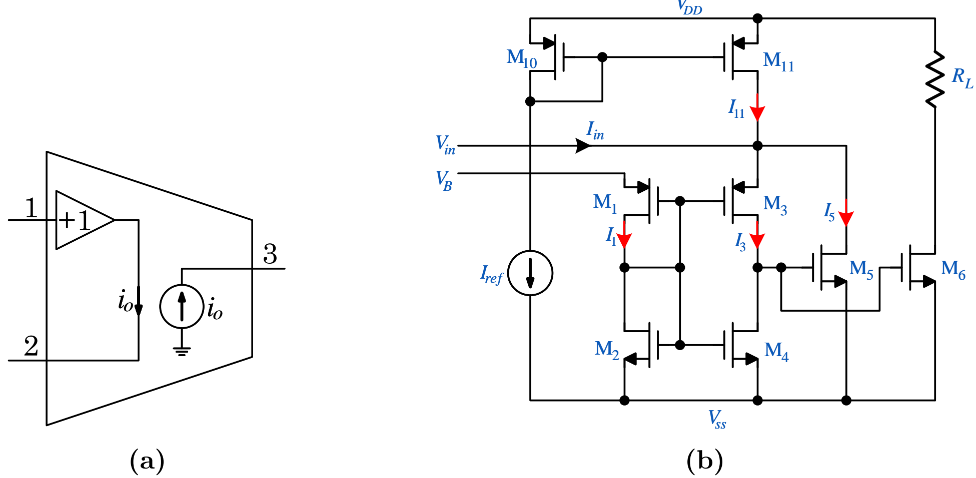


Figure 4.8: (a) Current conveyor principle and (b) schematic of proposed current sensing circuit.

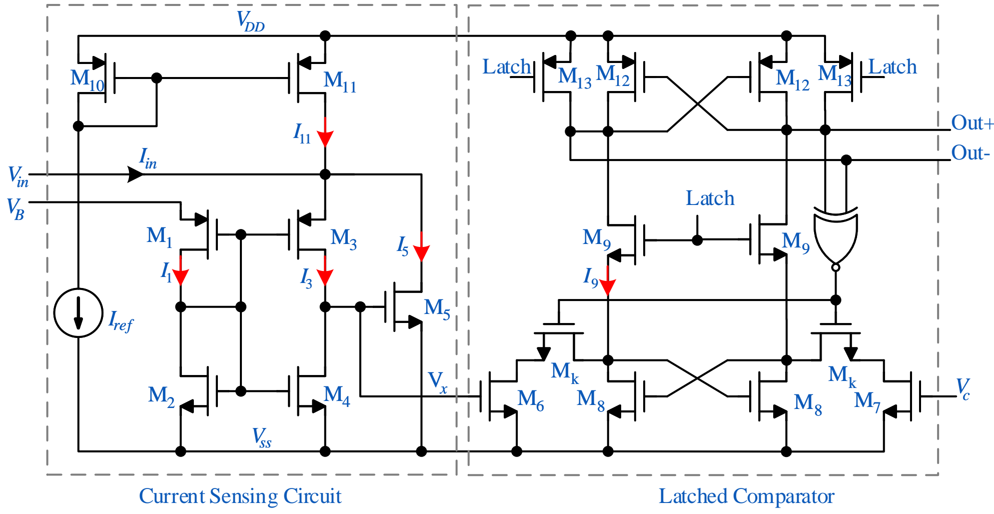


Figure 4.9: Schematic of the full proposed readout circuit.

Table 4.1: The proposed readout circuit parameters.

Transistors	Aspect ratio	Transistors	Aspect ratio
$M_{1,3}$	$360nm/250nm$	$M_{2,4}$	$250nm/0.6\mu m$
M_5	$120nm/300nm$	$M_{6,7}$	$360nm/300nm$
M_8	$240nm/60nm$	M_9	$120nm/60nm$
M_{10}	$250nm/250nm$	M_{11}	$500nm/250nm$
$M_{12,13}$	$480nm/120nm$	M_k	$120nm/120nm$
V_{DD}	$1.2V$	I_{ref}	$0.5\mu A$

Latched Readout Circuit

Instead of reading the output current from the loading resistor, this current can be connected directly to a latched comparator as shown in Fig. 4.9. The gate voltage of M_5 changes depending on the current passing though it which is mirrored in M_6. The mirrored current is compared with the constant current generated by M_7 due to constant voltage V_c. The M_k transistors are used to reduce kick back noise where the latch signal voltage goes back to the input signal which may alter the data. This latched comparator is introduced and analyzed in [107]. In the reset case, both outputs are equal to V_{DD} where the output of the XOR gate is zero. On the other hand, in the set case, $V_{Latch} = V_{DD}$. Fig. 6.4d illustrates an example of transient simulation of reading random data from a certain column using the proposed circuit. The readout circuit is designed using TSMC65nm to satisfy the aforementioned conditions with the parameters tabulated in Table I.

The area of the entire readout circuitry is about $55\mu m^2$ with $1.92\mu W$ total power consumption, at 1.2V supply.

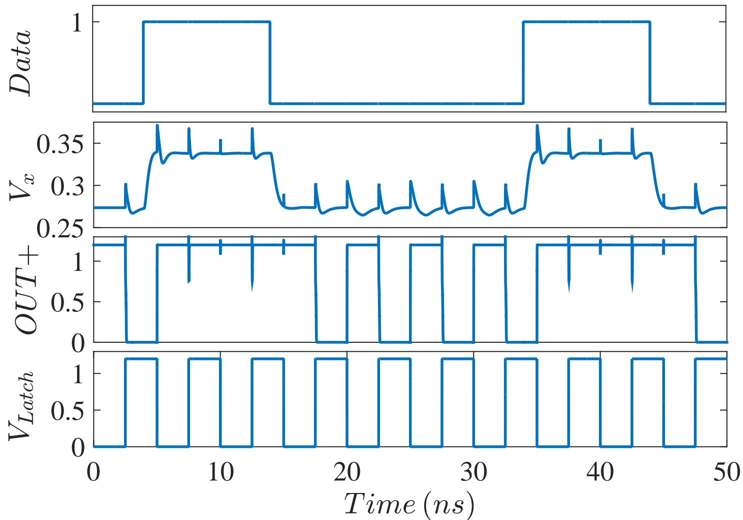


Figure 4.10: Simulation results of the proposed circuit.

4.2.2 Discussion and Comparison

4.2.3 Power Consumption Estimation

Power consumption is very critical in resistive memories since they consume a lot of power during the reading and the writing operations due to their inherent resistive nature. Thus, it is essential to estimate the power consumption inside the crossbar array as well as the power density. The device nonlinearity highly affects the power consumption where the higher the nonlinearity, the higher the resistance which implies less power.

In case of linear devices, where the resistive device states are constant and not a function of the applied voltage, the power consumption for reading one wordline (for both with and without banking) can be approximated by multiplying the voltage drop times the input current, ignoring wire resistance, $P_{wl_i} = \sum_{j=1}^{N} \frac{(V_{DD}-V_B)^2}{R_{m,i,j}}$. Thus, the maximum and minimum power consumption are around $M\,N\,R\,(V_{DD} - V_B)^2/LRS$ and $M\,N\,R\,(V_{DD} - V_B)^2/HRS$, respectively. on the other hand, in case of the nonlinear devices, the voltage across the switching device is still around $V_{DD} - V_B$. Thus, the power consumption per wordline

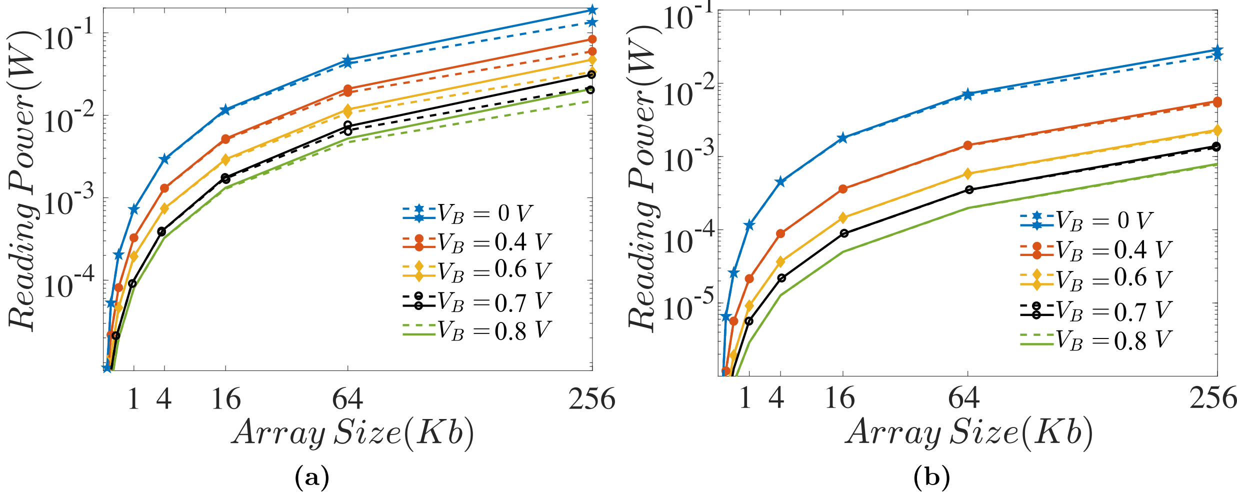


Figure 4.11: Reading power versus the array size at $V_{DD} = 1.2V$ with and without wire resistance (dashed lines and solid lines) for (a) linear switching and (b) nonlinear switching devices

is $P_{wl_i} = \sum_{j=1}^{N} k_{i,j}(V_{DD} - V_B)\sinh(a_{i,j}(V_{DD} - V_B))$ by ignoring wire resistance. Figure 4.11 shows the reading power consumption with and without including the line resistance for different biasing voltages and different array sizes. The figure is plotted for nonlinear switching devices, with k_{on}, k_{off} and a are $1e-8, 1e-11$ and 3, respectively[46]. Clearly, by increasing the biasing voltage, V_B, a lower power consumption is obtained. However, this reduces the sensing current margin which highly affects the sensing circuit. Therefore, it is important to study the effect of changing V_B.

Figure 4.12a shows the effect V_B over the voltage swing and the delay. As previously discussed, the bias voltage should be greater than twice the transistor threshold voltage which is around $0.63V$. The output voltage swing exhibits critical curve with maximum point at around $V_B = 0.75V$. Also, the delay for both reading scenarios (reading LRS then HRS and vice versa) is also shown. In our design, we set a practical target of $1ns$ for the delay. The best bias voltage is $0.7V$ to maximize the voltage swing. Another aspect that should be studied is the value of the input voltage. Figure 4.12b shows the effect of changing V_{in} with fixed $V_B = 0.7$. The higher the input voltage, the more the voltage swing, the less delay and

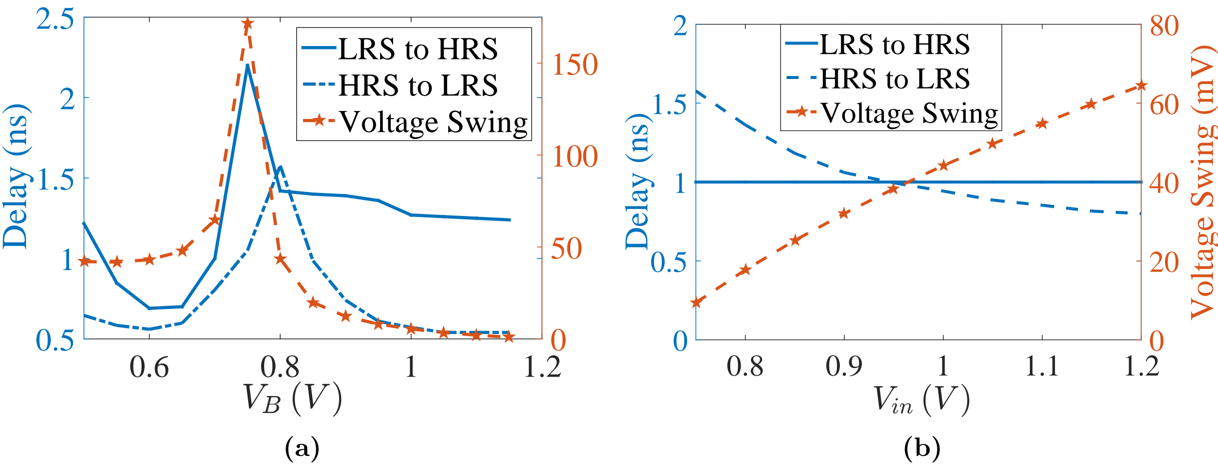


Figure 4.12: Input comparator swing and delay versus (a) bias voltage with $V_{in} = 1.2V$, (b) applied voltage with $V_B = 0.7V$.

more power consumption as shown in Fig.4.11.

$M_1 - M_4$ transistors work as as a gain boosting circuit. Hence, the input impedance of the sensing circuit is high ($\approx 1M\Omega$). Due to the abrupt current changes while reading, V_{in} is disturbed ($+20mv$ and $-35mV$ around V_B). The negative feedback of the gain boosting circuit works to recover V_{in} to V_B. In our design, the loop recovers in $1ns$. Thus, the delay of the designed circuit is $1ns$. Practically, two phases are needed due to the latched comparator; (a) a reset phase where the output is set to V_{DD} and the data is setup and (b) a latch phase where the data is latched and stored. Thus, another $1ns$ is needed to latch data for 50% duty cycle clock. Per these parameters, the energy consumption of the readout circuit per bit is $7.6fJ$ for $500MHz$ clock frequency.

Figure of Merit

In [103], a figure of merit (FOM) is defined for comparing different reading techniques taking into account important metrics such as throughput, and array usage. This FOM is defined

Table 4.2: Performance metric of reading a complete $N \times N$ array. These results are adopted from [12, 13].

Technique	**Readout Circuit**	**Throughput**	**Locality Needed**	**Array Usage**[1]	**Power**[2,*] (mW)	**FOM**[2] $(Tbits/W\mu m^2)$
Multistage	ADC + Comp	$\frac{1}{6}$	No	1	7	0.04
Multiport	ADC + Comp	$\frac{1}{3}$	No	$\frac{N-2}{N}$	2.1	0.265
Grounded Rows & Cols	VG + Comp	1	No	1	4	0.4194
Predefined Dummy Bits	VG + Comp	1	Yes	$\frac{N-1}{N}$	0.291	5.754
This work	VG + Comp	N/R	No	1	1.358 R	$633/R^2$

[1]Array Usage = number of data bits/total number of bits.
[2]Power and FOM results are reported for 256Kb array.
[*] Without bias mismatch
R total number of banks

as:

$$FOM = \frac{Throughput \times Array\,Usage}{Reading\,Power/cell \times pitch\,size} \quad (bit/Wm^2) \tag{4.6}$$

where the numerator reflects the array metrics; *Throughput*, which is the number of read bits per cycle (bank size), and *Array Usage* which is the number of usable data bits divided by the total number of bits. The denumerator, reflects per bit the physical parameters, *power per bit* and cell area. It is reported that the array density of memristor based selector-less crossbar arrays is approximately $640Gbit/cm^2$ where the feature size is $6.25nm$[12]. Table I shows a quantitative analysis for different reading techniques for a complete $N \times N$ array. We chose to compare with these four techniques which can accommodate high density crossbar arrays. This comparison illustrates the differences between prior work and the proposed approach in terms of power, throughput, array usage, and FOM.

The estimated area of the 512×512 crossbar is around $40.96\mu m^2$ and the estimated area of the read decoder is $4.26\mu m^2$ based on the technique published in [108, 109] with two pre-decoding stages. To estimate the sensing circuitry area, a 128 bit word is considered where the array is divided into 4 banks. The estimated area of 128 sensing circuit including the MUX is around $41.65\mu m^2$. The area of CMOS circuitry is estimated with respect to $5nm$

technology which is expected to be used with crossbar arrays. According to these estimated numbers, the entire CMOS circuitry can be placed under the crossbar array. And, the static power dissipation is dominated by the sensing circuits which is around $285.7\mu W$ based on $1.92\mu W$ per bit.

In this comparison, the exponential RRAM model is used with the aforementioned parameters values and feature size. The main advantage of the proposed technique is the ability to read the entire row bits in one clock cycle which is vital in memories on the contrary with the other techniques which requires at least N clock cycles to read the entire row. However, the proposed technique consumes more power due to the nature of the reading technique. It is worth noting that other published works do not account for the readout circuitry, which should be accounted for in addition to any building blocks such as ADCs and comparators [46, 28]. The circuits proposed in this work can be used in other approaches as well such as with predefined dummy bits and grounded rows and columns [101, 12], which requires virtual ground and comparator.

4.2.4 Bias Mismatch Effect

In resistive memories, it is required to bias wordlines and bitlines to specific voltages based on the technique used. For example, in the proposed reading scheme, the unselected wordlines are connected to V_B, and all the bitlines are connected to V_B through the sensing circuit. Thus, there would be a mismatch among the wordlines bias voltages themselves and with the bitlines bias voltages creating undesirable current due to PVT variation, wire parasitics and switch's resistance. This undesirable current is unavoidable and needs to be taken into consideration since it would limit the array size.

The voltage mismatch is column independent and is not correlated to other columns. Hence, the number of resistive devices that are affected by the mismatch is $N-1$ devices. This mis-

match is referred to as ΔV. In case of linear switching devices, the current passing through each unwanted device is either $I_{LRS} = \Delta V/LRS$ or $I_{HRS} = \Delta V/HRS$. For equal probable states, the total unwanted current is $I_{unW} = \Delta V(0.5N/LRS + (0.5N-1)/HRS)$. Usually the ratio between HRS and LRS is 10^3 or more, then the total unwanted current is approximated to $I_{unW} \approx 0.5N\Delta V/LRS$. However, the desired current for high/low resistance state is $I_W = (V_{DD} - V_B)/HRS$ or $I_W = (V_{DD} - V_B)/LRS$, respectively. Consequently, the extreme input current to the sensing circuit is $I_t = I_W \pm I_{unW}$ for high resistance and low resistance states, respectively. This sensed current affects the input voltage of the comparator V_x directly, which should not exceed the noise margin of the comparator. For $10mV$ noise margin for the comparator, the maximum/minimum input sensed current, which are corresponding to LRS and HRS, are $I_{max} = 0.22\mu A$ and $I_{min} = 0.195\mu A$, respectively. The maximum column width, N, is the minimum of $2 * I_{max} * LRS/\Delta V$ or $2 * I_{min} * LRS/\Delta V$ which is 195 for $2mV$ bias mismatch.

On the other hand, in the case of nonlinear switching devices, the current passing through each unwanted device is $I_{LRS/HRS} = k_{on/off}\sinh(a\Delta V)$ for low/high resistance state. Since ΔV is very small in range of millivolts, the current can be approximated to $I_{LRS} \approx k_{on}a\Delta V$ or $I_{HRS} \approx k_{off}a\Delta V$. The total unwanted current is $I_{unW} = a\Delta V(0.5Nk_{on} + (0.5N-1)k_{off})$. Usually the ratio between k_{off} and k_{on} is 10^3 or more, then the total unwanted current is approximated to $I_{unW} \approx 0.5Na\Delta V k_{on}$. However, the wanted current for high/low resistance states is $I_W = k_{off/on}\sinh(a(V_{DD} - V_B))$. Consequently, the extreme input current to the sensing circuit is $I_t = I_W \pm I_{unW}$ for high resistance and low resistance states, respectively. The maximum column width,N, is the minimum of $2 * I_{max}/(\Delta V * a * k_{on})$ or $2 * I_{min}/(\Delta V * a * k_{on})$ which is around 6500 for $2mV$ bias mismatch. With nonlinear devices, the column width of the crossbar is highly increased due to the high resistance facing the mismatch voltage.

4.3 Modeling the write disturb problem

The switching voltage of the switching devices (RRAMs) is a random variable and has a Gaussian distribution with mean V_{wr} and variance σ. Figure 4.13 shows the general bias scheme which is a function of all the bias parameters $(\alpha, \beta, \Delta V)$ where α and β are the bias voltages for the wordlines and bitlines, respectively. The 1/2 and 1/3 bias schemes are special cases where $\alpha = \beta = V_{wr}/2$ and $\Delta V = 0$ for 1/2 bias scheme and $\alpha = V_{wr}/2, \beta = (2V_{wr})/3$ and $\Delta V = 0$ for 1/3 bias scheme. This architecture can be divided into three partitions; a) unselected partition having $(M-1) \times (N-1)$ unselected devices, b) half selected partition having $(M+N-2)$ half selected devices and c) the selected partition having one selected device. Each partition has probability of error based on the applied voltage across each device. The switching probability of a certain device having a voltage drop, V_d is

$$P_{sw}(v < V_d) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{V_d} e^{-\left(\frac{v-V_{wr}}{\sqrt{2}\sigma}\right)^2} dv = \frac{1}{2} + \frac{1}{2} erf\left(\frac{v - V_{wr}}{\sqrt{2}\sigma}\right) \tag{4.7}$$

where $erf()$ is the error function. It is desired that the unselected and half-selected partitions do not switch, thus the probability of error in these partitions is the same as the switching probability ($P_e = P_{sw}$). For the selected device, the probability of error is the inverse of the switching probability which is $P_e = 1 - P_{sw}$. In this problem, it is required to minimize the overall error of writing crossbar memories. The bit error in each partition is the number of devices per partition times the probability of error in this partition. Thus, the total bit error

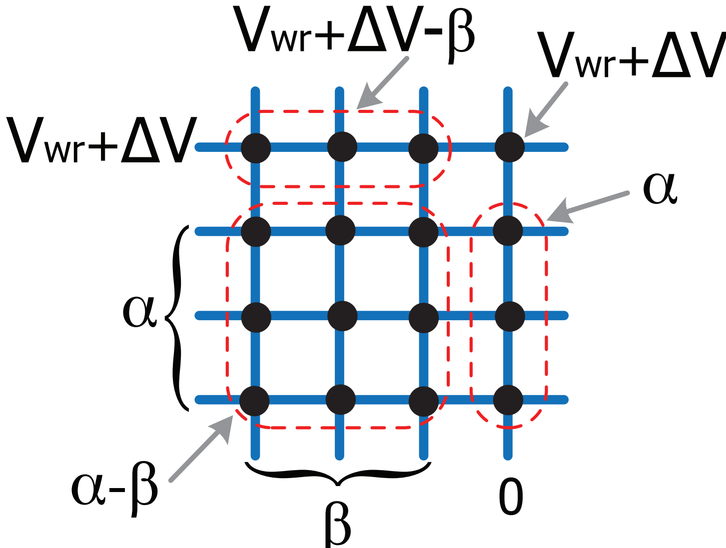


Figure 4.13: General Bias scheme of writing crossbar based resistive memories

BE is the sum of the bit errors of all the partitions and can be written as

$$
\begin{aligned}
BE(\alpha, \beta, \Delta V) &= (M-1)(N-1)P_e(v < \beta - \alpha) + (M-1)P_e(v < \alpha) \\
&+ (N-1)P_e(v < V_{wr} + \Delta V - \beta) + (1 - P_e(v < V_{wr} + \Delta V))
\end{aligned}
\tag{4.8}
$$

where the first term represents the unselected partition, the second and the third terms represent the half-selected partition and the fourth term represents the selected partition.

4.3.1 Optimality problem formulation and solution

The goal of the optimization formulation is to minimize the number of disturbed bits. This problem can be formulated as follows:

$$
\underset{\alpha, \beta, \Delta V}{\text{minimize}} \quad BE(\alpha, \beta, \Delta V) \tag{4.9}
$$

subject to $0 \leq \alpha, \beta, \Delta V \leq V_{wr}$. This problem can be solved analytical by finding the gradient of BE and equating it to zero. The gradient of BE can be written as:

$$\nabla BE(\alpha, \beta, \Delta V) = \left[\frac{\partial BE}{\partial \alpha} \;\; \frac{\partial BE}{\partial \beta} \;\; \frac{\partial BE}{\partial \Delta V} \right]^T = \mathbf{0} \tag{4.10}$$

To find these derivatives, the differentiation under integral sign (Leibniz rule) is used where

$$\frac{d}{dx} \int_a^{b(x)} f(x,t)dt = f(x, b(t)) \cdot \frac{db(x)}{dt} \tag{4.11}$$

After applying the Leibniz rule and simplification, a system of three nonlinear equations is obtained and can be written as follows:

$$\begin{aligned} \alpha &= \frac{\beta}{2} + \frac{\sigma^2 ln(N-1)}{2V_{wr} - \beta} \\ \beta &= \frac{\alpha + V_{wr} + \Delta V}{2} - \frac{\sigma^2 ln(M-1)}{\alpha + V_{wr} - \Delta V} \\ \Delta V &= \frac{\beta}{2} - \frac{\sigma^2 ln(N-1)}{\beta} \end{aligned} \tag{4.12}$$

By solving the three equations simultaneously, the optimal values can be obtained. Figure 4.14 shows the normalized optimal bias parameters with respect to V_{wr} for different square crossbar arrays at $\sigma = 0.1V_{wr}$.

4.3.2 Results and Discussion

Figure 4.15 shows the bit error rate (BER=BE/crossbar size) for different crossbar arrays and different variance. The figure also shows BER for the 1/2 and 1/3 bias schemes with $\Delta V = 0.1V_{wr}$. Clearly, the optimal bias scheme has a significant improvement compared to the other techniques. Also, both 1/2 and 1/3 bias schemes have a comparable bit error rate (BER).

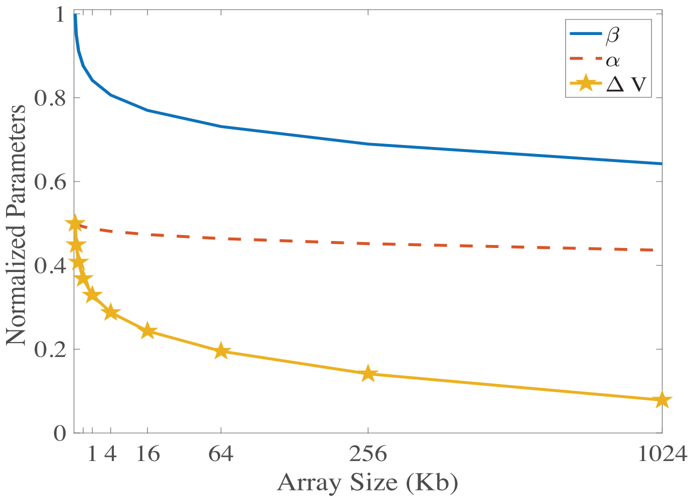


Figure 4.14: Normalized optimal bias parameters versus the array size with 10% tolerance.

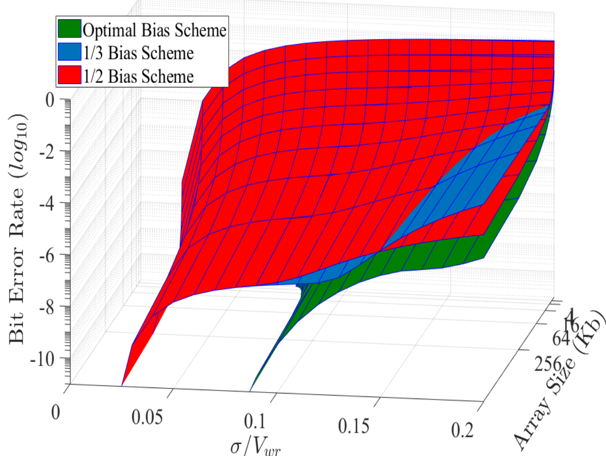


Figure 4.15: Bit Error rate for the optimal Bias scheme versus the known bias schemes.

Figure 4.16 shows the estimated power consumption for the three bias schemes. The 1/2 bias scheme has the lowest power consumption since almost all the power is consumed in the unselected cells. The optimal bias scheme consumes slightly more power than the 1/3 bias scheme. Thus, the 1/2 bias scheme is the best for low power operation and the optimal bias scheme is preferred for high performance operation.

4.4 One Step Row Write Technique

Our target is writing the entire data word in the resistive memories in the same clock cycle. Thus, the one step writing by performing the following three steps shown in Fig. 4.17: a) all

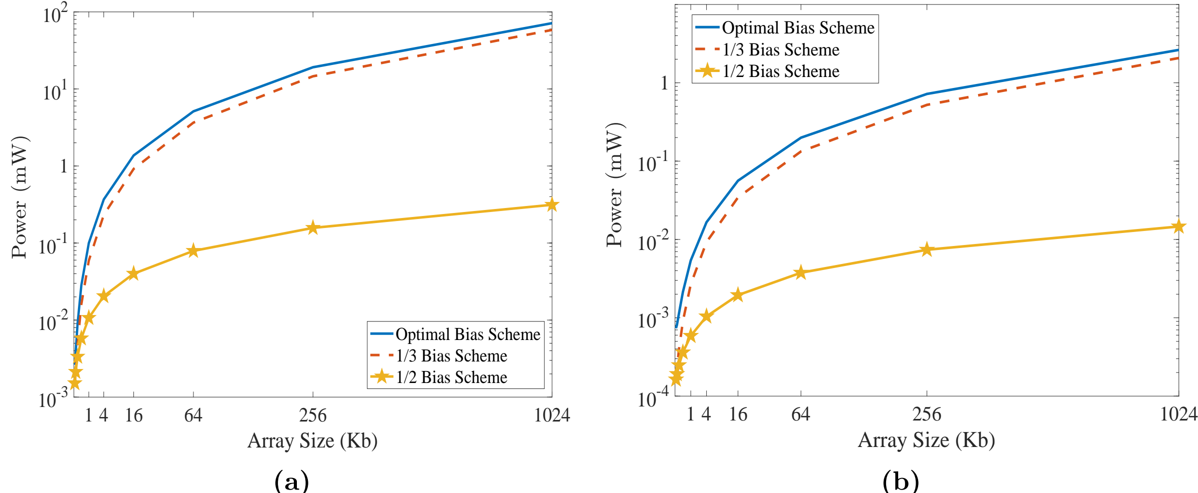


Figure 4.16: Estimated power consumption of the optimal Bias scheme versus the known bias schemes for (a) linear $I-V$ switching devices, and (b) nonlinear $I-V$ switching devices.

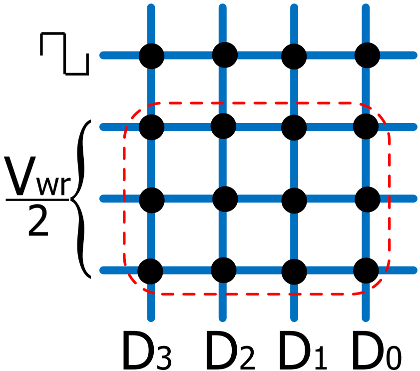


Figure 4.17: Bias scheme for one step writing technique.

unselected worldines are biased to $V_{wr}/2$. b) the data word is applied to the bitlines. and c) a clock signal is applied to the selected wordline.

The following is an example of writing a 2×2 array, shown in Fig. 4.18, with $[10; 01]$ data array. In order to write these data, the voltage across each device should be $[V_{wr} - V_{wr}; -V_{wr}V_{wr}]$. In order to write [1 0] in the first row (wordline) during the first clock cycle, V_{w2} is biased to $V_{wr}/2$ and data is applied to the bitlines where $(V_{B1}, V_{B2}) = (-V_{wr}, V_{wr})$. While a bipolar clock signal is applied to V_{w1} with amplitude $\pm V_{wr}$. During the first (positive) half cycle of the clock signal, the voltage drop across device R_{11} is V_{wr} which means the device is set (switch to low resistance state). And, the voltage across device

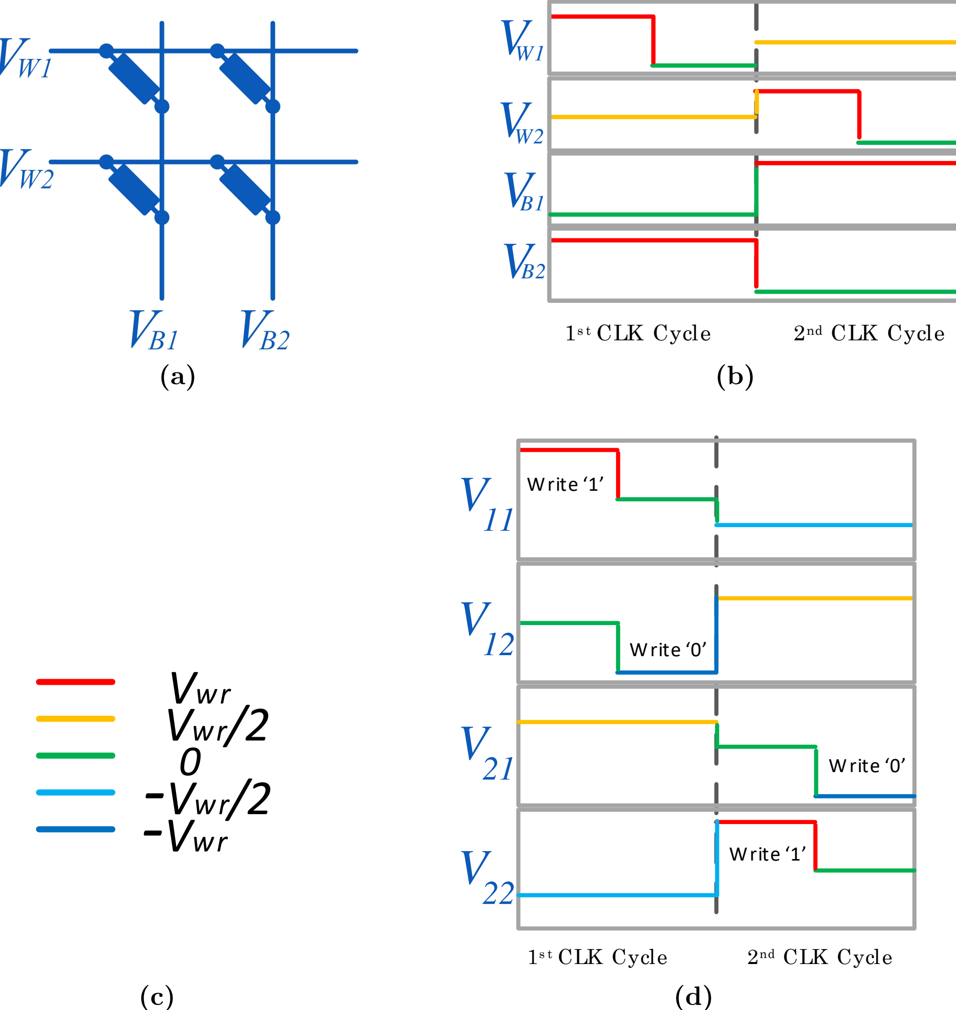


Figure 4.18: An illustrative example of writing a) 2×2 crossbar array; b) The wordline and bitline voltages, and c) the voltage drop across each device.

R_{12} is zero which means no change (non-destructive writing). On the other hand, during the second (negative) half cycle, the voltage drop across device R_{11} is zero which means no change in the written data (no overwrite problem). However, the voltage drop across device R_{12} is $-V_{wr}$ which reset the device (switches to high resistance state). The voltages across the 2nd wordline devices during the entire clock cycle are either $-0.5V_{wr}$ or $0.5V_{wr}$ which means no switching occurred. Similary, the same behavior is obtained for 2nd wordline where V_{wr} is biased to $0.5V_{wr}$, and a clock signal is applied to 2nd word line while the data are applied to bitlines as shown in Fig. 4.18.

4.5 Combined Reading and Writing Circuitry

Figure 4.19 shows the complete architecture of the memory with one step reading and writing techniques. The crossbar is connected as shown where the vertical lines (bitlines) are connected to the sensing circuits at the bottom with switches for reading mode and are connected word data during write operation through switches. While the horizontal lines are connected to an address decoder with some switches to enable both write and read operations. While, the reading operation is performed when (), which means one wordline is activated (biased to) which is selected by the address decoder and the rest are grounded. At the same time, the sensing circuitry is connected to the bitlines to read the data as shown in the Fig.5. On the other hand, when (), the write operation can be performed where a clock signal is connected to desired address and the other rows are connected to . Meanwhile, the input data are biased to bitlines of the crossbar array and the writing operation is performed as explained previously.

4.6 Non-Stationary Polar Codes for Resistive Memories

In this section, the afromentioned parallel reading of an entire crossbar row is adopted (see Fig. 4.20). It eliminates the multi-path problem in single-cell reading [12], one of the causes of sneak path. But, the inevitable wire resistances lead to undesirable voltage drops, another type of sneak path problem. These voltage drops are functions of the stored data and the wire resistance. At the expected feature size of $F = 5nm$ of RRAMs, the wire resistance per cell reaches as high as 90Ωas discussed in the previous chapters.

Fig. 4.21a shows the measured current of each cell in a (32×32) array with 25Ω wire resistance, storing random data, which is generated by the SPICE-like simulator of [5].

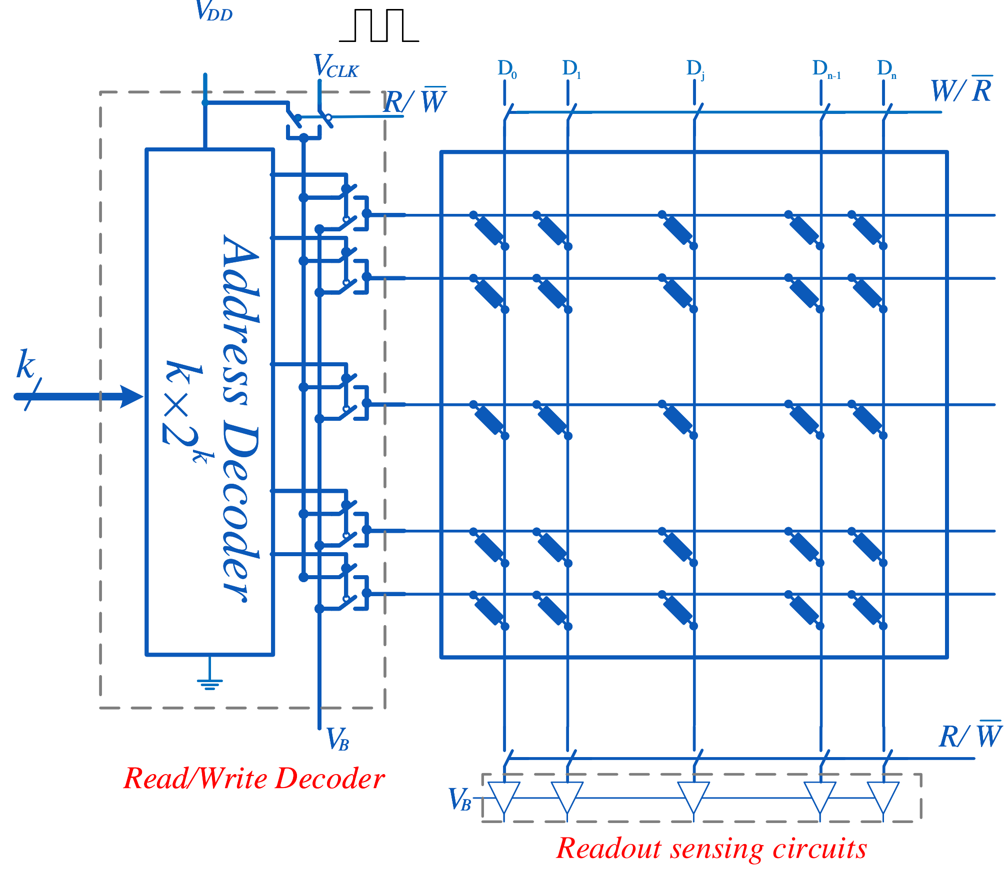


Figure 4.19: The whole schematic the resistive memory based on one step reading and writing techniques.

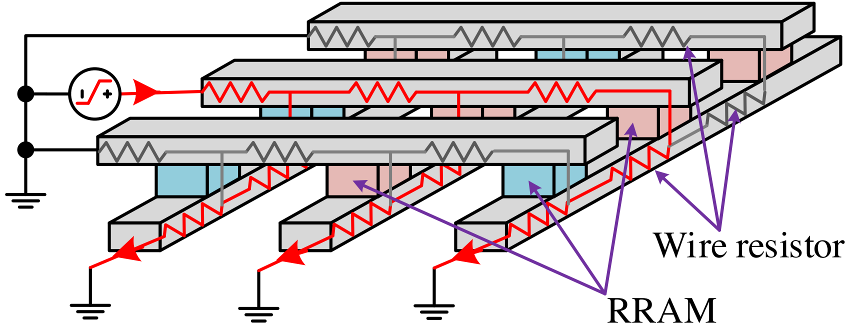


Figure 4.20: Parallel reading of the entire row in the crossbar. The columns and rows are grounded, except the row being read. The red arrow shows the sensed current flowing through wire resistances and RRAMs.

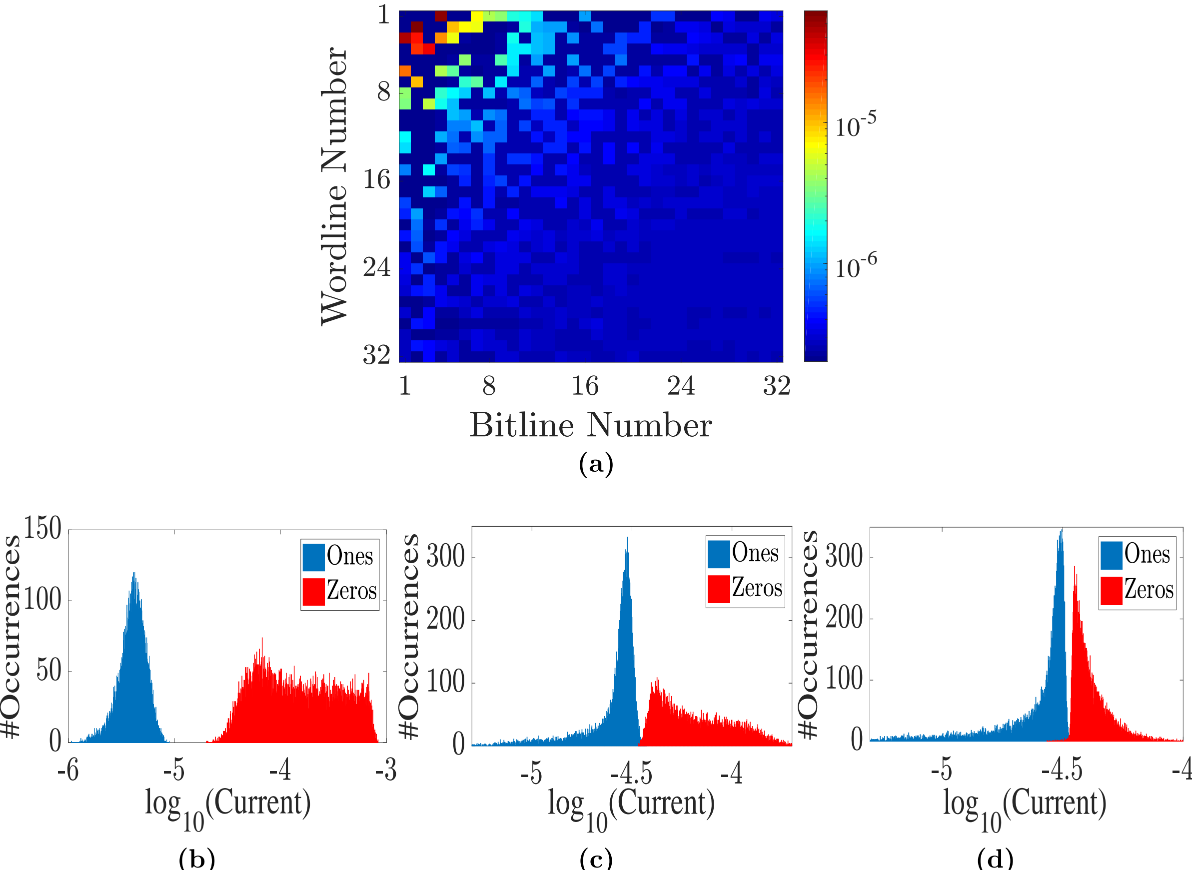


Figure 4.21: (a) Measured current per cell, and its histogram for bitline number (b) 1, (c) 16, and (d) 32.

Due to sneak path, the sensed current of low resistance state decreases in both vertical and horizontal directions in the array. The top left cells have distinguishable distributions for the stored ones and zeros. On the other hand, bits in the right-bottom cells are indistinguishable due to the read margin overlap. Fig. 4.21b, Fig. 4.21c and Fig. 4.21d show the histogram of the measured currents of the 1st, 16th and the 32nd bitline (column), respectively. Clearly, the larger the bitline index is, the more errors occur. We can see that the channels of the cells have varying reliability.

Addressing the sneak path problem has attracted a lot of interest from both research and industry communities. Proposed solutions include hardware-based approaches, e.g., transistor gating [47], and/or techniques based on communication and coding theory [110, 111].

The focus of this chapter is on the latter approach and our scheme is based on polar codes. Polar codes [112] are the first family of explicit error-correcting codes to provably achieve the capacity of binary symmetric channels, with a low-complexity encoding and successive cancellation decoding (SCD). For a code of length N, encoding/decoding has a complexity of $O(N \log N)$. For the above reasons, polar codes constitute an attractive error correction scheme.

Notation: Vectors are denoted with lower-case bold letters. A permutation π over the integers $\{0, \ldots, N-1\}$ is denoted as $\pi = [\pi(0), \ldots, \pi(N-1)]$, where $\pi(j)$ is the image of j under π. For a vector $\mathbf{x} = [x_0, \ldots, x_{N-1}]$, $\mathbf{x}_\pi$ denotes the vector $\mathbf{x}_\pi = [x_{\pi(0)}, \ldots, x_{\pi(N-1)}]$.

4.6.1 Non-stationary polar code construction

For a binary-input discrete memoryless channel (B-DMC), W, with output alphabet $\mathcal{Y}$, we denote its transition probabilities by $W(y|x), x \in \{0,1\}, y \in \mathcal{Y}$, and define the symmetric channel output probability as $W(y) = \frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)$. Define the symmetric capacity $I(W)$ as

$$I(W) \triangleq \sum_{y \in \mathcal{Y}} \sum_{x \in \{0,1\}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{W(y)}. \tag{4.13}$$

Polar codes [112] manufacture out of N independent copies of a given B-DMC channel, a second set of N *synthesized* binary-input channels. The channels show a polarization effect, namely, as N becomes large, the symmetric capacities of the synthesized channels tend towards 0 or 1 for all but a vanishing fraction.

In our framework, in contrast to the original polar codes, we consider the extension of polar codes to the setting where the underlying channels are of *varying reliability levels* (see Fig. 4.22).

Using the terminology in [113], we refer to such polar codes as non-stationary polar codes.

The basic polarization transformation is applied to two independent channels $W_0 : \{0,1\} \rightarrow \mathcal{Y}_0$ and $W_1 : \{0,1\} \rightarrow \mathcal{Y}_1$, resulting in two channels, $W' : \{0,1\} \rightarrow \mathcal{Y}_0 \times \mathcal{Y}_1$ and $W'' : \{0,1\} \rightarrow \mathcal{Y}_0 \times \mathcal{Y}_1 \times \{0,1\}$, given by

$$\begin{aligned} W'(y_0, y_1|u_0) &= \frac{1}{2} \sum_{u_1 \in \{0,1\}} W_0(y_0|u_0 \oplus u_1) W_1(y_1|u_1), \\ W''(y_0, y_1, u_0|u_1) &= \frac{1}{2} W_0(y_0|u_0 \oplus u_1) W_1(y_1|u_1), \end{aligned} \tag{4.14}$$

where $y_0 \in \mathcal{Y}_0, y_1 \in \mathcal{Y}_1$, and $u_0, u_1 \in \{0,1\}$. We denote this single-step transformation by $(W_0, W_1) \mapsto (W', W'')$.

The transformation preserves the average symmetric capacity, while exhibiting a polarization effect. Suppose $(W_0, W_1) \mapsto (W', W'')$. Then,

$$\begin{aligned} &I(W') + I(W'') = I(W_0) + I(W_1), \\ &I(W') \leq I(W_i) \leq I(W''), i = 0, 1. \end{aligned} \tag{4.15}$$

The Bhattacharya parameter of a binary discrete memory-less channel $W : \{0,1\} \rightarrow \mathcal{Y}$, denoted by $Z(W)$, is a measure of the reliability of W, and has been used to bound the error probability of polar codes in [112]. The parameter $Z(W)$ is defined as

$$Z(W) \triangleq \sum_{y \in \mathcal{Y}} \sqrt{W(y|0)W(y|1)}.$$

4.6.2 Polar Code Encoding and Decoding

Let $\mathbf{u} = [u_0, u_1, \ldots, u_{N-1}]$ and $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]$ be the input and the output of a length-N polar code, respectively, with $N = 2^n$ for some integer n. The encoding of polar codes is

given by

$$\mathbf{x} = \mathbf{u}G, \quad G = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n},$$

where the symbol $^{\otimes n}$ denotes the n-th Kronecker power operator.

After polar encoding, one obtains N synthesized channels $\{W^{(i)} \triangleq W_{n,i}, 0 \leq i \leq N-1\}$. A polar code of dimension k transmits k information bits in the k synthesized channels with the highest $I(W^{(i)})$ (we denote the corresponding information set by $\mathcal{I}$), and $N-k$ arbitrary but fixed bits in the remaining $N-k$ synthesized channels (denoted by $\mathcal{F}$). Decoding of polar codes is carried out using successive cancellation decoding (SCD) as in [112], taking into account the appropriate likelihood ratios of the original channels.

Role of the Channel Ordering

The performance of polar codes depends on the synthesized channels of the information set $\sum_{i\in\mathcal{I}} I(W^{(i)})$ [112]. As illustrated in Fig. 4.22, to construct a non-stationary polar code, we propose to apply a permutation π to the vector $\mathbf{x}$ in order to enhance the overall performance. Ideally, we want to find a permutation π^* such that for all permutation π,

$$\sum_{i\in\mathcal{I}_{\pi^*}} I(W_{\pi^*}^{(i)}) \geq \sum_{i\in\mathcal{I}_{\pi}} I(W_{\pi}^{(i)}), \tag{4.16}$$

where $I(W_{\pi}^{(i)})$ is the symmetric capacity of the i-th synthesized channel under permutation π and $\mathcal{I}_{\pi}$ is its information set.

The permutation π is defined such that $\mathbf{z}_{\pi} = \mathbf{x}$, or equivalently $\mathbf{z} = \mathbf{x}_{\pi^{-1}}$. Then, $z_{\pi(i)} = x_i$, implying that symbol x_i goes through channel $W_{\pi(i)}$, for $0 \leq i < N$. Correspondingly, a reverse permutation is required for the output of the channels. The polar decoder receives

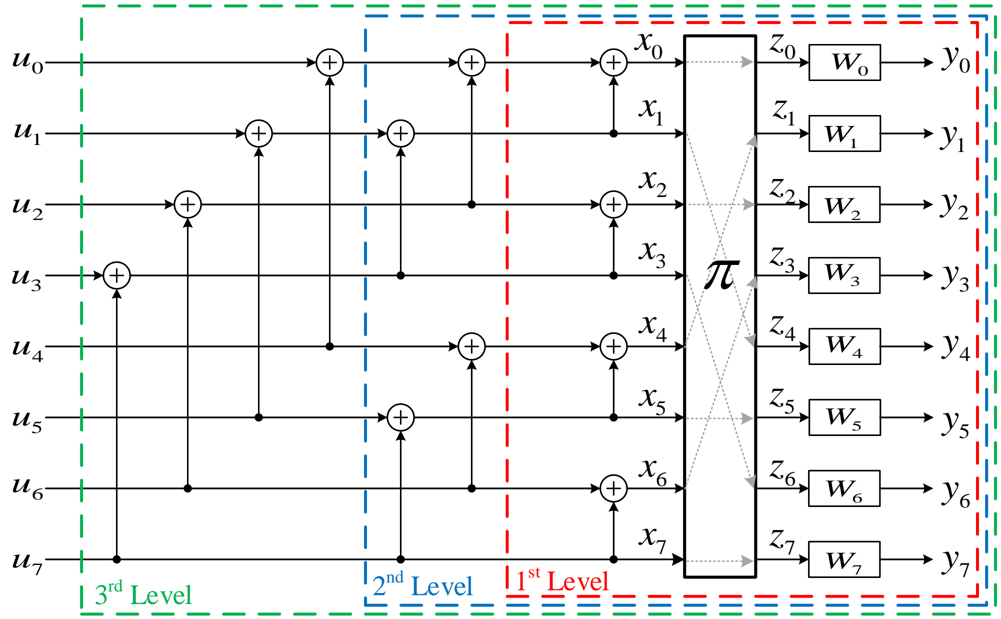


Figure 4.22: Polar encoding with $N = 8$ channels, with permutation $\pi = [0, 4, 2, 6, 1, 5, 3, 7]$.

as input the vector $\mathbf{y}' = [y_{\pi(0)}, \ldots, y_{\pi(N-1)}] = \mathbf{y}_\pi$. See Fig. 4.23 for a schematic picture of the full encoder and decoder.

We define ψ to be the bit-reversal permutation. For each integer $i \in \{0, \ldots, 2^n - 1\}$, $\psi(i)$ is the integer obtained by reversing the binary representation of i. That is, let $i = \sum_{j=1}^{n} b_j 2^{j-1}$, then $\psi(i) = \sum_{j=1}^{n} b_j 2^{n-j}$. As an example, when $N = 8$, $\psi = [0, 4, 2, 6, 1, 5, 3, 7]$, as illustrated in Fig. 4.22.

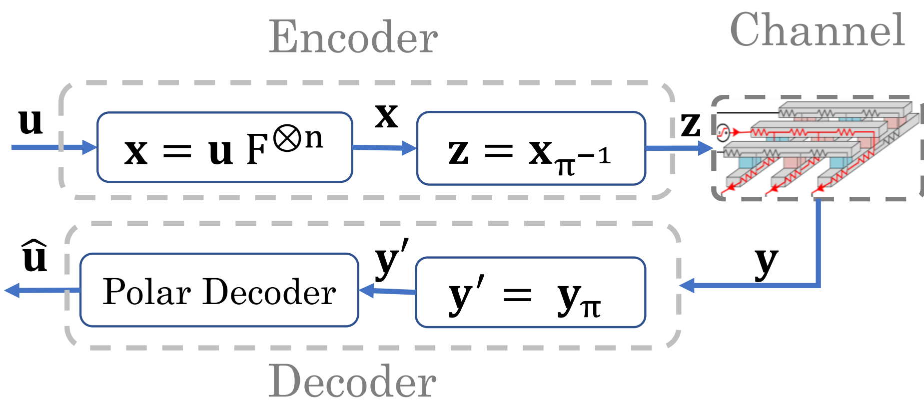


Figure 4.23: Full system model.

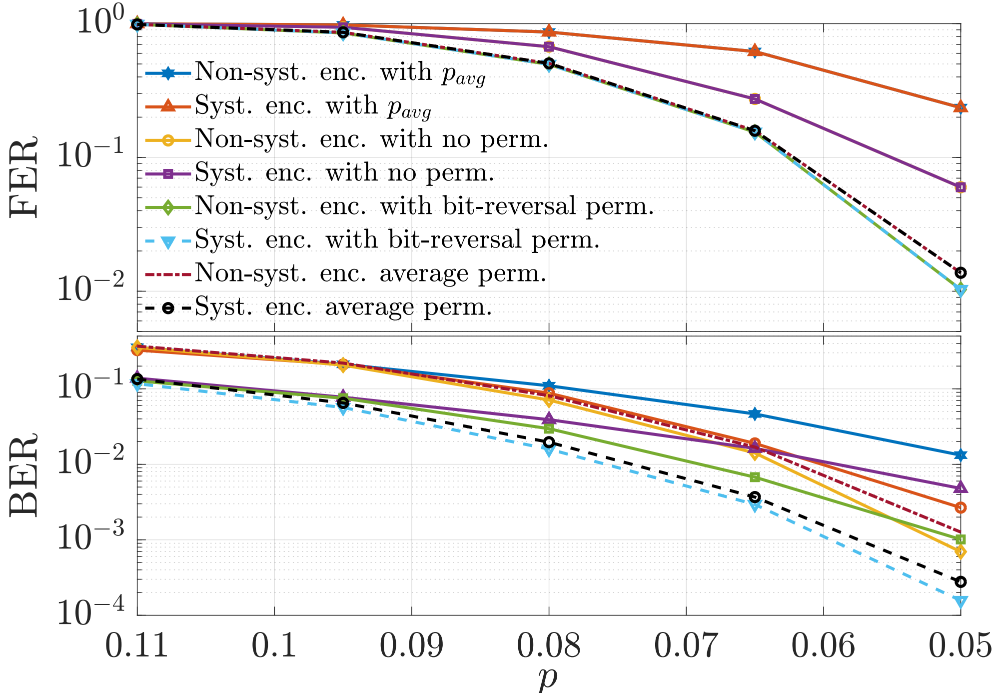


Figure 4.24: Performance evaluation for BSCs with linearly spaced cross-over probabilities. $N = 1024, k = 512$.

We evaluate the performance with the default ordering (no permutation) and the performance with the bit-reversal permutation. We also evaluate the performance obtained by averaging 200 random permutations. For each scenario, we evaluate the frame error rate (FER) and the bit error rate (BER) with non-systematic encoding and systematic encoding [114], for 10^4 runs. Fig. 4.24 illustrates the results. Similar to [114], we observe that systematic encoding of non-stationary polar codes enhances the BER performance compared to non-systematic encoding, while keeping the FER unchanged. The regular polar code exhibits the worst BER, while the non-stationary polar code under systematic encoding with $\pi = \psi$ performs the best. In particular, the latter scenario outperforms all 200 random permutations for all values of p.

Based on the observation that bit-reversal permutation achieves competitive BER numerically, we choose to apply $\pi = \psi$ for our non-stationary polar codes, so as to mitigate the sneak path problem in crossbar arrays.

4.6.3 Application to Resistive Crossbar arrays

General Framework

As outlined in the introduction, the crossbar array cells exhibit different reliability levels. For this reason, we propose the application of non-stationary polar codes to address the problem. We apply a two-step approach:

1. We first estimate a single detection threshold for each wordline (row) to minimize the overall uncoded BER per word. This transforms the read channels into BSCs. The threshold for each wordline is estimated by generating large training data and then applying a good binary clssifier. For instance, we observe that a logistic regression-based classifier gives superior performance in terms of accuracy and speed.
2. Based on the estimated thresholds in step 1), we estimate the cross-over probabilities of each cell in the array. We can then apply non-stationary polar codes using the cell characterizations.

Assuming the crossbar array size is $(N_1 \times N_2)$, then, the blocklength is $N = N_1 N_2$. The encoded output symbol $z_{in+j}, 0 \leq i < m, 0 \leq j < n$, is stored at the (i, j)-th entry in the crossbar array (i.e., we vectorize the array row by row).

Instead of using high-level models for the sneak path problem, such as in [110, 111], we use a SPICE-like simulator that is built based on accurate modeling of the resistive crossbar array [5]. This numerical simulator offers a fast alternative to SPICE simulators while maintaining the same simulation accuracy. In our simulations, the high resistance state, representing 1, and the low resistance state, representing 0, are set to $1M\Omega$ and $1k\Omega$, respectively.

In Fig. 4.25, we simulate the BER performance of systematic polar codes for four cases: (i) equivalent regular polar codes correspond to BSCs with parameter p_{avg}, (ii) permutation

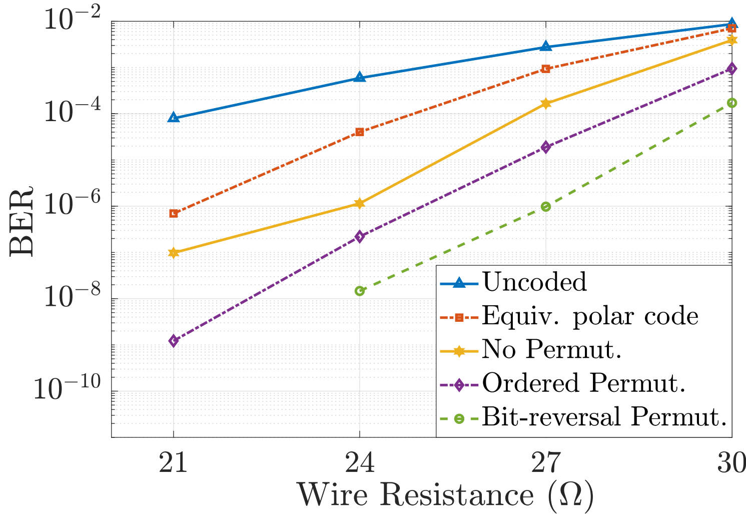


Figure 4.25: Performance evaluation for a (32×32) crossbar array, code rate $k/n = 0.8$.

$\pi = [0, \ldots, N-1]$, (iii) permutation π_{ord}, and (iv) permutation $\pi = \pi_{\text{ord}} \circ \psi$. Clearly, the BER permutation under $\pi = \pi_{\text{ord}} \circ \psi$ outperforms the other permutations.

4.6.4 Binary Asymmetric Channel Modeling

In subsection 4.6.3, the array cells are modeled as BSCs. Analyzing further the uncoded error distribution, as one may infer from Figures 4.21b, 4.21c, and 4.21d, we find that the conditional error distributions under 0's and 1's are different, i.e., $P(\text{error}|0) \neq P(\text{error}|1)$. Taking this observation into consideration, we model the crossbar array cells as binary asymmetric channels (BACs) and apply the non-stationary polar codes developed in Section 4.6.1.

In Fig. 4.26, we compare the systematic BER performance under both BSC and BAC modeling. As expected, the BER performance under the more accurate BAC model is higher, and the gain increases as the wire resistance decreases.

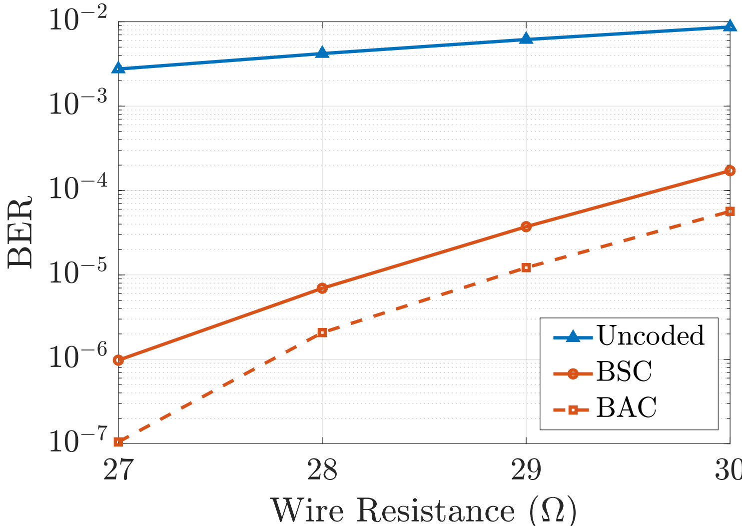


Figure 4.26: BER performance under BSC and BAC modeling for a (32×32) crossbar, code rate $k/n = 0.8$.

4.6.5 Punctured Polar Codes

In this subsection, we propose a technique that can enhance the BER performance in some scenarios by biasing the fraction of high resistance cells. The sneak paths exist through the cells having low resistances, causing inter-cell interference [110]. Intuitively, having more high resistances in the array helps mitigate the sneak path problem. To leverage this intuition, we investigate the use of a (punctured) polar code of a shorter length, say $N - N_p$, while storing high resistances in the corresponding punctured N_p cells in the array. Clearly, there is a trade-off between two opposite factors: puncturing reduces the number of redundant codeword symbols, hence degrading the performance of polar codes, while high resistances decrease the sneak path effect resulting in fewer read errors. In the following, we investigate the application of the above approach to the crossbar array.

A *punctured* polar code is obtained from the N-length parent polar code using a puncturing vector $\mathbf{w} = [w_1, \ldots, w_N]$, with $w_i \in \{0, 1\}, i = 1, \ldots, N$, where the 0s imply the punctured positions. We note that the information set $\mathcal{I}$ should be recomputed, when we consider puncturing. Punctured polar codes have been investigated by many works, and several

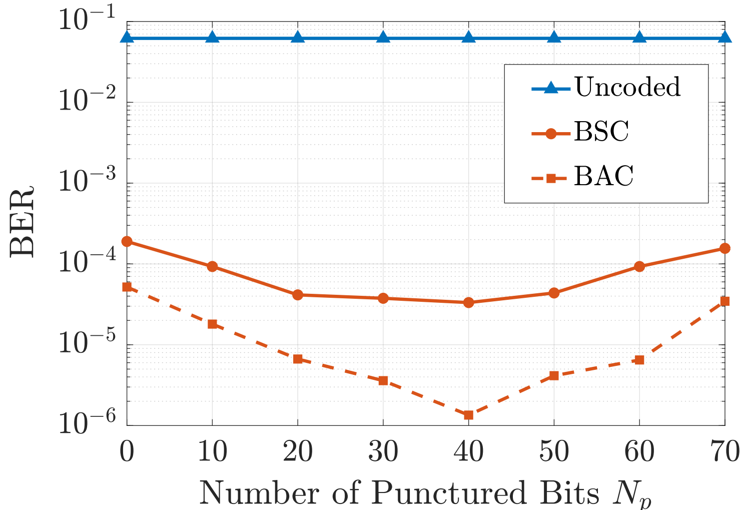


Figure 4.27: Punctured polar encoding over the (32×32) crossbar array for code rate $k/n = 0.8$.

efficient puncturing patterns have been proposed in the literature [115].

In [115], the authors proposed an empirically good puncturing algorithm, termed quasi-uniform puncturing (QUP). QUP-polar codes were shown through simulations to outperform the performance of turbo codes in WCDMA (Wideband Code Division Multiple Access) or LTE (Long Term Evolution) wireless communication systems in the large range of code lengths. We adopt QUP as our puncturing pattern and we highlight its advantages below.

Fig. 4.27 illustrates the systematic BER performance of QUP for a (32×32) array with $R_w = 35\Omega, \pi = \pi_{\text{ord}} \circ \psi$. We observe that the BER decreases as more bits (N_p) are punctured and as the frequency of 1's increases. This gain is reversed for $N_p > 40$ and the BER increases as N_p and the codeword redundancy decrease. The BER is improved by a factor of 5.7 for a symmetric channel model and by a factor of 38.5 for an asymmetric channel model.

4.7 Conclusion and Future Perspective

The proposed reading technique with the readout circuitry can read the entire row without using reference bits giving the maximum utilization of RRAM and the highest throughput

for high speed applications. These advantages come with more power consumption (4.7X more than [12]). According to the defined FOM taking into consideration, power consumption, array usage, effective array size, and throughput, the proposed technique is more than 100X better than [12] without banking. Moreover, according to the discussed studies for the sneak path immunity, power consumption and bias mismatch, the nonlinear devices are most recommended for high dense resistive memories. In addition, the proposed technique is compatible with the published writing techniques [116] where switches are placed around the array to enable reading or writing since reading and witing can not be performed simultaneously in the same array.

One of the features of the resistive memories is its ability to store multi-levels/multi-states such as ternary and quaternary data enabling higher higher radix processing units. The proposed technique can be applied for multilevel memories as well due to its ability to read the device resistance especially with nonlinear switching devices. However, the readout circuitry needs to modified to accommodate the multi-states and be able differentiate between them. This topic will be investigated in future research.

Stack-ability of resistive memories is another feature enabling ultra dense memory arrays [117]. The discussed readout technique alongside the circuitry can be used to read each crossbar layer by connecting the corresponding crossbar outputs together then to readout circuits as shown in fig. 4.28. A level decoder is needed to select the readout level. Using this configuration, only one row in a certain level is selected at a time where the other outputs result in zero output current. The stacked layers share the same reading circuitry which decreases the overhead of readout circuits. For instance, Xpoint memory is 2 layers resistive memories sharing the same bitlines and having two different wordlines [118]. The proposed readout circuitry can be connected to the bitlines and the wordlines are used to access the memory cells. The power density is one of the important aspects of any electronic circuit. By using the aforementioned technique, the power density is approximately constant due to

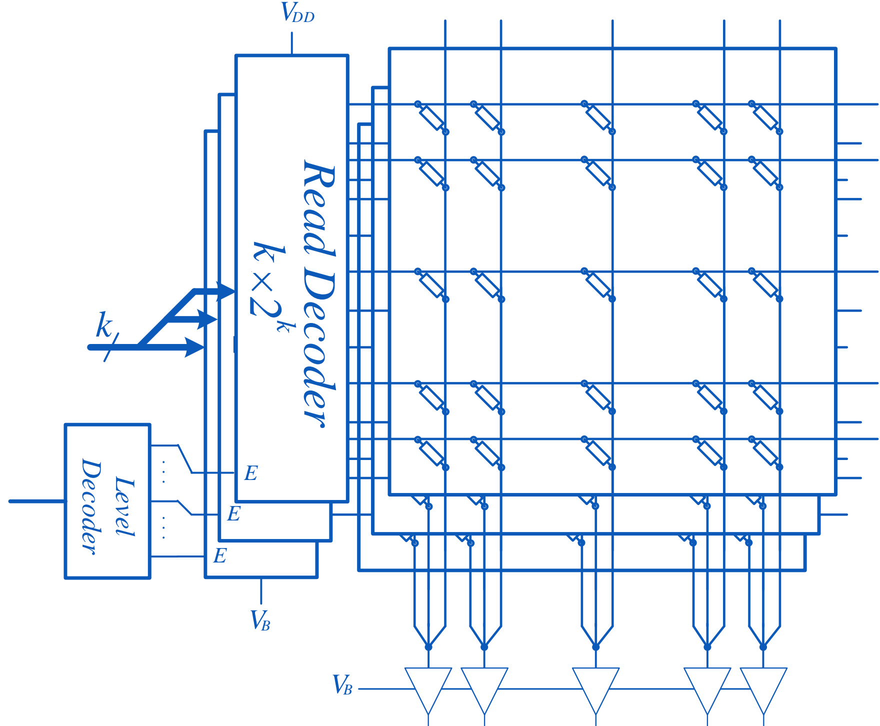


Figure 4.28: Stacked crossbar arrays sharing the readout circuitry.

reading only one row per level at time. However, stack-ability might cause other reliability issues which limits the number of stacked layers, and is currently a subject of intensive research by the community.

In future work, the BER and power results will be extended to take into account the effect of wire resistance which highly affect the performance of the crossbar array [5]. Furthermore, power will be included in the objective function to be optimized as well to find the optimal bias scheme in terms of both power and BER jointly.

Motivated by the sneak path problem in resistive memories, we studied polar coding over channels with different reliability levels. In particular, we argued that the channels' ordering is important and proposed a channel ordering whose attractive performance was shown numerically. We then applied our framework to the sneak path problem in resistive memories. Simulation results on SPICE-like resistive crossbar showed significant bit-error rate performance improvement, especially for low uncoded BER. Additionally, we proposed two

approaches to further lower the BER. The first approach relies on a more accurate channel modeling. The second approach consists in biasing the frequency of high-resistance values in the array so as to mitigate the sneak path occurrences. We note that while in this work, we modeled each cell individually as a BSC (or a BAC), the cost of such modeling is amortized by using the same characterization over several crossbar arrays, which makes the model parameter costs justifiable from a practical perspective. Moreover, it is possible to *cluster* multiple cells together in a way to reduce the number of overall crossbar model parameters.

The present work represents another step toward coding for the resistive crossbar arrays. The performance of polar codes can be improved by using more enhanced decoding algorithms (e.g. list-decoding [119]), at the expense of higher complexity. Moreover, by the sneak path nature, errors in neighboring cells are not independent. A future direction for research is to investigate enhanced polar decoding algorithms, taking into consideration such correlations. Another avenue for research is to investigate other techniques for biasing the distribution of high-resistance values, along with techniques for coding for asymmetric channels as in [120].

Chapter 5

Wire Resistance-aware Training for efficient Offline Learning

Memristive crossbar arrays, also known as RRAM (resistive random access memory) crossbar arrays, have been suggested as a way to achieve high cost and energy-efficient implementations of matrix multiplication, a crucial operation used in many applications including deep learning [65]. While the previous work in this area typically uses a simplified model of crossbar arrays, in order to tackle large applications such as deep neural networks (DNNs), it is important to consider physical realities or RRAM's non-idealities such as nonlinear and asymmetric potentiation and depression, device variability, and device-to-device mismatch [65, 121].

In particular, a recent analysis using a detailed model [5] reveals that in the nanoscale era *wire resistance* in a crossbar array results in non-negligible sneak path issues (see Section 5.3). In addition, the previous works, such as [122, 70], showed the high degradation in the performance due to the existence of the wire resistance. Over the past few years, a number of techniques have been proposed to mitigate the effects of sneak path for memory applications, where most solution assume that only one wordline is accessed at any given moment (as is expected in typical memory operations). However, these solutions are not applicable to crossbar-based *computing* applications such as DNNs, where typically all cells

of the entire array are accessed simultaneously. Furthermore, accurate modeling using circuit simulation tools that includes wire resistance becomes computationally challenging due to the scale associated with DNNs.

5.1 Background and Related Work

Crossbar arrays offer a fast vector matrix multiplication (VMM) platform which reduces VMM's complexity to $O(1)$ instead of $O(N^2)$ using conventional approaches. While continuous memristors (RRAMs) show good potential for full precision neural networks, current device fabrication methodologies are not mature enough to support fine tuning of the device's resistance, endurance and variability. Under these conditions, a direct weight transfer of a trained neural network would fail unless sophisticated simulators with accurate device and interconnect models are used to model these non-idealities. Alternatively, training could be performed via in-situ learning (online learning), where the training is done directly on the devices, however this is only possible for small neural networks [123, 62].

The high compute-intensity of DNNs has motivated hardware DNN processors using digital and mixed-signal CMOS as well as emerging technologies [124, 125, 126]. Recently, several works show through simulation and hardware implementation that RRAM crossbar-based hardware neural networks can be trained with very high accuracy – typically with less than 1% drop in accuracy for networks designed for MNIST and CIFAR-10. In [127, 128, 129], the authors use 1T1R crossbar structure and proposed two approaches; 1) sequential BNNs where the current sensing circuit (neuron) is shared between all the neurons so only one column is activated at a time and 2) parallel BNNs where a current sensing circuit is assigned for each neuron. In [130], binary RRAM-accelerated CNN is designed and optimized to feature massive parallelism with high energy efficiency. In [131], an implementation of a multi-bit binary conventional neural network was introduced using selector-less crossbar arrays with

pipelined implementation included the device variations. Moreover, [132] presents a full hardware implementation for robust RRAM-based convolutional block using single-ended XNOR sensing capable of performing dot product operations in a single cycle. This block can be used for computer vision and image processing applications. Although these works provide good implementation techniques of BNNs using RRAMs, the effect of the wire resistance has not been considered which is inevitable in the crossbar arrays and would highly degrade the performance especially for both 0T1R arrays and 1T1R with parallel computations.

On the other hand, circuit level mitigation techniques of the sneak path problem, such as [133], are useful only for memory applications where one cell is read at a time where the sneak path problem arises from the multi-path effect occurred reading technique. But in case of the vector matrix multiplication in neural networks, the sneak path happens because of the wire resistance which creates residual voltages across the array. These residual voltages create leakage currents which disturb VMM operation.

5.2 Multi-bit RRAM Device Under Study

The authors in [4] demonstrated the fabrication of Au/Al_2O_3/HfO_2/TiN RRAM device with a junction area of $10 \times 10 \mu m^2$ patterned via photolithography and followed by a wetetching process. The thicknesses of the top electrode (Au) and the bottom electrode (TiN) were 150 and 100 nm, respectively, and the switching material thicknesses are 2 and 6 nm for Al_2O_3 and HfO_2, respectively. This device was optimized to have self-compliance and gradual set-switching behavior and is capable of generating up to 16 states with good reliability.

To precisely program the device, an incremental step pulse programming technique with error correction is used. Starting from a low conductance state, incremental step pulses are applied to the device using a pulse generator until the device reaches the required state. Figures 5.1a and 5.1b show the gradual incremental-step programming and the current-voltage hysteresis

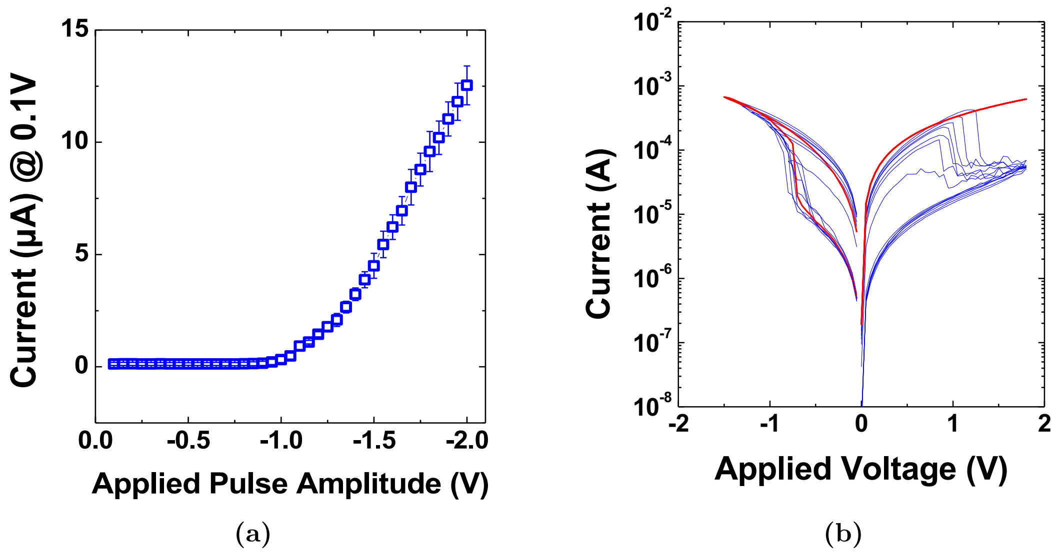


Figure 5.1: $Au/Al_2O_2/HfO_2/TiN$-based RRAM device adopted from [4] (a) device behavior under incremental step pulse programming and (b) current-voltage characteristics.

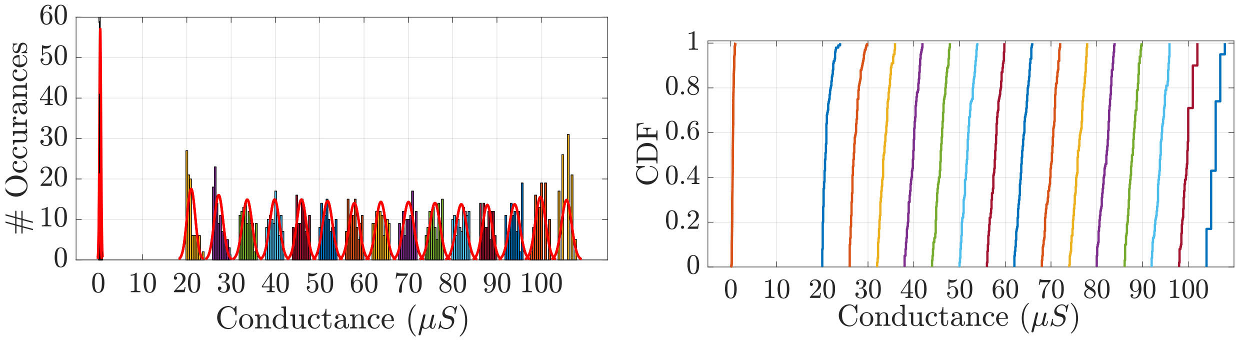


Figure 5.2: Histogram and cumulative distribution function (CDF) of 100 measured samples per state.

of the programmed device under different programming conditions. Figures 6.3b and 6.3a show the histogram and cumulative distribution function of the measured conductances, respectively. The measured samples are curve-modeled into a Gaussian distribution and we have found that the mean value of the device's conductance can be modeled as

$$G_i = 14 + 6 \times i \quad (\mu S) \tag{5.1}$$

where G_i is the ith highest conductance state for $i \in [1, 15]$ while the low conductance state is $46.7nS$.

There are two ways to integrate the device model in hardware simulation: (i) random sampling of the Gaussian model of each state, and (ii) random sampling from the measured data. In this work, we choose the latter, random sampling from the measured data, since the Gaussian distribution may not accurately describe the randomness of the device's states and device to device variations.

5.2.1 MVM using RCAs

RRAM crossbar arrays can perform the MVM operation, which is equivalent to n^2 multiply and accumulate (MAC) operations, with $O(1)$ time complexity compared to $O(n^2)$. The matrix is programmed/stored in the RRAM array cells as conductance values, and the input is applied as a voltage at the rows of the array. By grounding the columns of the array, the output current per column is proportional to the inner product between the input voltage vector and the conductance vector of the column, which can be written as

$$I_j = \sum_{i=1}^{m} G_{ij} V_i \tag{5.2}$$

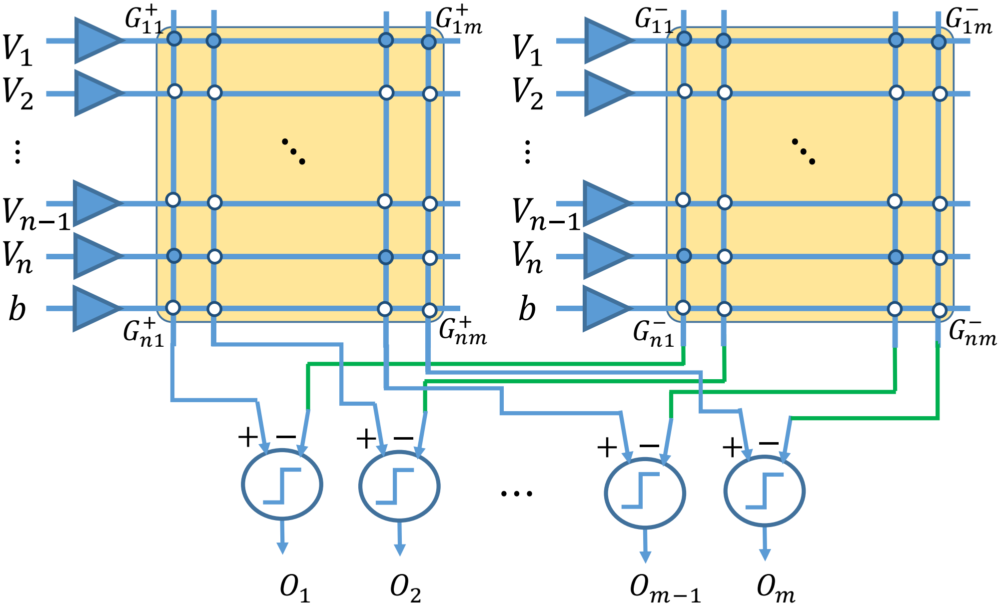


Figure 5.3: Matrix Vector Multiplication using separate RCAs.

where I_j is the current of the jth column (i.e. *post-synaptic current*), G_{ij} the synaptic weight in conductance, V_i the ith input voltage (i.e. pre-synaptic voltage).

The conductance of RRAM can only realize a positive value; however, both negative and positive weight realizations are needed in any neural network. In order to create negative weights, two weight realization techniques have been introduced: (i) using two RRAM cells per weight [61] as shown in Fig. 5.9, which is referred to as *balanced realization*, and (ii) using one RRAM as weight in addition to one shared reference RRAM with the conductance of $G_r = (G_{\max} + G_{\min})/2 \approx G_{\max}/2$, which is referred to as *unbalanced realization* [134, 135] where $G_{\max}$ and $G_{\min}$ are the minimum and maximum achievable conductances, respectively. In this work, we consider the first realization method, which has double the dynamic range (conductance range $\approx (-G_{\max}, G_{\max})$), making it less susceptible to noise and variability at the expense of doubling the area and power. The differential output current can be written as

$$I_j = \sum_{i=1}^{m}(G_{ij}^+ - G_{ij}^-)V_i = \sum_{i=1}^{m} G_{ij}V_i \tag{5.3}$$

where G_{ij} is the differential conductance and can be written as matrix-vector multiplication as follows

$$\mathbf{I} = (\mathbf{G}^{+} - \mathbf{G}^{-})\mathbf{V} = \mathbf{G}\mathbf{V} \tag{5.4}$$

where $\mathbf{V}$ is the input voltage vector (e.g. input image) and bias value. The current vector, $\mathbf{I}$, is sensed, and shaped by the activation function, which is mathematically described as

$$\mathbf{O} = f(\mathbf{G}\mathbf{V}) \tag{5.5}$$

where $f(\cdot)$ is the activation function.

In practice, a DNN layer can be too large to be realized in hardware using a single crossbar array. Thus, these large layers are partitioned into smaller crossbar arrays which are connected to perform MVM as a single layer [134, 136]. In this work, we partition each layer into differential 128×128 arrays, which is the same size as recently fabricated arrays [137]. In addition, bigger array sizes lead to worse sneak paths causing higher degradation in terms of performance as discussed in [134].

5.2.2 Quantized Weight Mapping

Each weight is translated into a pair of conductance values, which can be mathematically formulated as

$$G = G^{+} - G^{-} = \frac{W}{W_{\max}} \Delta G, \tag{5.6}$$

where $W_{\max}$ is the maximum value of the weight. If it is required to realize $W_{\max}$, G^{+} and G^{-} are set to $G_{\max}$ and $G_{\min}$, respectively, and $\Delta G = G_{\max} - G_{\min}$. The difference between the two conductance values is constant and proportional to the required weight value, and each

#bits	weight (W)	Mapping-I	Mapping-II	Range
1 bit	0 & ± 1	0 & $\pm G_{15}$	0 & $\pm G_1$	-
2 bit	0 & $\pm \frac{i}{2}$	0 & $\pm G_{8i-1}$	0 & $\pm G_{2i-1}$	$1 \leq i \leq 2$
3 bit	0 & $\pm \frac{i}{4}$	0 & $\pm G_{4i-1}$	0 & $\pm G_{4i-1}$	$1 \leq i \leq 4$
4 bit	0 & $\pm \frac{i}{8}$	0 & $\pm G_{2i-1}$	0 & $\pm G_{2i-1}$	$1 \leq i \leq 8$
5 bit	0 & $\pm \frac{i}{16}$	0 & $\pm G_{i-1}$	0 & $\pm G_{i-1}$	$1 \leq i \leq 16$

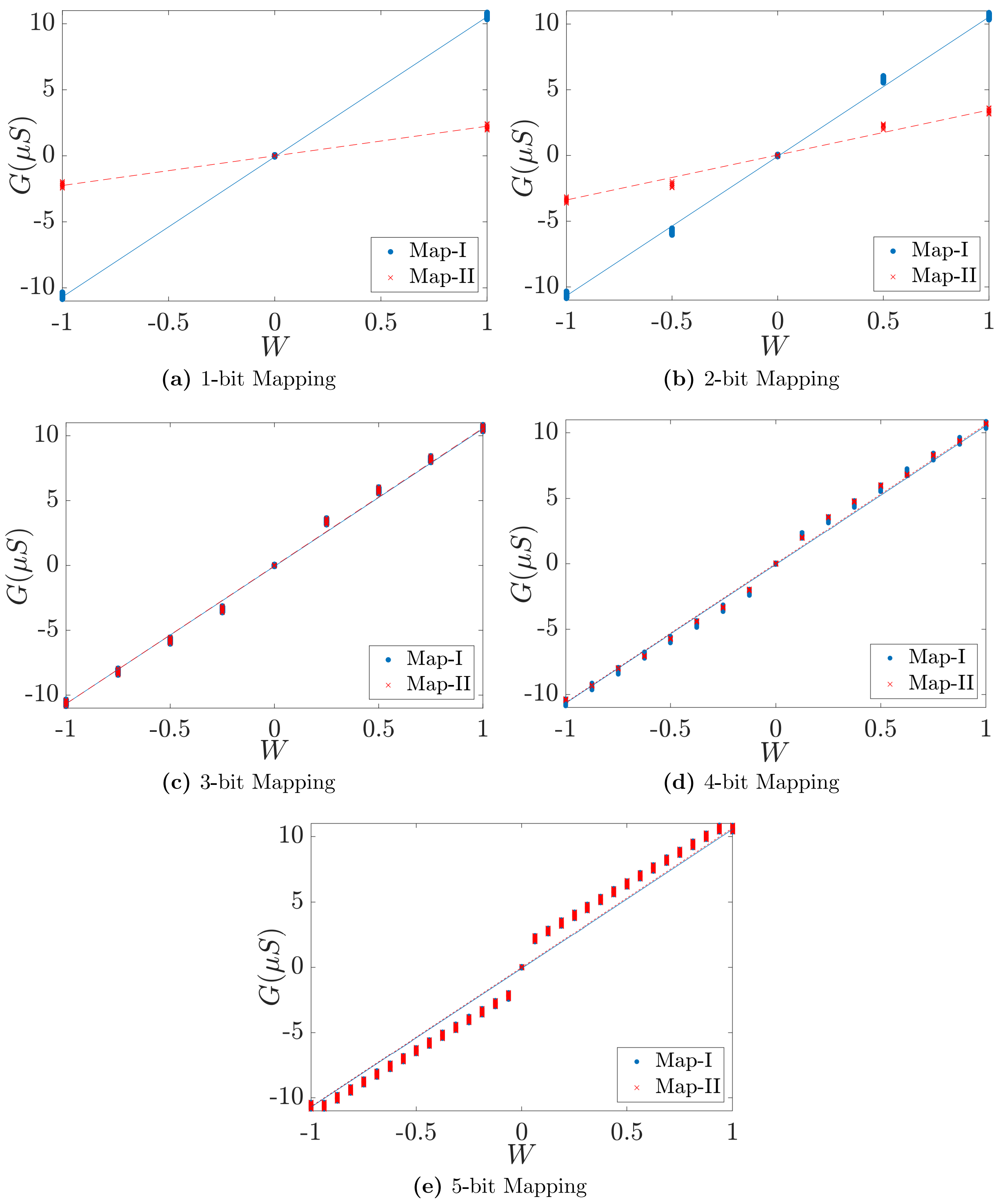
Table 5.1: Weight-conductance mapping for quantized states.

conductance is constrained to be between $G_{\min}$ and $G_{\max}$. Thus, there are many possible realizations for each weight; for example, the zero weight can be realized with any equal values of G^+ and G^-. In our realization, we set G^- and G^+ to $G_{\min}$ for positive and negative weights, respectively. By this setting, it can be shown that the power consumption during the inference is minimized, since the power consumption of RRAMs is directly proportional to the device conductance ($P = GV^2$).

Using the aforementioned 4-bit device, it is possible to realize up to 5-bit weight when two devices per weight are used. Table 5.1 and Figure 5.4 show the weight mapping from quantized weight to device's conductance. Clearly, there is a linear relation between the device conductance and the weight if chosen properly except for the 5-bit case because of the high gap between low conductance state and first high conductance state. Thus, in this work, we consider up to 4-bit quantized neural network.

5.2.3 Sensing Circuit Realization

Unlike sequential processors that require extremely high operational frequency, neural network processors exploit their inherent massive parallelism to enable high throughput computations. Thus, a current sensing circuit integrated with the activation function is vital, This circuit is replicated hundreds of times per layer. Thus, it should be low power and ultra-compact circuit. In this work, we use binarized activation function $\{-1, 1\}$ during the inference for simple and fast communication between the crossbar layers and eliminate



(a) 1-bit Mapping

(b) 2-bit Mapping

(c) 3-bit Mapping

(d) 4-bit Mapping

(e) 5-bit Mapping

Figure 5.4: Possible weight mappings for the used RRAM device.

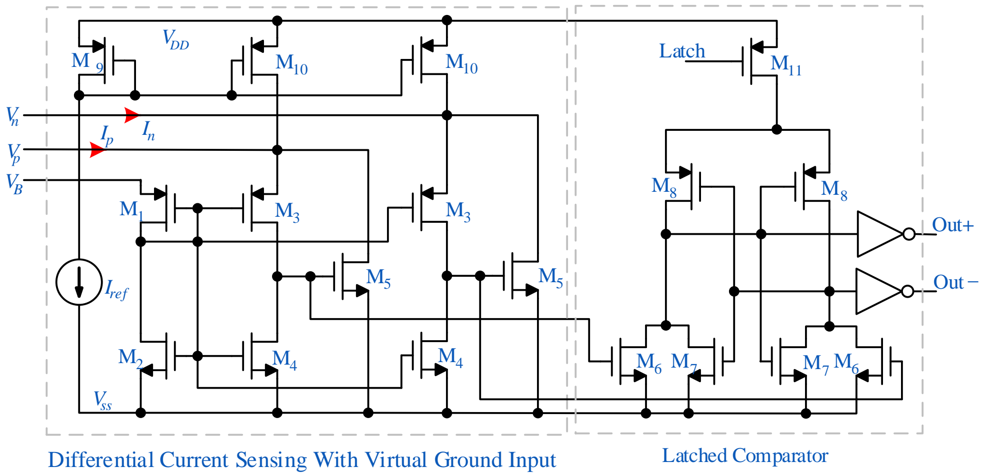


Figure 5.5: Schematic of proposed latched current sensing.

area- and power-expensive blocks such as ADCs and DACs in addition to simple buffer (i.e. driver) circuits can be used to support resistive loading.

In order to have accurate MVM operation, three conditions should be satisfied: 1) the output ports of the crossbar array should be biased to constant voltage (i.e. virtual ground), 2) the current sensing circuit should not load the crossbar array, and 3) the sensing circuit should be a transimpedance amplifier and nonlinearly shape the output voltage by the required activation function. In order to satisfy the aforementioned conditions, current conveyor principle is used. In [138], we proposed a single input circuit with sigmoidal activation function. By utilizing the same sensing concept, a differential circuit is designed with latched compactor shown in Fig. 5.5. The bias voltage V_b is mirrored to the other inputs creating constant bias voltage without loading the crossbar array. The detailed analysis and Monte Carlo simulations are discussed in details in [138]. The positive input is connected to the positive column (i.e. positive crossbar array) and the negative input is connected to the negative column (i.e. positive crossbar array). The input currents are absorbed and mirrored to the inputs of a simple latched circuit, which gives either V_{DD} or V_{SS} if the current absorbed from the positive crossbar array is greater or less than the current absorbed from negative crossbar array, respectively.

By latching the outputs of each layer, the overall operation of the neural network can be

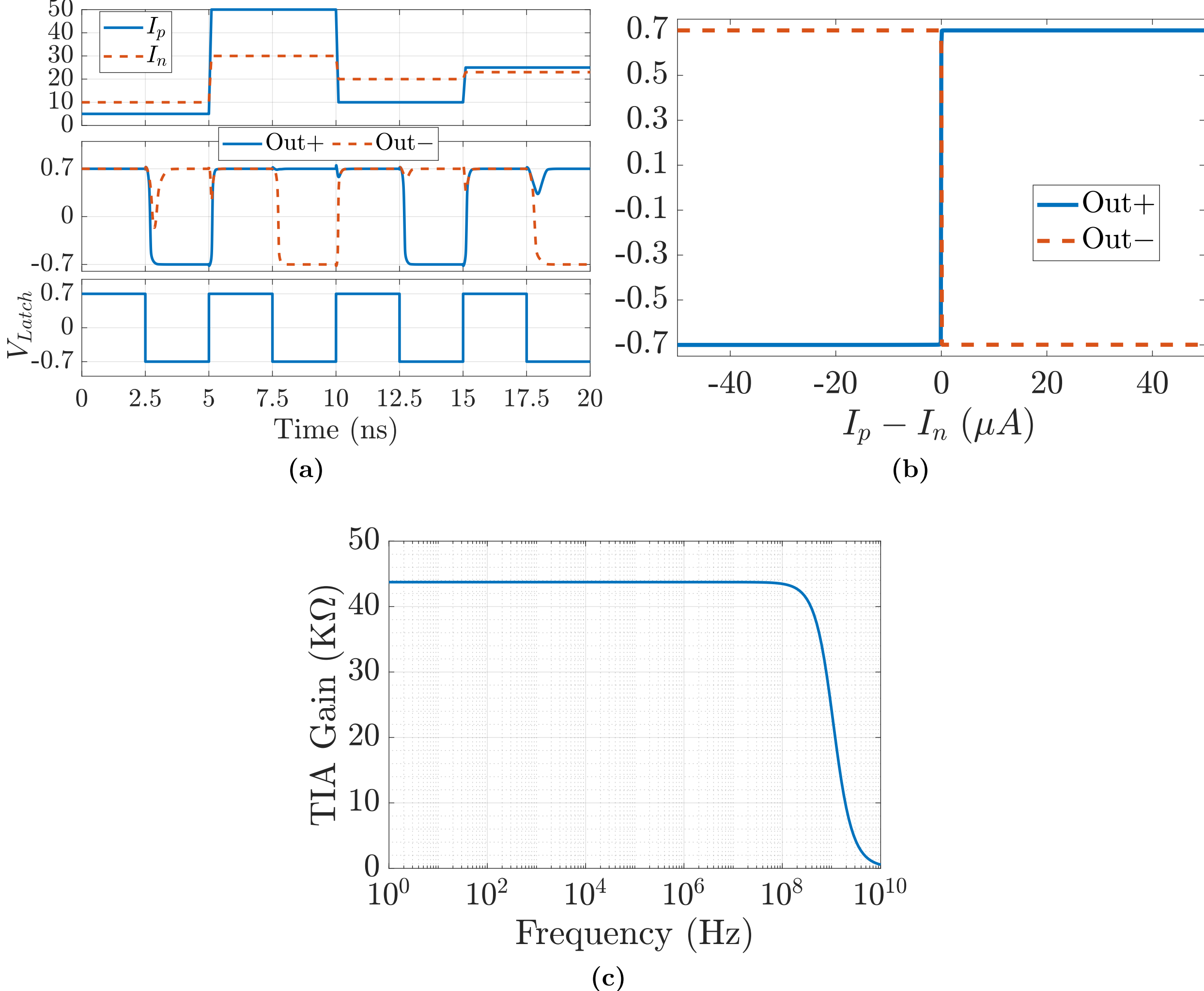


Figure 5.6: (a)Transient simulations for different Input currents, (b) output voltage versus differential input current characteristics, and (c) AC response of the transresistance gain.

pipelined which results in higher throughput. The circuit has been designed using TSMC 65nm technology. Figure 5.6b shows the response of the circuit with changing the positive input current for different negative input currents. The output-input curve exhibits hard switching relation which is needed for sign activation function. In addition, Figure 5.6c shows the AC response of the transresistance gain of the circuit. The 3db cut off frequency is found to be 747MHz, which is the maximum operating frequency. The total area of the sensing circuit is $15\mu m \times 6\mu m$.

5.3 Effect of Wire Resistance on Sneak Path Problem

5.3.1 Nano-scale Crossbar Parasitics

The dimensions of switching devices such as RRAMs or memristors are expected to be in range of a few nanometers [25], thus suffering from nano-scale effects caused by interconnect parasitics, namely, wire resistance, inductance, capacitance and conductance [5]. However, for most practical frequencies wire inductance and conductance can be neglected compared to the wire resistance and capacitance, respectively as discussed in detail in [5].

Interconnect Wire Resistance

Interconnect wire resistance is calculated as $R_w = \rho l/wh$ where ρ, l, w and h are the resistivity, wire length, wire width and wire height, respectively. The width and height of the wire are usually comparable and have a $1-2$ aspect ratio. The wire resistivity increases exponentially with technology scaling, not only because of the reduction in minimum wire dimensions, but also due to the increase in electron scattering at grain boundaries, surfaces, and interfaces as well. Thus, the resistivity is wire size dependent especially with wire width less than $100nm$ [88]. Another phenomenon that affects the wire resistance is the skin effect, but it was shown that it becomes non-negligible at THz range frequencies, thus not a concern at the MHz/GHz expected range of operation, [5]. In [5], two interconnects are considered and simulated; the first one for devices with feature size $50nm$ representing a recently published switching device and the other one with feature size $5nm$ which is the projection of ITRS for resistive crossbar arrays [25]. The wire resistance per cell was found to be 1.908Ω and 91.2Ω for $F = 50nm$ and $F = 5nm$, respectively, showing a $47\times$ increase.

Interconnect Wire Capacitance

The crossbar structure contains different types of parasitic capacitances; stray capacitance between the interconnect and the substrate, coupling capacitance between the interconnect wires, in addition to the device capacitance. The stray and coupling capacitances are in the range of few attoFarad and sub attoFarad for $50nm$ and $5nm$, respectively [5]. However, the main capacitance source is the device capacitance due to the high dielectric switching material. The device capacitance would be around $10 - 100aF$ [5]. The capacitance affects only the transient behavior of the crossbar array causing a delay for the signals. In this work, we focus on the steady-state behavior which is only affected by the parasitic resistances.

5.3.2 Source and Neuronal Resistance

The source resistance, R_{src}, represents the resistance from the input driver to the crossbar. On the other hand, the neuronal resistance, R_{nrn}, represents the resistance from the crossbar to the sensing circuit which working as a neuron that sums all the currents. The source and neuronal resistances depend only on the wire length connecting the crossbar array. The effect of the source and neuronal resistances with and without the wire resistance can be easily calculated by applying KCL rules where there is a negligible leakage from one neuron to the other for equally probable LRS and HRS. The sensed current at cell ij, with zero wire resistance, can be calculated by

$$I_{i,j} \approx \frac{2V_i R_{on}}{(R_{src} + R_{ij})(nR_{nrn} + 2R_{on}) + 2R_{nrn}R_{on}} \tag{5.7}$$

where V_i is the ith input voltage, R_{ij} is the target cell resistance and n is the number of synapses per neuron. The ideal case current is V_i/R_{ij}. Thus, the sensed current with the existence of the source and neuronal resistance is a scaled version of the ideal case.

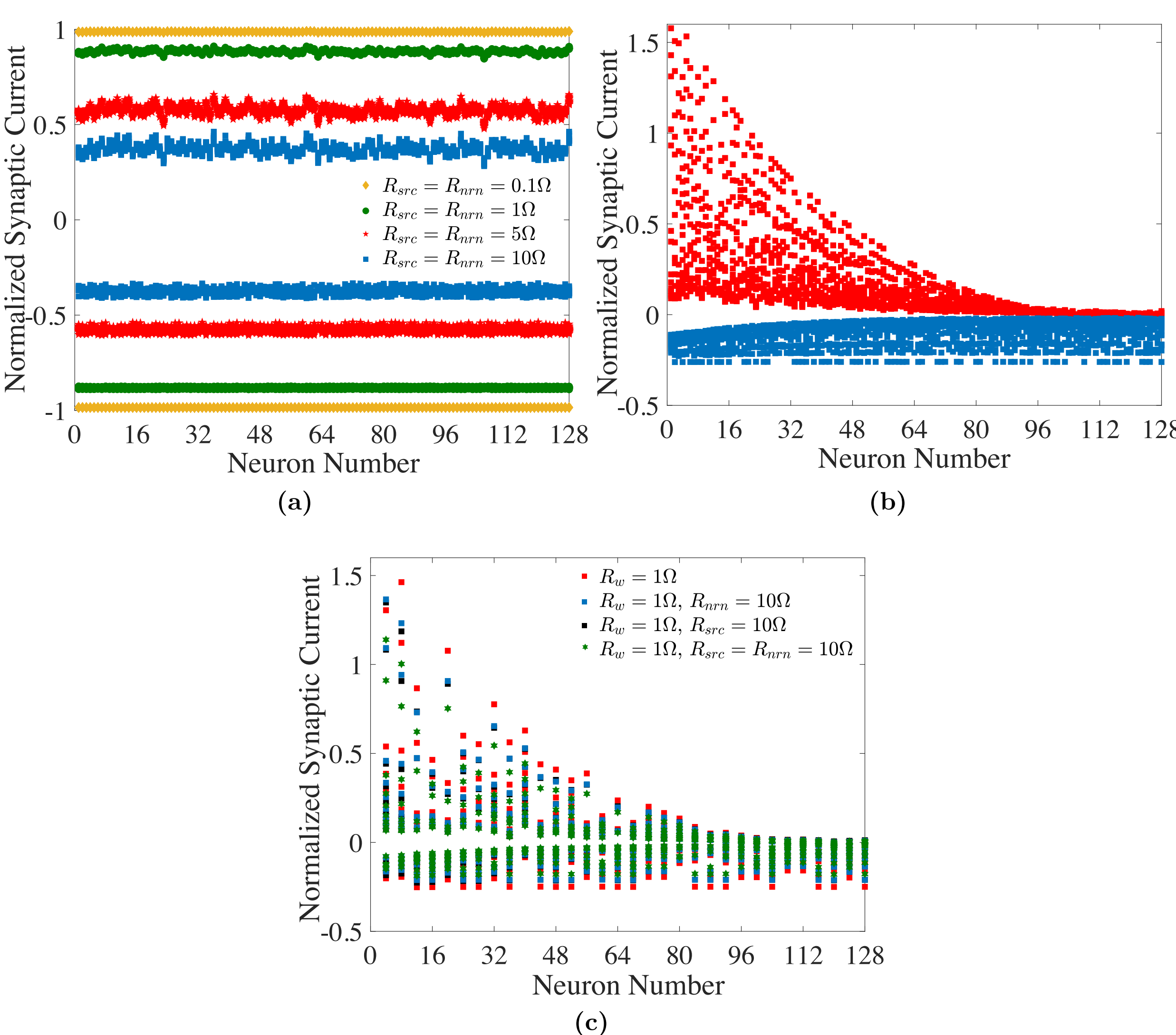


Figure 5.7: Effect of parasitic resistances on the normalized synaptic current

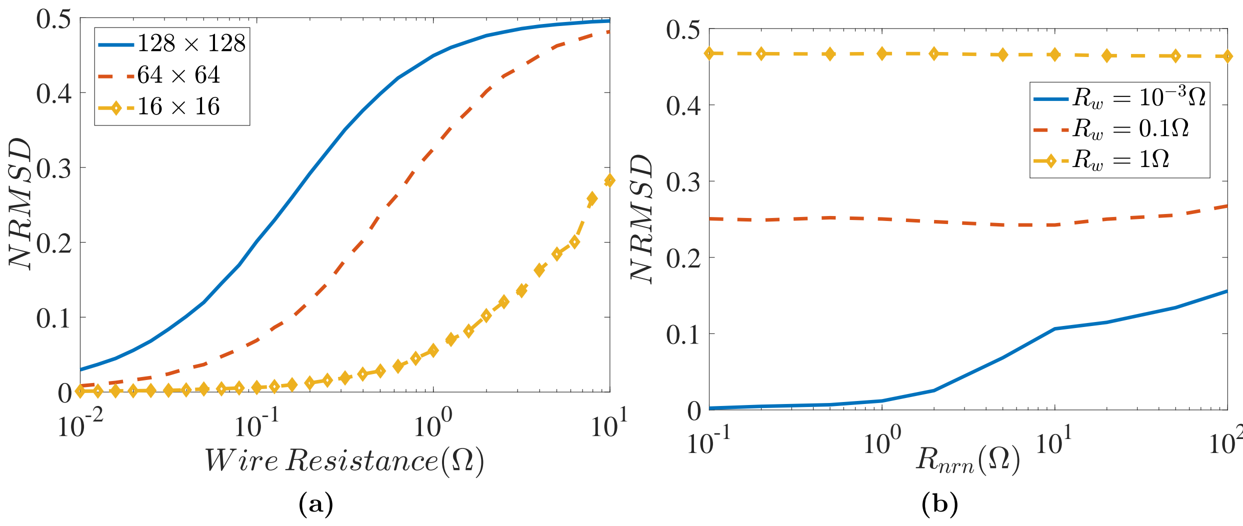


Figure 5.8: Normalized root mean square deviation of the sensed current compared to the ideal case versus changing (a) the wire resistance and fixing the neuron resistance for different crossbar arrays and (b) the neuron resistance for different wire resistances for 128×128 array.

5.3.3 Crossbar Simulation and Sneak Path Problem

To see the effect of wire, source, and neuronal resistances on the sneak path problem for passive RRAM arrays, we use a generic resistive network model that is as accurate as SPICE simulation [5]. The steady-state behavior of a crossbar array can be modeled as a resistive network, which can be parameterized using two main factors, $r = LRS/R_w$ and switching ratio, $K = HRS/LRS$, where LRS and HRS are the low and high resistance states, respectively, and R_w is the wire resistance per cell in addition to the source and neuronal resistances. LRS of a switching device varies depending on the switching and electrodes materials as widely as $100\Omega \sim 1M\Omega$ [139]. K is 100 or higher in good switching devices [27]. In this chapter, we vary the parameter r to study the effect of different wire resistance values, thus providing insight on the effect of wire resistance induced sneak path. But, in some cases that require a comparison, we use the following device parameters: $LRS = 1K\Omega, HRS = 1M\Omega, V_{set} = 1.1V$ and $V_{reset} = -1.3V$ for $Ta/HfO_2/Pd$ device which has linear switching behavior (i.e device's resistance is voltage independent)[137].

Figure 5.7a shows the effect of changing the source and neuronal resistances with zero wire

resistance. Clearly, these resistances attenuate the sensed current where it works as a current divider and is not a function of the neuron location. The more source and neuronal resistances are, the more attenuation appears in the sensed current. On the other hand, Figure 5.7b shows the effect of wire resistance only (with zero source and neuronal resistances) with $R_w = 1\Omega$. The positive and negative weights have a wide margin of variation within the same column, and attenuate within the rows due to the sneak path problem. As a result, the wire resistance is much more critical than the source and neuronal resistances. Figure 5.7c shows the combined behavior of the three resistances.

Figure 5.8 shows the normalized root mean square deviation (NRMSD) which measure the deviation of the sensed current from the ideal case. Figure 5.8a shows the strong dependence of NRMSD on wire resistance, irrespective of the array size. This graph partly explains why the previous work [140] concluded that the sneak path effect is insignificant — it is true only if wire resistance is very small. On the other hand, Figure 5.8b shows the effect of changing the neuronal resistance while fixing the wire resistance for 128×128 array. With increasing the wire resistance, the deviation becomes constant which means that the deviation is mostly caused by the wire resistance.

Sneak Path Problem

The measured current of a RRAM cell is ideally constant regardless the RRAM's position inside the array. Nevertheless, due to the existence of wire resistance, IR voltage drops are created in the array. These voltage drops accumulate throughout the array due to the grounded bitlines (columns). The accumulated voltages create leakage currents (sneak currents) through the LRS devices. Thus, the measured current can vary depending on the RRAM cell position. The magnitude of the problem can be significantly reduced by using selector material or access transistors among RRAM cells, or if the wire resistance is nearly zero. In this section, we address the question of whether wire resistance of typical values

will create a sneak path problem that is significant enough to disturb the functionality of RRAM-based computing applications. As shown in Section 5.5, the smaller the r is, the more severe the sneak path problem becomes.

5.4 BNNs Realization on Binary Crossbar Arrays

5.4.1 BNNs on RRAM crossbars

Binary neural networks, where only two states are needed, have emerged as a means to avoid the aforementioned issues plaguing continuous-value neural networks. BNNs have binary neuron activations, either $\{0, 1\}$ or $\{-1, +1\}$, which is very beneficial in terms of hardware (area and power saving) by eliminating the need for analog to digital and digital to analog converters or sophisticated analog circuits [136], in addition to allowing bitwise communication between the layers. Furthermore, in terms of devices, bipolar RRAMs have shown promising performance in terms of endurance, variability and energy [141].

BNNs [142] use only two values $\{-1, +1\}$ to represent synaptic weights and activations (i.e., neuron outputs). A typical memristive BNN stores weights in the RRAM crossbar array (RCA). In order to create negative weights, two weight realization techniques have been proposed: (i) using two RRAM cells per weight [61] as shown in Figure 5.9a, and (ii) using one RRAM as weight in addition to one shared reference RRAM with the conductance of $G_r = (G_{\max} + G_{\min})/2 \approx G_{\max}/2$ for high K as shown in Figure 5.9a [135].

The first technique has double the dynamic range (conductance range $\approx (-G_{\max}, G_{\max})$), making it less susceptible to noise and variability, at the cost of doubling the area and power dissipation. It also exacerbates the sneak path problem due to having approximately double the wire segments. The second technique, despite having a smaller dynamic range (conductance range $\approx (-0.5G_{\max}, 0.5G_{\max})$), requires less area and power dissipation, and

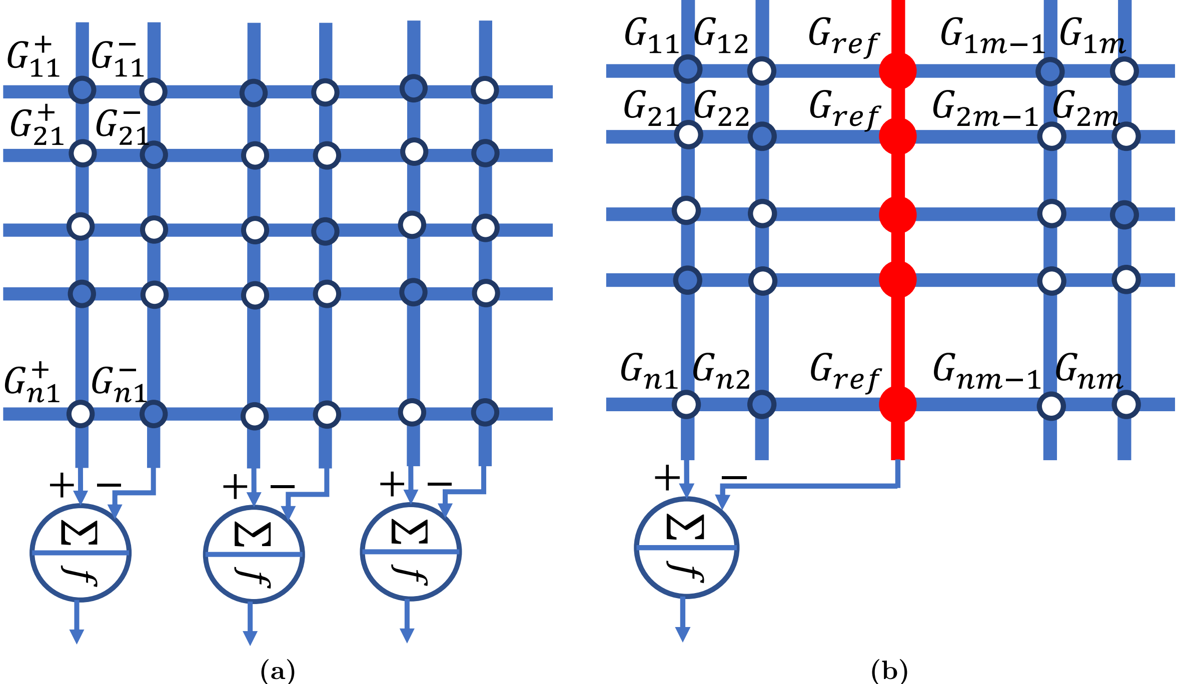


Figure 5.9: Synaptic weight realizations using RRAMS; (a) balanced and (b) unbalanced realizations.

has a less severe sneak path problem.

In this work, we consider the second weight realization technique with reference columns having double the low resistance state, due to its superior area and power dissipation.

5.4.2 Necessity of Large Weight Array Partitioning

Partitioning each layer into the small arrays is necessary for three main reasons;

- Sneak Path problem: partitioning helps to reduce the effect of the problem compromising the main benefit of RRAMs which is the density.
- Driver nonideality: each crossbar is driven by a driver circuit or buffer. The loading of the driver circuit is equivalent to parallel resistors of the driven row creating a voltage divider with the output resistance of the driver circuit. For example, the worst case is when all the devices within the same row have a low resistance state (LRS) and the

output resistance of the driver is R_o. Thus, the voltage delivered to the crossbar input is $V_d = V_{in}\frac{LRS}{LRS+NR_o}$. So, it is necessary to reduce the number of devices per row, N, to mitigate the effect of the driver. Otherwise, it has to be taken into consideration during the training. We do not consider it in this work for two reasons 1) with well designed peripheral circuit, its effect can be eliminated or at least mitigated. and 2) the sneak path problem is the main cause of the performance degradation [134]. However, we study its effect on the performance to find the required output resistance value of the driver circuit for the designers in section V.

- Fabrication problem: It is less complex to fabricate small crossbar arrays with high reliability.

In addition, it is recommended to use two separate crossbar arrays for positive and negative conductances to have symmetric sneak path behavior. The corresponding conductances are scaled by the same value unlike using a single crossbar array for both positive and negative conductances where each conductance will be scaled with different values.

To have accurate inference results, it is needed to run the inference with SPICE simulation where all circuits included. A SPICE simulator is adopted where the weight matrices are partitioned into small crossbar arrays as discussed in [134] and simulated using a transient simulation for different input samples. Figure 5.10 shows the simulation time of matrix-vector multiplication using SPICE for a 256×256 array partitioned into smaller arrays and for different input samples. The SPICE simulation time, without the peripheral circuits such as neuronal sensing circuit and drivers, increases exponentially with increasing the crossbar array size and linearly with increasing the input samples. On the other hand, the same figure shows the numerical SPICE-equivalent simulator adopted from [5]. The numeric simulator runs 140× faster than SPICE for one input sample and 1000× faster than SPICE for 10 successive input samples. It is worth highlighting that the numerical results of the MVM are the same as the SPICE results.

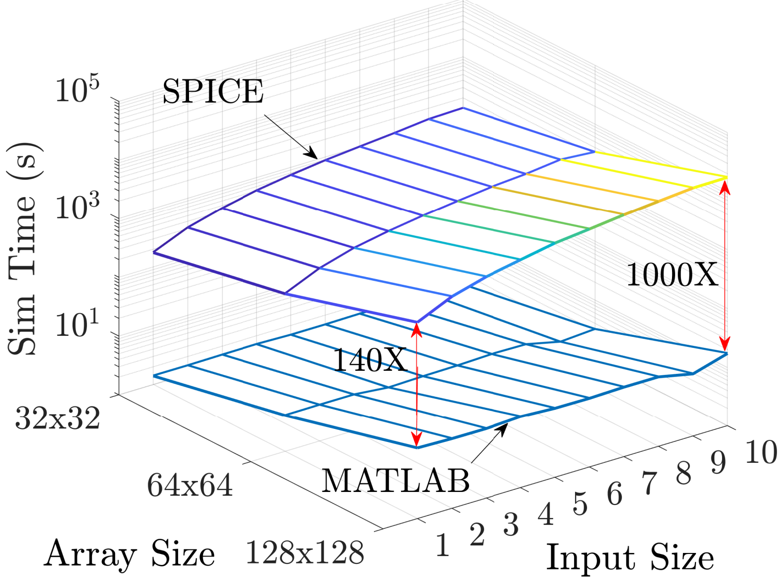


Figure 5.10: Simulation time comparison between SPICE and numerical simulator, adopted from [5], for performing MVM of 256×256 array partitioned into 32×32, 64×64 and 128×128 and for different number of input samples.

Although the numerical model runs orders of magnitude faster than SPICE simulations, it is better not to be included in the DNN framework. it would take reasonable training time for small networks such as MNIST data-set. However, it would take much time to simulate for deeper and larger networks with many convolutional layers. Also, it is better to have solutions that can be used to describe fabricated hardware. In this work, we use the numerical or SPICE simulator without loss of generality as a reference due to the lack of the hardware.

The large layer matrices should be partitioned into small matrices that can be implemented using realizable crossbar arrays. Figure 5.11 shows the partitioned crossbar arrays and the interconnect fabric between them to realize the complete VMMs. In order to have the same structure of a large crossbar array, vertical and horizontal interconnects are placed under the crossbar arrays. This horizontal interconnect is used to connect the inputs between the crossbar arrays within the same array rows. The vertical interconnect is used to connect the outputs of the vertical crossbar arrays. The vertical interconnects are grounded through the

sensing circuit to absorb the currents within the same vertical wire. The sensed currents are compared with the reference current to generate the outputs.

In this work, we ignore the effect of the interconnect fabric parasitics, since we assume that drivers are added to the inputs of each crossbar array to avoid loading effect. These drivers are not shown in Figure 5.11 for clarity. In the case of BNNs, CMOS inverters can be used as drivers since the input signals are binary (i.e., $\{0, 1\}$ or $\{-1, 1\}$). The drivers of each crossbar can be placed under the interconnect fabric. The wire resistance of the interconnect fabric and the input capacitance of the drivers would cause some delay, which can be calculated using the Elmore delay model. The wire resistance of the interconnect per array is mR_w where m the number of columns and R_w is the wire resistance per cell. The Elmore delay of such an interconnect wire is $0.67mR_wC_d$, where C_d is the input capacitance of the driver. Thus the total input delay is $0.67(M-m)R_wC_d + (M/m)\tau_d$, where M/m is the number of horizontal crossbar arrays and τ_d is the driver delay. On the other hand, due to the current sensing of the outputs of the crossbar, the crossbar delay can be neglected compared to the neuronal circuit. A more detailed analysis of this and the routing issue will be pursued in future work.

5.4.3 Evaluation Results

Figure 5.12 shows two cases of SPICE simulation for the normalized, sensed current of each weight in a 128×128 passive crossbar array. The sensed current of each cell is adjusted by subtracting it from that of the corresponding reference cell located at the center of the row. Clearly, the wire resistance highly affects the sensed current where the sensed current decays exponentially in both directions of the crossbar array (from G_{00} cell toward G_{NN} cell in Figure 5.9).

These results suggest that simple models that ignore the wire resistance of crossbar arrays

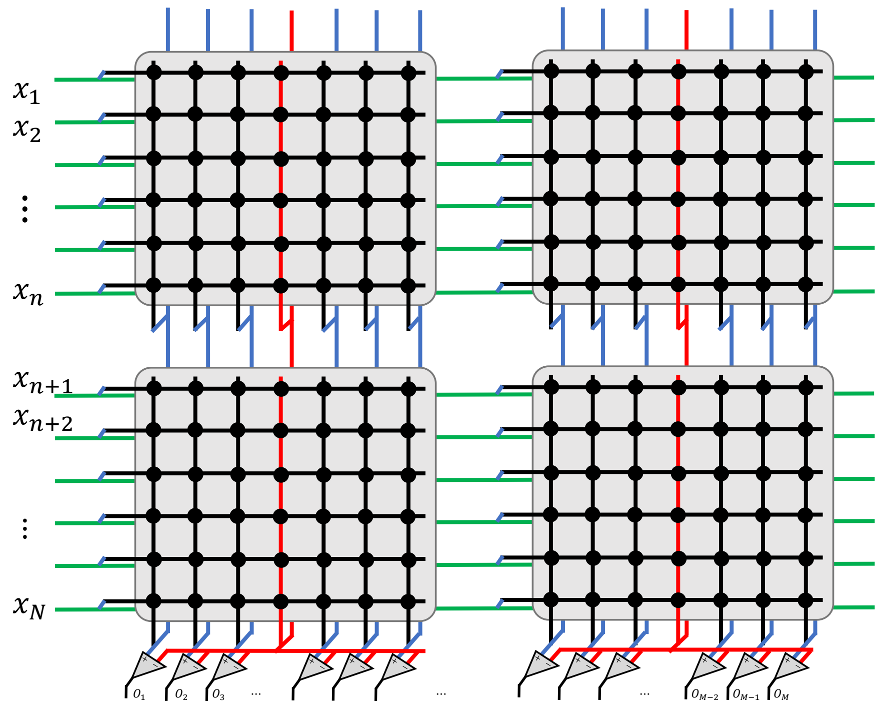


Figure 5.11: Realization of the partitioned matrices.

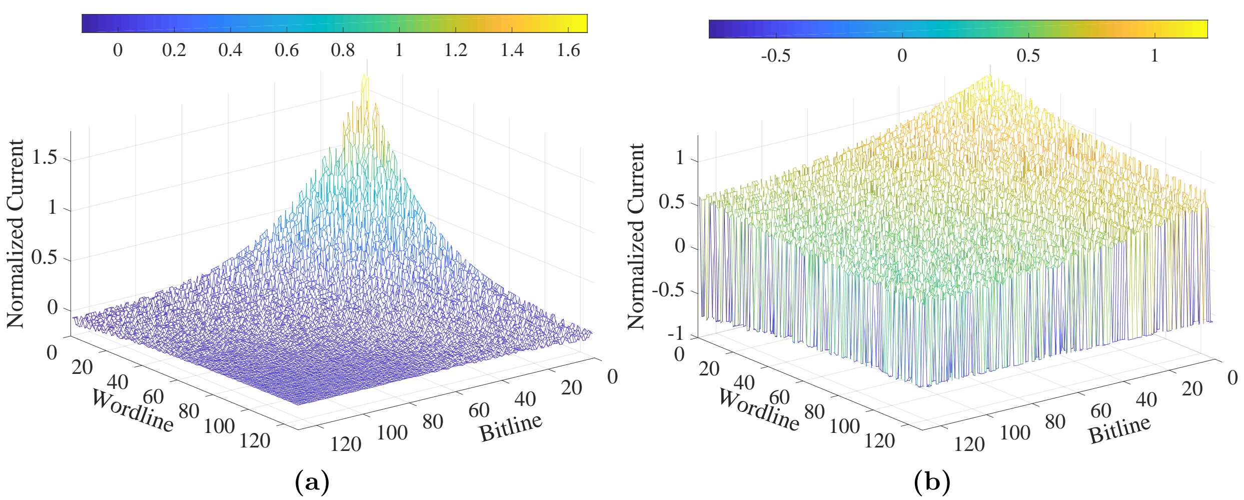


Figure 5.12: Normalized sensed current of two crossbar arrays with a) $r = 500$ and b) $r = 10^4$.

would give a false prediction on the accuracy of the applications. It also means that BNN weights optimized without the knowledge of wire resistance could be far inferior than what may be achievable by incorporating wire resistance knowledge, as shown in the next section.

5.4.4 DNN Framework

To evaluate the accuracy of our RRAM crossbar-based DNN hardware, we use the BinaryNet framework [142] extended with our RRAM crossbar array simulation module. Our RRAM crossbar array simulation module supports both SPICE simulation and an equivalent numerical method, which is an implementation of a generic resistive network model discussed in chapter III. For steady-state behavior both simulation methods yield the same result, but the latter can be faster for typical size arrays (e.g., 128 $\times$ 128). This combination of the BinaryNet framework and the RRAM crossbar simulation allows for accurate examination of the effect of non-ideal behavior of RRAM crossbars on the accuracy of DNN inference. While the BinaryNet framework supports both training and inference, we use RRAM crossbar simulation for inference only due to its high runtime overhead. It is worth mentioning that BinaryNet retains real-valued weights for training, and binarizes them only for inference, which is the common practice in BNN training; otherwise, training in the binary domain would not converge.

Algorithm 5.2 is the basic BinaryNet algorithm [142] with modification to include the mask technique in the forward and backward propagations. In the feedforward propagation section, the modifications are as follows: (1) partitioning the binarized weights of each layer, W_k^b, into small weight matrices P_k^b, (2) application of the mask to each partitioned array to create W_{sp} to be used in (3) the forward propagation and (4) backward propagation instead of the binarized weights W^b. Similar modifications can be added to any other Binary DNN framework to capture sneakpath problem.

Algorithm 5.1 Modified BinaryNet algorithm

Require: a minibatch of inputs and targets (a_0, a), previous weights W, previous BatchNorm parameters θ, weights initialization coefficients from γ, and previous learning rate η

Ensure: updated weights W^{t+1}, updated BatchNorm parameters θ^{t+1} and updated learning rate η^{t+1}.

{1. Computing the parameters gradients:}
{1.1.Forward propagation:}
for $k = 1$ to L **do**
 $W_k^b \leftarrow \text{Binarize}(W_k)$
 $P_k^b \leftarrow \text{Partition}(W_k^b)$ ▷ Modification-1
 $W_{sp_k} \leftarrow P_k^b \odot M$ ▷ Modification-2
 $s_k \leftarrow a_{k-1}^b W_{sp_k}$ ▷ Modification-3
 $a_k \leftarrow \text{BatchNorm}\ (s_k, \theta_k)$
 if $k < L$ **then**
  $a_k^b \leftarrow \text{Binarize}(a_k)$
 end if
end for
{1.2. Backward propagation:}
{Compute $g_{aL} = \frac{\partial C}{\partial a_L}$.}
for $k = L$ to 1 **do**
 if $k < L$ **then**
  $g_{a_k} \leftarrow g_{a_k}^b \circ 1_{|ak| \leq 1}$
 end if
 $(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}\ (g_{a_k}, s_k, \theta_k)$
 $g_{a_{k-1}^b} \leftarrow g_{s_k} W_{sp_k}$ ▷ Modification-4
 $g_{W_k^b} \leftarrow g_{s_k}^T a_{k-1}^b$
end for
{2. Accumulating the parameters gradients:}
for $k = 1$ to L **do**
 $\theta_k^{t+1} \leftarrow \text{Update}(\theta_k, \eta, g_{\theta_k})$
 $W_k^{t+1} \leftarrow \text{Clip}(\text{Update}(W_k, \gamma_k \eta, g_{W_k^b}), -1, 1)$
 $\eta^{t+1} \leftarrow \lambda\eta$
end for

Table 5.2: Understudy Binary neural networks.

	MNIST	CIFAR10	SVHN
Network type	MLP	CNN	CNN
#Layers	4	9	9
#Crossbar Arrays	640	864	402
Initial training #epochs	100	500	200
Retraining #epochs	50	200	80
Baseline test accuracy on GPU	98.41%	88.62%	97.18%

Table 5.3: MLP neural network validation accuracy using 128 × 128 crossbar arrays configuration.

Ref Cols \ r	10^4	10^3	500
1	78.4%	17.9%	18.2%
5	77.1%	10.7%	10.8%
9	75.9%	9.6%	10.5%

In this work, we run the experiments on one multilayer perceptron (MLP) network and two convolutional neural networks (CNNs) that are supported by BinaryNet framework [142]. Table 5.2 summarized the networks' structures. For instance, the binary neural network for MNIST dataset classification consists of 4 layers; the layer connections are as follows: $784 \Rightarrow 2048 \Rightarrow 2048 \Rightarrow 2048 \Rightarrow 10$. Each layer is partitioned into an integer number of 128×128 arrays. The missing elements in the remaining arrays are filled with random data. Thus, the overall network requires 640 of 128 × 128 crossbar arrays.

Table 5.3 and 5.4 list the DNN inference accuracy for test images when using the weights trained for the original binary neural networks. While the BNNs achieve very high accuracy that is nearly indistinguishable from that of floating-point version networks, our evaluation shows that RRAM-based BNNs show unacceptable accuracy. This is mainly caused by the sneak path problem, causing a discrepancy between the ideal crossbar behavior and RRAM-based crossbar behavior. In the next section, we present hardware and software techniques to minimize this discrepancy.

Table 5.4: Convolutional neural network validation accuracy using 128×128 crossbar arrays configuration.

	CIFAR10		SVHN	
Ref Cols \ r	10^4	10^3	10^4	10^3
1	10.00%	10%	19.16%	19.59%
9	10.69%	10%	11.36%	9.69%
15	9.76%	10%	9.68%	9.69%

5.5 Mask Technique: *Making Training Possible*

The results in Table 5.3 and 5.4 are calculated without any (re)training for RRAM crossbar arrays. DNN retraining means performing additional iterations of weight updates on top of already trained weights as opposed to DNN training, which refers to the initial training starting from random weights. Any kind of training incorporating the knowledge of RRAM crossbar behavior will likely improve the accuracy of RRAM-based DNNs significantly. However, training requires many updates to the BNN weight parameters, each of which changes the RRAM crossbar arrays' behavior, thereby requiring very expensive simulation, whether by SPICE or other numerical methods.

5.5.1 Mask Method

Our solution is to use the resulting normalized synaptic current values as a *mask*, to be multiplied to the original binary weight matrix as follows. The mask enables us to predict more realistic behavior of a crossbar array.

$$\mathbf{W}_{sp} = \mathbf{W}_b \odot \mathbf{M} \tag{5.8}$$

where $\mathbf{W}_{sp}$ is the wire-resistance-effect-compensated weight matrix, $\mathbf{W}_b$ the original weight matrix, $\mathbf{M}$ a mask matrix, and $\odot$ element-wise multiplication.

The details of how to generate and use the mask may vary (see the next subsection), but the core idea of using a very light weight modification rule manifested as a *mask* above enables very quick retraining of DNNs for RRAM crossbar arrays and can be integrated into any training framework. During training, our mask is applied to the forward evaluation and error back-propagation. The weight update is done on the original weight matrix, which is kept as real-valued (see Appendix A). Other details about the training are exactly the same as in the original BNN training, including how and when weights are binarized. This training is performed offline, meaning that we obtain optimized BNN weights from the BinaryNet framework, which need to be programmed only once to RRAM crossbars. The programmed weights remain binary during the inference time, as inference does not change the weights. Applying masks during training does not increase training time significantly.

Since masks are only an approximation to the real behavior of RRAM crossbar arrays, we also validate the results using an accurate numerical simulation, as reported in the experimental results Section 5.6.

5.5.2 Mask Generation

The mask matrix can be generated from either SPICE simulations or equivalent numerical methods, and is normalized to the ideal current (ie., sensed current without sneak path problem), which is $V/(2 \cdot LRS)$. Choosing the suitable mask is very critical, since it should reflect the behavior of the sneak path for any given data. In this work, we investigate different mask generation methods. A mask is generated by simulating a crossbar array 100 times with different (random) data patterns to get a smooth mask.

In this work, we consider three types of masks.

1) *Best/worst case mask*: This is generated using a crossbar array with all switching devices set to LRS (worst) or HRS (best). This mask may not be very accurate and does not describe

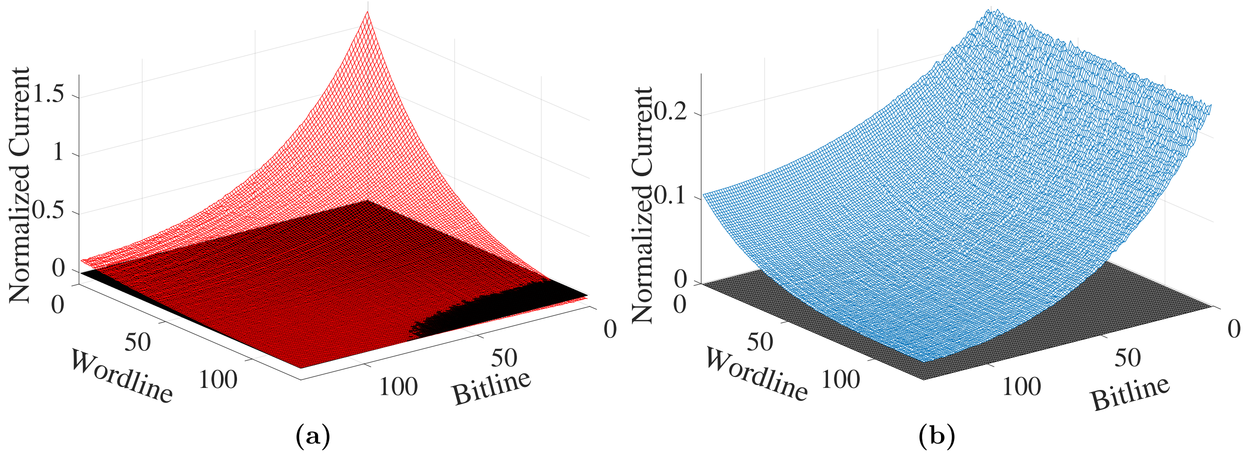


Figure 5.13: Positive and negative masks of the 128x128 crossbar array

the typical sneak path. This mask is used to retrain the network giving good results [64]. However, the validation results are not good since the mask does not accurately describe the real synaptic weights.

2) *Asymmetric mask set*: This mask is developed to capture the effect of the sneak path problem for both positive and negative weights. Thus, two average masks are generated; Positive Mask and Negative Masks to capture the behavior of positive and negative weights, respectively. The positive and negative masks are generated by extracting the positive ones and negative ones of the array then averaged over each one separately. Application of the mask set during DNN training is done as follows.

$$\mathbf{W}_{sp} = \frac{1}{2}\left((\mathbf{W}_b + 1) \odot \mathbf{M}_P\right) + \frac{1}{2}\left((\mathbf{W}_b - 1) \odot \mathbf{M}_N\right) \tag{5.9}$$

where $\mathbf{M}_P$ and $\mathbf{M}_N$ are the positive and negative mask matrices, respectively.

3) *Average case mask*: This type of mask can be generated by either one of two ways: (a) averaging the positive and negative masks extracted using the asymmetric mask set which is referred to as *average mask*, and (b) averaging the absolute value of the normalized measured currents directly which is referred to as *random mask*. We found that their behavior is similar. Application of the mask set during DNN training is done using (5.8).

We have tested the proposed mask sets and we found that the average mask captures the sneak path profile and helps the network to recover despite its simplicity.

5.5.3 Architecture Solutions

Logical Subarray Method

One way to reduce the effect of the sneak path problem is by using only a subarray of a larger physical array. The used part of the array is called *logical subarray* where the rest is set to HRS. Figure 5.13 shows 3D plots for both positive and negative masks. Clearly, they are not identical and the positive mask spans a wider range of values than the negative mask but both capture the sneak path effect. Moreover, the positive mask suffers a sign flip problem where a part of the array has negative values which is expected according to Figure 5.7. Figure 5.13a shows that a sign flip region exists in the last 18 rows which is around 8.4% of the array. This means any data written there would give negative values. The logical subarray method would help to remove the sign-flipped region found in the positive weights (Figure 5.7). It was expected that this problem can be solved by training, possibly by a redistributing of weights. However, since we are using a static mask, the sign flip region always exists. In order to solve this issue, there are two possible solutions: (1) Truncate the last rows that have sign flip problem, thus, the physical crossbar size would be 110×128 which is the same as logical array size. (2) Fix the sign-flipped rows to HRS, which means negative ones and use only the upper part of the array to store the weights. In other words, we would have a 128×128 physical crossbar array and 110×128 logical array size. These rows are biased to zero voltages to do not disturb the operation of the neural network.

Figure 5.14 shows both positive and negative masks with fixing the last 18 rows to HRS to eliminate the sign flip region. Figure 5.15 shows a comparison plot for the average mask of 110×128 logical array between 110×128 and 128×128 physical arrays where the last 18

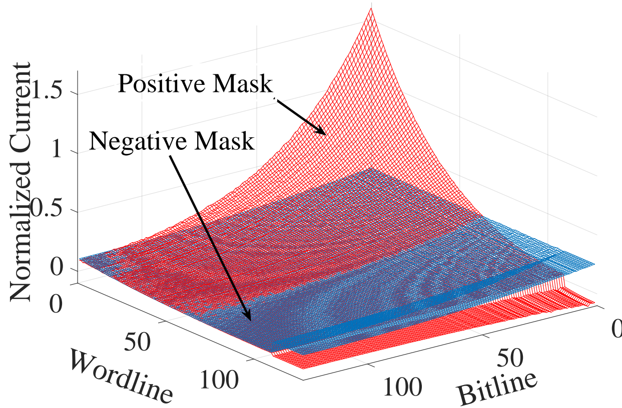


Figure 5.14: Positive and negative masks for 110×128 logical array using 128×128 physical array.

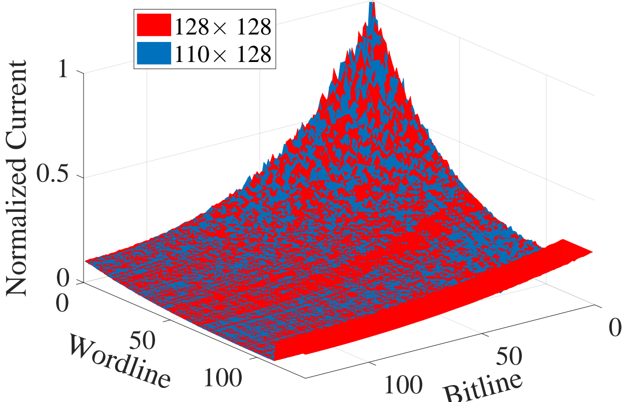


Figure 5.15: Comparison between 128×128 physical array size and 110×128 physical array size for 110×128 logical subarray mask

rows are fixed to HRS. Clearly, the masks have very close behavior which means the masks can be used interchangeably. Figure 5.16 shows the average masks for different r. Clearly, the smaller the wire resistance, the closer the mask is to the ideal case.

Multiple Reference Columns

An alternative method to solve the sign flip problem is to increase the number of reference columns. Adding more reference columns makes the weight realization close to the first weight realization discussed in section 5.4.1. However, adding too many reference columns would make the sneak path problem worse and also increase the area and power consumption. Figure 5.17 shows the positive mask for 1 and 9 reference columns. Increasing the number of

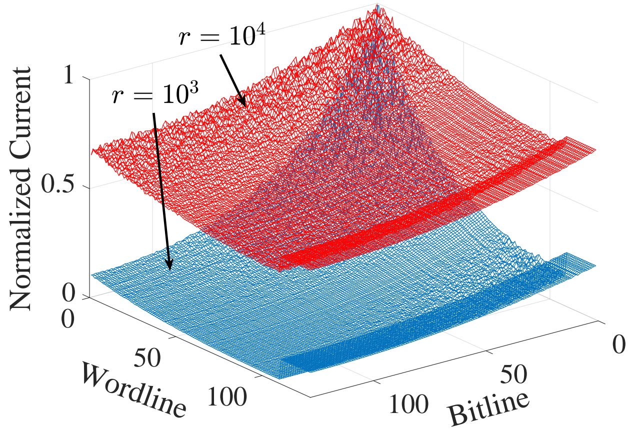


Figure 5.16: Effect of changing Wire resistance on the sneak path problem.

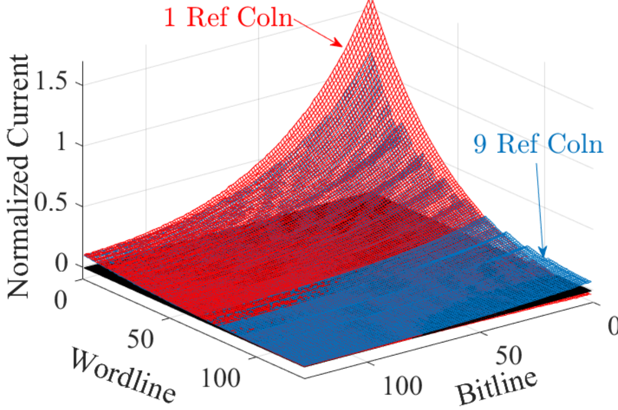


Figure 5.17: Effect of changing number of reference columns on the mask.

reference columns can decrease the range of variation in positive weights, which might help improve the sneak path problem. For instance, with 1 reference column, the mask suffers the sign flip problem (red curve below the black plane) where it spans from $\{-0.03 \sim 1.65\}$. However, with 9 reference columns, there is no sign flip problem and the curve has less variation range where it spans from $\{0.01 \sim 1.15\}$ which would improve the vector-matrix multiplication accuracy. Figure 5.18 shows that increasing the number of reference columns helps to have symmetric spread in the positive and negative weight values. This method is very simple, and it is found that it can effectively address the problem as will be shown in the next section.

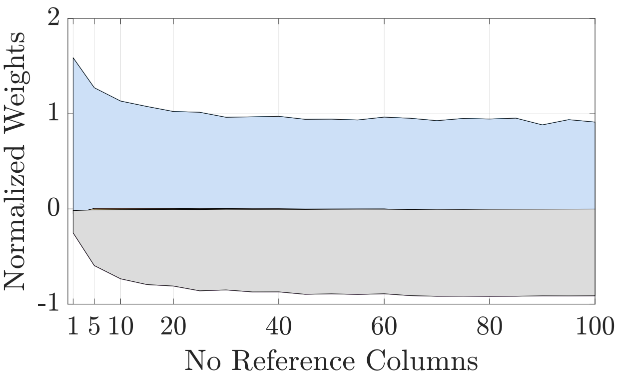


Figure 5.18: Normalized measured weight with changing the number of reference columns.

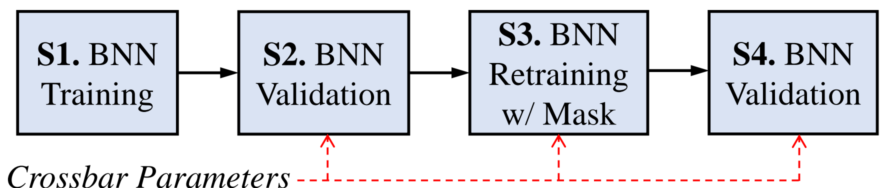
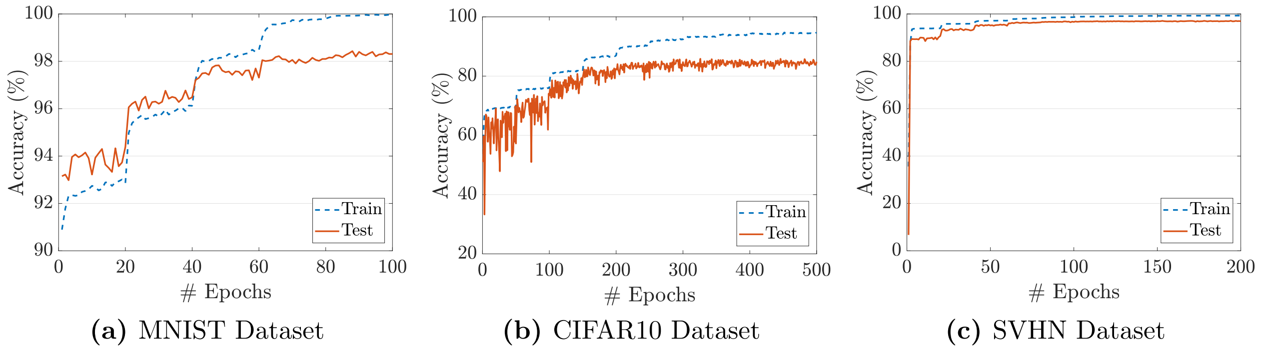


Figure 5.19: DNN experiment flow (Sn means Step n).

5.6 BNN Results and Discussion

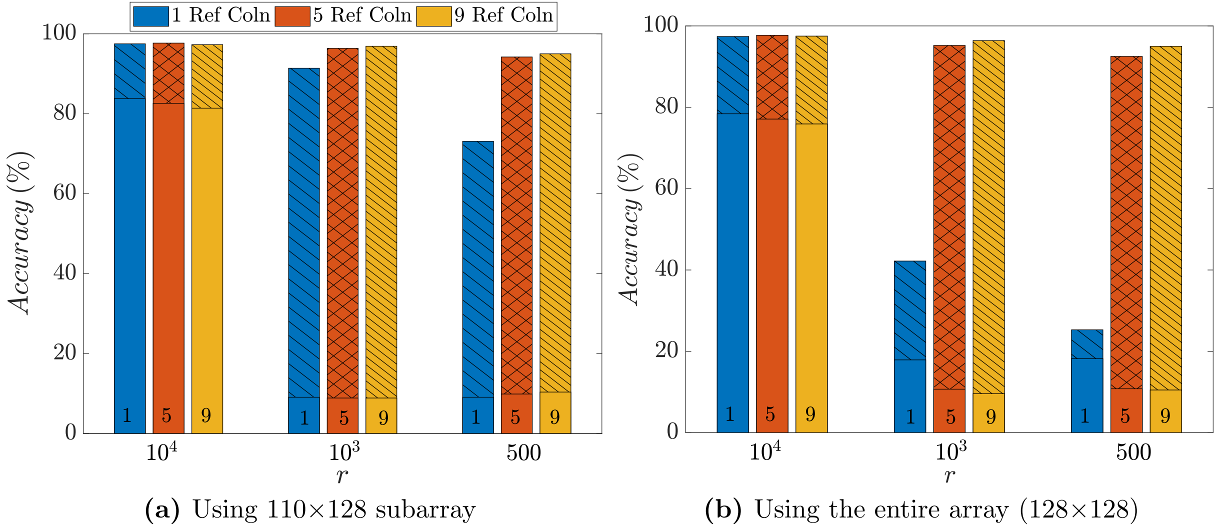
Figure 5.19 illustrates the flow of the DNN experiment. In *Step 1*, we use an existing BNN framework [142] to find trained weights for a BNN. In *Step 2*, the trained binary weights are fed to the BNN validation tool, which is the inference part of the same BNN training framework extended with SPICE-equivalent simulation of RRAM crossbar arrays. The result of the validation step is the classification accuracy that is expected of real hardware. In *Step 3*, we retrain the network. The most important difference of our retraining from *Step 1* training is that the use of mask(s) during retraining allows training the network against the more realistic real-valued reading of RRAM crossbar arrays vs. idealized binary weights. Finally in *Step 4*, the binary weights are again fed to the validation tool, which is the same as that of *Step 2*. All DNN classification accuracy results are for test images, which are not seen during training.

Both proposed mask techniques were tested, where each network is constructed using 640 and 757 crossbar arrays for 128×128 and 110×128 logical array sizes, respectively.



(a) MNIST Dataset **(b)** CIFAR10 Dataset **(c)** SVHN Dataset

Figure 5.20: DNN train and test accuracy during retraining the network.



(a) Using 110×128 subarray **(b)** Using the entire array (128×128)

Figure 5.21: DNN validation accuracy before retraining (un-hashed bars) after retraining (hashed bars).

Table 5.5: MLP network validation accuracy before retraining

Logical Array Size	Ref Cols \ r	10^4	10^3	500
110×128	1	83.8%	9.1%	9.1%
	5	82.6%	8.9%	9.9%
	9	81.4%	8.9%	10.3%
128×128	1	78.4%	17.9%	18.2%
	5	77.1%	10.7%	10.8%
	9	75.9%	9.6%	10.5%

5.6.1 Impact of Retraining on Binarized MLP Network

Table 5.5 shows the classification accuracy for the MNIST dataset before retraining, i.e., the accuracy from *Step 2* of Figure 5.19. Not surprisingly, the classification accuracy is very low, which is due to the crossbar non-idealities distorting the result of VMM computation significantly. It is worth mentioning that the validation accuracy decreases slightly with increasing the number of reference columns because of the fluctuations that exist in the structures with more than one reference column.

Figure 5.20a shows the progress of train/test accuracy during retraining MNIST network for $r = 10^4$ with 9 reference columns using 128×128 array. Figure 5.21 shows the validation accuracy before and after retraining versus the ratio r expressing the wire sneak path effect. Retraining is done for 50 epochs. The validation accuracy after retraining is from *Step 4* of Figure 5.19, as opposed to the accuracy from *Step 3*, which is during training. During training, classification accuracy converges rather quickly to a very high level, eventually reaching 97∼98% in all cases. For each of the cases, a different mask is generated as per the method explained in Section 5.5.2. Thus, different trained weights are used for different cases.

Interestingly, the drastic accuracy drop caused by wire resistance can be mostly recovered by retraining the network with one of the proposed masks. The post-retraining accuracy is upwards of 95% even in the worst case (with 9 reference columns), which reinforces the need for considering crossbar non-idealities in DNN inference as well as training. We also observe that increasing the number of columns improves the accuracy as well; in some cases, using 9 reference columns improves accuracy by an extra $1 \sim 3\%$ point over using 5 reference columns.

Per the accuracy results shown in Figure 5.21, the 110×128 logical mask shows much better results compared to the 128×128 logical mask, where the entire array is used, for

Table 5.6: MNIST validation Accuracy for different hardware configurations

r	10^4		10^3	
# Ref Columns per 128 columns	5	9	5	9
128×128 Array	96.8%	97.23%	95.34%	95.84%
256×256 Array	93.88%	94%	84.48%	88.17%

one reference column case. This result is expected since using 1 reference column has the sign flip problem which requires increasing the number of reference columns to eliminate it. By increasing the reference columns, the accuracy results of both mask techniques are very comparable with less than one point difference in favor of 110×128 logical mask. Thus, 128×128 logical mask with multiple reference columns would be preferred for less area and power.

It is worth noting that the retrained weights are sensitive to the wire resistance value where the usage of the trained weights with higher wire resistance does not work for low wire resistance case. For instance, the retrained weights with $r = 10^3$ are used to validate the $r = 10^4$ case with 1 reference column. We found that the accuracy drops to 79% from 91% for 110×128 logical mask. This means that retraining is always necessary.

Choosing the suitable hardware configuration is critical for a nonideality such as wire resistance. Our choice is to maximize the crossbar array size without scarifying the accuracy. Larger crossbar arrays reduce the routing fabric, but also degrade the performance. Thus, we have conducted an experiment using 256×256 crossbar array for MNIST classification. Table 5.6 shows the validation accuracy for both configurations. For a fair comparison, we doubled the number of reference columns for 256×256 array. Despite the effectiveness of the mask technique to achieve reasonable accuracy values, 128×128 array configuration approaches the baseline accuracy and outperforms 256×256 configuration.

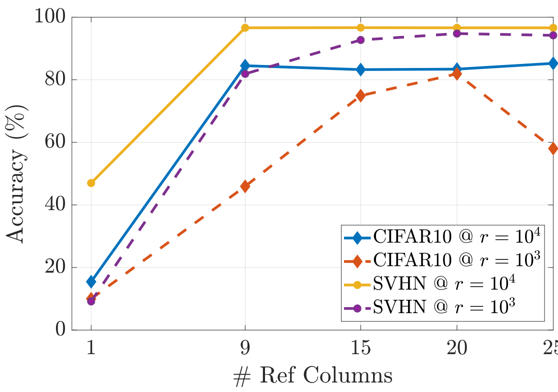


Figure 5.22: Conventional neural networks validation accuracy for different reference columns.

5.6.2 Impact of Retraining on Binarized CNNs

According to the previous discussion, the same or better results can be obtained by using 128×128 logical array instead of using 110×128 logical array, by adding more reference columns to alleviate the sign flip problem. Thus, in CNNs, we report the results using 128×128 logical array only. The validation accuracy without retraining the CNNs (*step 2*) is around 10% for both CIFAR10 and SVHN datasets as shown in Table 5.4.

Figure 5.20b and 5.20c show the progress of train/test accuracy during retraining of CIFAR10 and SVHN networks, respectively for $r = 10^4$ and with 9 reference columns using 128×128 array. Per these results, the train accuracy saturates after 200 and 80 epochs for CIFAR10 and SVHN networks, respectively. Thus, we stop retrain when this number of epochs is reached. Also, Figure 5.22 shows the training validation accuracies for CIFAR10 and SVHN datasets for different scenarios. Due to the high sensitivity of these datasets, we increased the number of reference columns to improve the validation accuracy. Clearly, validation accuracy keeps improving with increasing the number of reference columns approaching training test accuracy.

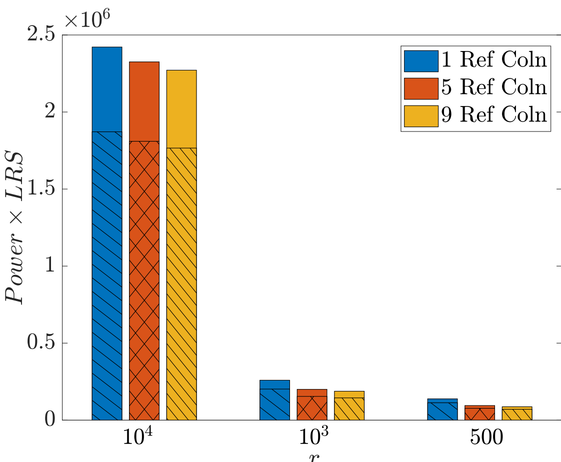
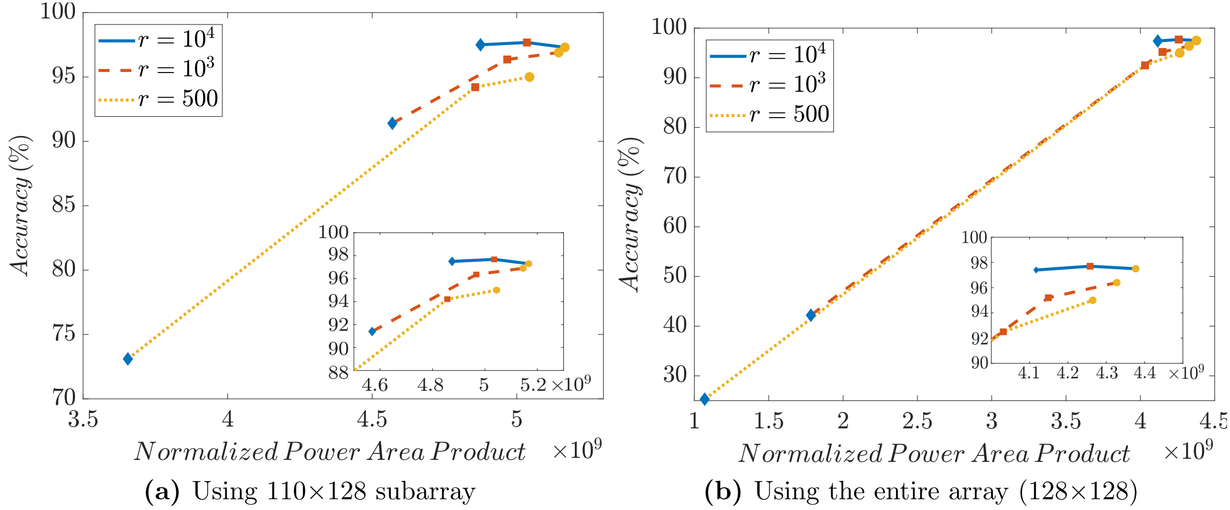


Figure 5.23: Power dissipation comparison between proposed masks, 110 × 128 logical mask (un-hashed bars) and 128 × 128 mask (hashed bars) in the crossbar arrays used for the MNIST network.



(a) Using 110×128 subarray

(b) Using the entire array (128×128)

Figure 5.24: DNN validation accuracy after retraining and validation of MNIST network for different r values and for one (diamond), five (square) and nine (circle) reference columns
.

5.6.3 Power Consumption in Resistive Crossbar Arrays

Power is inversely proportional to resistance. Thus, it is essential to have high resistance devices to reduce total power consumption. Figure 5.23 shows the normalized power consumption to LRS in resistive crossbar arrays, to consider different devices, for both masks and for different r. It is important to mention that these power results are calculated for crossbar arrays only to have a fair comparison with the proposed mask. Also, both configurations have the same number of drivers and sensing circuits consuming same amount of power. It is expected that the power consumption using 110×128 logical array is higher than the other case. Interestingly, increasing the wire resistance reduces power consumption, due to the increased resistivity of the network. Also, for the same reason, increasing the number of reference columns decreases the power consumption as well.

5.6.4 Discussion and Comparison

Due to the simplicity of the binary neural networks, simple drivers and sensing circuits can be used, which can be placed beneath the crossbar arrays. The area can be estimated based on the number of crossbar arrays and size of each cell. Thus, the 110×128 logical method would occupy 18.2% extra area compared to the 128×128 logical method.In addition, increasing the number of columns increases the total area by less than 1%.

Figure 5.24 shows the validation accuracy versus the normalized power area product, PAP, for different wire resistance ratios and different crossbar arrays. Each mask has comparable results in terms of PAP and accuracy for 5 and 9 reference columns. The 110×128 mask method has 18% higher PAP than 128×128 mask method with a higher negligible difference in the validation accuracy. In conclusion, the 128×128 mask method is recommended with nine reference columns.

One of the important issues is the device to device variability in both low and high resistance states. In addition, the reading operation might disturb the resistance as well. Thus, the device variability should be taken into consideration while the training operation or the retrained weights should be validated in the presence of the device variability. In this work, we validated the results with 10% and 20% variations in both high and low resistance states. We got the same performance results, obtained with the absence of the variations, with a negligible difference which means that the retrained network is immune against device variations. This conclusion matches the results in [131] since this is one of the advantages of BNNs, being less sensitive to weight variations.

The higher r (the low wire resistance or the higher LRS) is, the less severe sneak path problem is and the better validation accuracy is, without the retraining. For instance, for $r = 10^4$, the accuracy without the retraining is $1 - 2\%$ point less than the baseline accuracy. After the retraining a slight improvement is achieved of $0.5 - 1\%$ point.

One way to avoid the sneak path problem in crossbar structures is by using 1T1R-based structures where the transistor works as a selector to activated one column at a time while other columns are not selected. As a result, there are no other paths from the inputs to the outputs which means no sneak path problem. This method avoids the sneak path problem however it causes a high latency where each VMM operation is performed in at least N cycles where N is the number of neurons per layer. Also, adding a transistor increases the overall area by a constant factor which is approximately $3(1 + W/L)/4$ where W/L is the aspect ratio of the transistor (i.e., 2). In addition, in 1T1R structure, only one shared neural activity circuit can be used per layer. But, it requires adding some memory cells (register) to store the VMM outputs, which means N bit register is needed with dual supply $\pm V_{dd}$, since the neural activity is ± 1.

A current sensing circuit is designed based on the circuit introduced in [138] using TSMC 65nm and a level shifter with some buffers are used to have ± 1 voltage signal to be delivered

Table 5.7: Comparison between 1T1R and 0T1R architectures.

	Area (μm^2)		Latency (ns)		Energy (J)	
	1T1R	0T1R	1T1R	0T1R	1T1R	0T1R
MNIST	2.54E+03	2.41E+03	9.23E+03	3.00E+00	1.57E-02	2.81E-06
CIFAR10	1.47E+04	1.36E+04	6.91E+05	4.04E+03	1.54E-01	2.05E-04
SVHN	1.06E+04	1.03E+04	5.44E+05	4.04E+03	2.54E-02	5.93E-05

to the next layer. In addition, registers have been used to store the outputs for 1T1R structure. The register size for each layer equals the number of neurons existing in this layer, e.g. for MNIST case, the used register sizes are 2048, 2048, 2048 and 10 for the first, second, third and fourth layer, respectively.

Table 5.7 shows a comparison of the area, latency and energy while processing one input image. The reported area is calculated for crossbar arrays, drivers, sensing circuits and registers. The 0T1R-based hardware configuration has slightly less area compared to 1T1R-based hardware configuration. It is worth mentioning that in the 0T1R-based configuration, the CMOS circuits area (e.g. sensing circuits, and registers) are approximately equal to crossbar arrays' area which means 50% area saving if CMOS circuits are placed under the crossbar arrays, unlike the the other implementation where the active area already occupied by the transistor associated with each resistive device. On the other hand, the 0T1R is 3000×, 170× and 134.7× faster compared to 1T1R-based implementation for MNIST, CIFAR10 and SVHN datasets, respectively. The proposed technique results in a slight drop in the accuracy while achieving highest parallelism. Also, 0T1R-based hardware configuration has around 5590×, 750×, and 428× energy saving for MNIST, CIFAR10 and SVHN networks, respectively due to less latency and power.

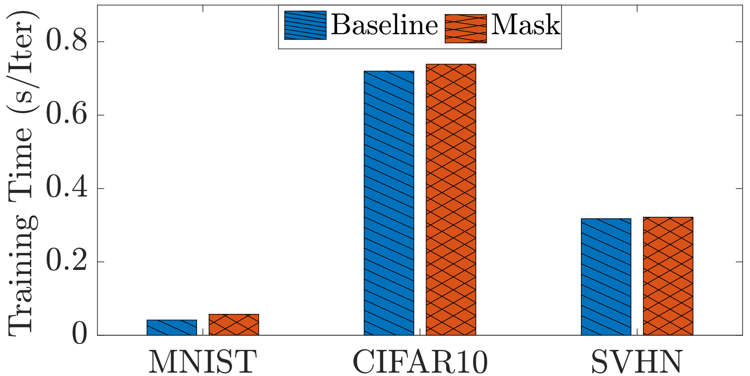


Figure 5.25: Training time comparison (per iteration).

5.7 Extending Mask technique for Quantized Neural Networks

In this section, we discuss the neuromorphic hardware based on RRAM-crossbar arrays (RCAs) where the MVM computations are carried out.

5.7.1 Effect of Wire Resistance Problem on QNNs

The wire resistance is inevitable in nanostructure crossbar arrays. It is expected that the wire resistance would reach around 90Ω for $5nm$ feature size [5]. The wire resistance creates IR voltage drops, which create multiple paths, called sneak paths, from the input ports to output ports because the columns are not grounded anymore, resulting in a highly distorted MVM result.

Steady-State model of MVM using Crossbar Array

Figure 5.27 show the crossbar array with wire and capacitive parasitics. According to [5], the nodal voltages can be obtained by solving the following 1st order system of differential

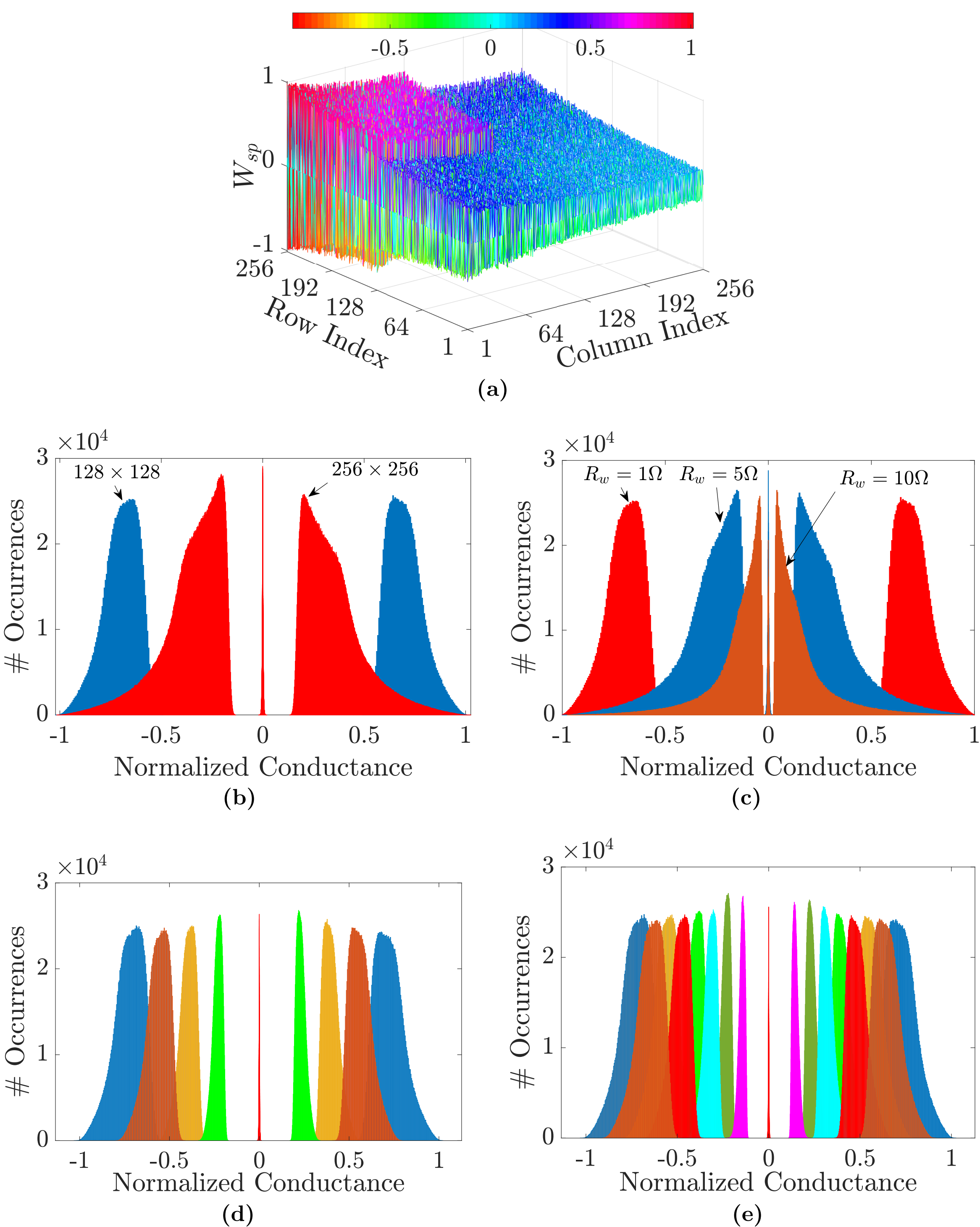


Figure 5.26: (a) Measured weight per cell for 128 × 128 and 256 × 256 crossbar arrays at $R_w = 1\Omega$, (b) Histogram of the measured conductance normalized to G_{max} of the crossbar arrays shown in (a), (c) (5) Histogram of the measured conductance for 1-bit case for different wire resistance, (d) and (e) Histograms of the normalized conductance for 3-bit and 4-bit cases at $R_w = 1\Omega$.

equations:

$$\mathbf{MV} + \mathbf{N}\dot{\mathbf{V}} = \mathbf{Gu} \tag{5.10}$$

where $\mathbf{V}$ and $\mathbf{u}$ are the nodal voltage and the excitation vectors, respectively and $\mathbf{M}$, $\mathbf{N}$ and $\mathbf{G}$ are the coefficient matrices containing the RRAM's conductances, wire and capacitive parasitics values. The construction of these matrices can be found in detail in [5]. Since our concern is the steady-state behaviour, the capacitive parasitics can be ignored. Thus, the steady-state nodal voltage vector can be written as

$$\mathbf{V}_{ss} = \mathbf{M}^{-1}\mathbf{Gu} \tag{5.11}$$

The output current is needed to have accurate MVM as discussed. Thus, the steady-state

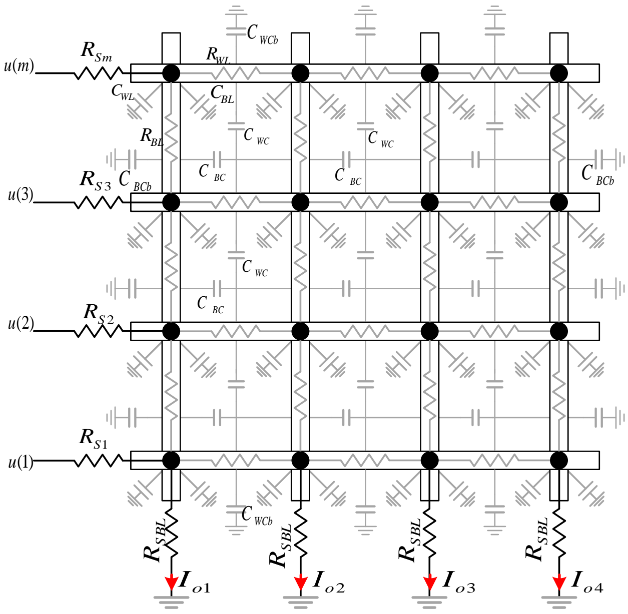


Figure 5.27: Circuit model of the crossbar array.

output current equation can be defined as

$$\mathbf{I}_{ss} = \mathbf{\Psi}\mathbf{V}_{ss} \tag{5.12}$$

where $\mathbf{I_o}$ is the output current vector and $\mathbf{\Psi}$ is the selection matrix which is given as

$$\mathbf{\Psi} = \frac{1}{R_{sBL}}\left[\mathbf{0}_{n\times mn}\ \mathbf{I}_{n\times n}\ \mathbf{0}_{n\times(m-1)n}\right]. \tag{5.13}$$

where R_{sBL} is the parasitic load resistance of the crossbar array and m and n are the array dimensions. Consequently, the output current can be written as

$$\mathbf{I}_{ss} = \mathbf{\Psi}\mathbf{M}^{-1}\mathbf{G}\mathbf{u} \tag{5.14}$$

This equation can be written as $\mathbf{I}_{ss} = \mathbf{W_{sp}}\mathbf{u}$ where $W_{sp} = \mathbf{\Psi}\mathbf{M}^{-1}\mathbf{G}$ Similar analysis can be adapted for nonlinear switching devices.

Thus, the output current with wire resistance effect can be modeled as

$$\mathbf{I} = g(\mathbf{W}, \mathbf{u}) = \mathbf{W}_{sp}\mathbf{u} \tag{5.15}$$

where $\mathbf{W}$ is the *programmed weight matrix* and $\mathbf{W}_{sp}$ the *effective sneak path weight matrix* which has the same size as the RCA, which is $n \times m$.

Figure 5.26a shows the normalized measured effective weights for differential crossbar array with 1Ω wire resistance for two crossbar arrays, 256×256 array and 128×128 array, filled with random data. The measured weights decrease exponentially across the diagonal. Clearly, increasing the crossbar array size increases the IR voltage drop across the array, creating more sneak paths. Figure 5.26b also shows the histogram of random data with 1-bit ternary quantization (i.e -1, 0, 1). The histogram of the 256×256 array has smaller mean value

and bigger standard variation compared to the 128×128 array. In addition, Figure 5.26c shows the effect of the wire resistance on the histogram of the measured data for $1\Omega, 5\Omega$ and 10Ω wire resistance for the 128×128 array. The higher the wire resistance, the more severe the sneak path. Figures 5.26 (d) and (e) show histograms of the quantized states for 3-bit and 4-bit weights for the 128×128 case. Ideally, each state should be a narrow pulse and non-overlapped but because of the sneak path problem, it becomes wider and overlapped.

5.7.2 Proposed IR-QNN Training and Inference

QNN Training Framework

Thanks to differential weight realization, it is possible to realize $2^n + 1$ states where $n = 1, 2, 3, 4$ is the device's precision in bits. We use binarized activation function $\{-1, 1\}$ for simple and fast communication between the crossbar layers and eliminate area- and power-expensive blocks such as ADCs and DACs.

IR-QNN framework is an extended version of BinaryNet [142] to support more weight states and a bipolar activation function. During training, real-valued weights are quantized through stochastic rounding to the equally distributed point sets within $[-1, 1]$. Activations are binarized into $\{-1, 1\}$. MLP and CNN models are used for MNIST and CIFAR10 dataset, respectively (see Table 5.9 and 5.10). Convolution filters are 4D tensors but reshaped to 2D matrices with the number of output channels as the leading dimension, so that convolution can be performed by matrix multiplication.

The modifications to the BinaryNet framework, to include higher quantized states and the sneak path estimation technique in the forward and backward computation, are shown in Algorithm 5.2. In the forward computation section, the modifications are as follows: (1) partitioning the quantized weights of each layer, W_k^b, into small weight matrices P_k^b, (2)

application of the sneak path estimation method (*SPestimate* function) to each partitioned array to create W_{sp} to be used in (3) the forward inference and (4) backward computation instead of the quantized weights W^q. Similar modifications can be added to other QNN frameworks to capture the effect of the sneak path problem.

Algorithm 5.2 Proposed IR-QNN Training Algorithm

Require: a minibatch of inputs and targets (a_0, a), previous weights W, device precision n_b, previous BatchNorm parameters θ, weights initialization coefficients from γ, and previous learning rate η

Ensure: updated weights W^{t+1}, updated BatchNorm parameters θ^{t+1} and updated learning rate η^{t+1}.

{1. Computing the parameters gradients:}
{1.1.Forward propagation:}
for $k = 1$ to L **do**
  $W_k^q \leftarrow \text{Quantize}(W_k, nb)$
  $P_k^q \leftarrow \text{Partition}(W_k^q)$
  $W_{sp_k} \leftarrow \text{SPestimate}(P_k^q)$
  $s_k \leftarrow a_{k-1}^b W_{sp_k}$
  $a_k \leftarrow \text{BatchNorm }(s_k, \theta_k)$
  if $k < L$ **then**
    $a_k^b \leftarrow \text{Binarize }(a_k)$
  end if
end for
{1.2. Backward propagation:}
{Compute $g_{aL} = \frac{\partial C}{\partial a_L}$.}
for $k = L$ to 1 **do**
  if $k < L$ **then**
    $g_{a_k} \leftarrow g_{a_k}^b \circ 1_{|ak| \leq 1}$
  end if
  $(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm }(g_{a_k}, s_k, \theta_k)$
  $g_{a_{k-1}^b} \leftarrow g_{s_k} W_{sp_k}$
  $g_{W_k^q} \leftarrow g_{s_k}^T a_{k-1}^b$
end for
{2. Accumulating the parameters gradients:}
for $k = 1$ to L **do**
  $\theta_k^{t+1} \leftarrow \text{Update}(\theta_k, \eta, g_{\theta_k})$
  $W_k^{t+1} \leftarrow \text{Clip}(\text{Update}(W_k, \gamma_k \eta, g_{W_k^q}), -1, 1)$
  $\eta^{t+1} \leftarrow \lambda \eta$
end for

System Level Estimation of Sneak Path Problem

In this section, we introduce different methods to estimate the sneak path problem without the need for SPICE or numerical simulations.

Training with multiplicative noise

It is clear in Fig. 5.26 that each state is spread with a certain statistical distribution due to the sneak paths. One way to overcome the sneak path issue and to enable quick estimation of realistic weights, is to create a statistical model for each state and include it into the DNN framework which can be done as follows:

$$\mathbf{W}_{sp} = \sum_{i=1}^{Q} \mathbf{W}_i^q \odot \mathbf{n}_{sp_i} \tag{5.16}$$

where $\mathbf{W}_{sp}$ is the wire-resistance-effect-compensated weight matrix, $\mathbf{W}_i^q$ is the quantized weight matrix having i^{th} state (such that $\sum_i \mathbf{W}_i^q$ equals the quantized weight matrix), Q the number of states, $\odot$ element-wise multiplication and $\mathbf{n}_{sp_i}$ is multiplicative noise of ith state.

Each state is statistically curve-fitted to different distributions and we found that the log-normal distribution is the best (e.g., the highest likelihood) to describe the noise per state. The positive states and negative corresponding states have similar histograms. Table 5.8 shows the curve-fitted model parameters for different wire resistance simulating different sneak path scenarios.

Although the method would have the same effect on summed current per column, this method is not very effective since it treats all the locations in the array equally, which does not describe the real behavior of the sneak path problem as shown in Fig. 5.26a.

Table 5.8: Fitted Lognormal distributions of multiplicative noise for each state.

		$R_w = 1\Omega$		$R_w = 5\Omega$		$R_w = 10\Omega$	
	State	μ	σ	μ	σ	μ	σ
1 bit	1	-0.362	0.121	-1.337	0.460	-2.118	0.750
2 bit	0.5	-0.941	0.112	-1.859	0.427	-2.605	0.707
	1	-0.343	0.113	-1.273	0.437	-2.031	0.717
3 bit	0.25	-1.466	0.109	-2.341	0.409	-3.066	0.680
	0.5	-0.926	0.107	-1.811	0.412	-2.540	0.682
	0.75	-0.580	0.107	-1.468	0.412	-2.204	0.687
	1	-0.328	0.108	-1.228	0.419	-1.964	0.693
4 bit	0.125	-1.924	0.109	-2.783	0.397	-3.500	0.665
	0.25	-1.453	0.105	-2.317	0.397	-3.034	0.663
	0.375	-1.144	0.104	-2.010	0.400	-2.732	0.668
	0.5	-0.914	0.103	-1.784	0.401	-2.506	0.668
	0.625	-0.723	0.104	-1.597	0.404	-2.321	0.671
	0.75	-0.567	0.104	-1.445	0.405	-2.173	0.673
	0.875	-0.432	0.104	-1.312	0.406	-2.041	0.674
	1	-0.316	0.104	-1.198	0.408	-1.931	0.678

Training with Masks

Another solution is to generate average mask that can account for the cell location in the array. This mask is element-wise multiplied by the quantized weight matrix similar to [134], as follows:

$$\mathbf{W}_{sp} = \mathbf{W}^q \odot \mathbf{M} \tag{5.17}$$

where $\mathbf{M}$ is the average mask matrix.

This mask method helps to predict more realistic behavior of a crossbar array and can be easily calculated with fabricated crossbar array. The mask matrix is generated from either SPICE simulations or equivalent numerical methods[5], and is normalized to the ideal desired current. Masks are generated by averaging the results of many (e.g., 1000) SPICE simulations using random input weight matrices [134, 64].

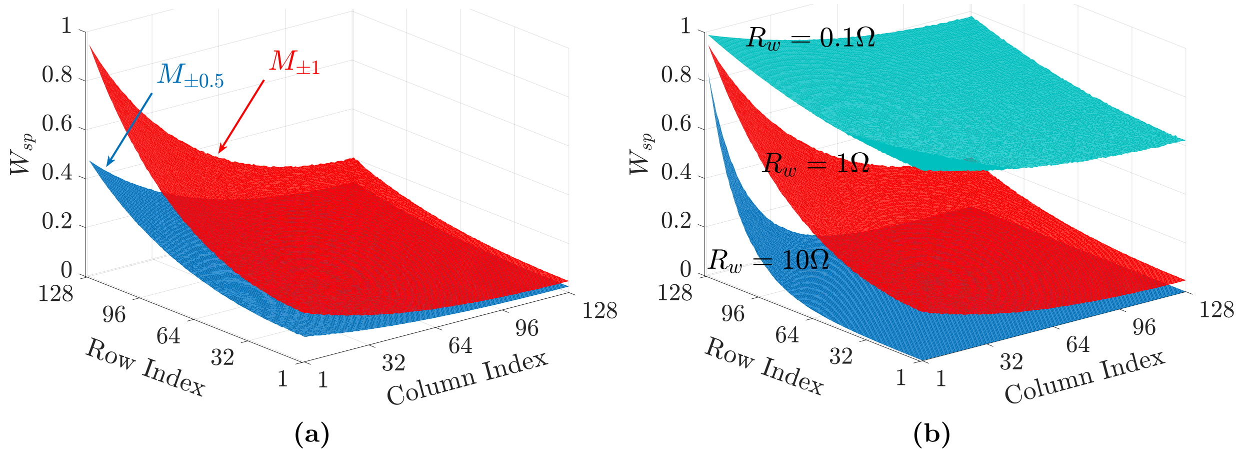


Figure 5.28: The generated masks for 2-bit neural network training.

Due to the averaging, the generated mask is static and fixed. However, in practice, $\mathbf{W}_{sp}$ has some stochastic behaviour around this average mask. Thus, an additive white Gaussian noise can be added to the mask to exhibits more practical behaviour, which we refer to as stochastic mask.

The third solution is to generate a mask for each state and element-wise multiplied by its corresponding state matrix. This solution can be mathematically formulated as follows:

$$\mathbf{W}_{sp} = \sum_{i=1}^{Q} \mathbf{W}_i^q \odot \mathbf{M}_i \tag{5.18}$$

where $\mathbf{M}_i$ is the corresponding mask matrix of ith state.

Figure 5.28 shows mask examples for training a 2-bit neural network having 5 states per weight. Figure 5.28a shows the $M_{\pm 0.5}$ and $M_{\pm 1}$ masks for 1Ω wire resistance as an example. In addition, Figure 5.28b shows the effect of the wire resistance on the $M_{\pm 1}$ mask. It is clear the high degradation with higher wire resistance values. It is worth mentioning that applying masks during training has a negligible effect on the training time.

5.7.3 Batch Normalization during Inference

One of the important practices in training deep neural networks is adding a batch normalization layers to speed up the training, improve the performance and overcome vanishing gradient problem in DNNs, without the need for small learning rates which slow down the convergence [143]. In other words, the batch normalization is data whitening (i.e removing the mean and variance of the data) which can be mathematically defined as follows for j^{th} neuron

$$\mathbf{BN}(\mathbf{y}_j) = \frac{\mathbf{y}_j - \mu_j}{\sigma_j}\gamma_j + \beta_j \tag{5.19}$$

where μ_j and σ_j are the mean and the standard deviation values of the input vector $\mathbf{y}_j$, and γ_j and β_j are trainable parameters.

During the inference, these batch normalization layers can be removed by merging them with the preceding layers. As aforementioned, we use the sign activation function, thus the output activation o can be computed as

$$\begin{aligned} o_j &= sign\left(\mathbf{BN}\left(\sum_i w_{ij}V_i + b_j\right)\right) \\ &= sign\left(\frac{\gamma_j}{\sigma_j}\left(\sum_i w_{ij}V_i + b_j - \mu_j + \frac{\sigma_j}{\gamma_j}\beta_j\right)\right) \end{aligned} \tag{5.20}$$

Using $sign(AB) = sign(sign(A)B)$ property yields

$$o_j = sign\left(sign\left(\frac{\gamma_j}{\sigma_j}\right)\left(\sum_i w_{ij}V_i + b_j - \mu_j + \frac{\sigma_j}{\gamma_j}\beta_j\right)\right) \tag{5.21}$$

Since σ always has positive value, batch normalized layer merged with the proceeding layer

Table 5.9: MLP network configuration (MNIST dataset)

layer	type	#output	#input
1	fully connected	512	784
2		512	512
3		512	512
4		10	512

Table 5.10: CNN configuration (CIFAR10 dataset)

layer	type	#output channels	#input channels	filter size
1	convolution	64	3	3x3
2		64	64	3x3
–	max pooling	–	–	2x2
3	convolution	128	64	3x3
4		128	128	3x3
–	max pooling	–	–	2x2
5	convolution	256	128	3x3
6		256	256	3x3
–	max pooling	–	–	2x2
7	fully connected	1024	4096	–
8		1024	1024	–
9		10	1024	–

can be written as

$$o_j = sign\left(\sum_i \tilde{w}_{ij} V_i + \tilde{b}_j\right) \tag{5.22}$$

where $\tilde{w}_{ij} = sign(\gamma_j) w_{ij}$ and $\tilde{b}_j = sign(\gamma_j)(b_j - \mu_j + \frac{\sigma_j}{\gamma_j}\beta_j)$. The matrix form of (5.22) can be written as

$$\mathbf{O} = sign\left(\tilde{\mathbf{W}}\mathbf{V} + \tilde{\mathbf{b}}\right) \tag{5.23}$$

where $\tilde{\mathbf{W}} = \mathbf{W}\mathbf{D}_{sign(\gamma)}$ and $\tilde{\mathbf{b}} = \left(\boldsymbol{b} - \boldsymbol{\mu} + \boldsymbol{D_\sigma} \boldsymbol{D}_{\frac{1}{\gamma}} \boldsymbol{\beta}\right) \mathbf{D}_{sign(\gamma)}$, where $\mathbf{D}_v$ is a diagonal matrix whose diagonal is v. It is worth highlighting that the weight parameters are kept quantized even after merging batch normalization.

Table 5.11: Validation accuracy without retraining.

	1-bit				2-bit				3-bit		4-bit	
	MNIST		CIFAR10		MNIST		CIFAR10		MNIST	CIFAR10	MNIST	CIFAR10
Map-	I	II	I	II	I	II	I	II	I & II	I & II	I & II	I & II
1Ω	87.31	11.35	12.65	10.00	92.74	24.87	33.38	10.00	92.92	41.26	94.90	62.32
5Ω	11.68	11.35	10.00	10.00	21.05	11.96	10.14	10.00	24.44	10.00	14.18	10.18
10Ω	11.35	11.35	10.00	10.00	11.36	11.45	10.00	10.00	10.27	10.39	11.83	10.00

Table 5.12: MNIST dataset validation results using Mapping-I after retraining with M. Noise: Multiplicative Noise, Single mask set, Sto. Mask: Stochastic Mask and Multiple Mask set

	1-bit				2-bit				3-bit				4-bit			
	M. Noise	Single Mask	Sto. Mask	Multiple Mask	M. Noise	Single Mask	Sto. Mask	Multiple Mask	M. Noise	Single Mask	Sto. Mask	Multiple Mask	M. Noise	Single Mask	Sto. Mask	Multiple Mask
1Ω	98.16	98.34	98.24	98.44	96.75	98.03	98.02	98.32	89.92	97.39	97.4	98.36	85.94	96.98	97.16	98.28
5Ω	95.55	97.36	97.54	97.70	95.07	97.59	97.72	97.48	92.91	97.43	97.48	97.3	89.15	97.09	96.93	97.31
10Ω	91.03	96.96	97.03	97.04	93.3	97.37	97.37	96.93	91.53	97.23	97.13	96.83	83.95	96.84	96.82	96.13

Table 5.13: MNIST dataset validation results using Mapping-II after retraining with stochastic mask and multiple mask sets.

	1-bit		2-bit		3-bit		4-bit	
	Stochastic Mask	Multiple Mask	Stochastic Mask	Multiple Mask	Stochastic Mask	Multiple Mask	Stochastic Mask	Multiple Mask
1Ω	98.35	98.33	97.98	98.48	97.4	98.36	97.03	98.41
5Ω	98.27	98.27	98.08	98.11	97.48	97.3	97.5	97.36
10Ω	98.06	98.01	97.9	97.92	97.13	96.83	97.22	96.69

5.7.4 QNN Experimental Setup

To evaluate the effectiveness of our proposed technique we use the MNIST and CIFAR10 datasets [144, 145]. For each dataset, we use the same network architecture as given in BinaryNet [142], with these two changes: (1) instead of binary weights we use up to 4-bit (or 17-state) quantized weights, (2) the model sizes are reduced. For MNIST the number of hidden neurons is reduced to one-fourth and for CIFAR10, the number of channels is reduced to half, roughly reducing the number of model parameters to about 1/16 and 1/4, respectively. Details of the networks can be found in Table 5.9 and 5.10.

The experiments are done in three main steps. The first step is the baseline training, which uses floating-point weights/activations to obtain the best test accuracy, where test accuracy is the ratio of the correctly recognized samples for unseen data. We use the default training parameters for 100 epochs of MNIST training and 500 epochs of CIFAR10. Learning rates halve every 20 or 50 epochs for MNIST and CIFAR10, respectively, initially from 2^{-6}. The baseline accuracies are 98.4% for MNIST and 88.5% for CIFAR10, which is similar to the best accuracies for the datasets reported in the literature. (The accuracy reported is test accuracy, that is, the inference accuracy for unseen data.)

The second step is fine-tuning, which is running additional training iterations using the weight from the first step as the initial weight. While the weight in the first step gives a very high accuracy on GPU, it is unlikely to give good results if used for RRAM crossbar arrays due to distorted MVM computation caused by IR drop and sneak paths. The fine-tuning re-trains the networks to mitigate the discrepancy, by using the aforementioned mask methods for different wire resistances and quantization levels. During fine-tuning, we use learning rates starting from 2^{-9}, training additional 50/200 epochs for MNIST/CIFAR10 models. The other parameters remain the same as the baseline training. At the beginning of fine-tuning, the accuracy plummets due to the introduction of mask but eventually recovers

Table 5.14: CIFAR10 dataset validation results after retraining.

R_w	1-bit				2-bit				3-bit		4-bit	
	Map-I		Map-II		Map-I		Map-II		Map-I & II		Map-I & II	
	Stochastic Mask	Multiple Mask	Stochastic Mask	Multiple Mask	Stochastic Mask	Multiple Mask	Stochastic Mask	Multiple Mask	Stochastic Mask	Multiple Mask	Stochastic Mask	Multiple Mask
1Ω	86.53	87.05	86.79	87.07	76.99	87.27	76.95	87.24	63.39	87.31	59.98	87.04
5Ω	83.95	83.62	86.32	86.71	75.84	85.2	78.93	86.85	65.14	85.96	58.79	85.21
10Ω	81.75	82.35	86.39	86.27	72.32	83.54	76.14	85.95	61.32	84.39	55.7	83.82

through fine-tuning. Note that the accuracy at the end of fine-tuning is not indicative of the real performance of RRAM crossbar arrays, for which we need a separate validation step.

The third step is validation. The output of the second step is quantized weights to be programmed to RRAM crossbar arrays. Our *validation setup* takes the quantized weights, and runs SPICE-based RRAM crossbar simulation, to get the effective output currents. The effective output currents are fed back to our QNN inference framework to obtain network-level inference results. Note that neither training nor mask is used during validation. The test accuracy obtained from validation is what we can expect to see if the quantized weights are perfectly programmed to RRAM crossbar arrays, barring stochastic and other unmodeled noise/faults during RRAM read. Some of those nonidealities are considered in our additional experiments (see Section 5.8).

5.8 QNN Results and Discussion

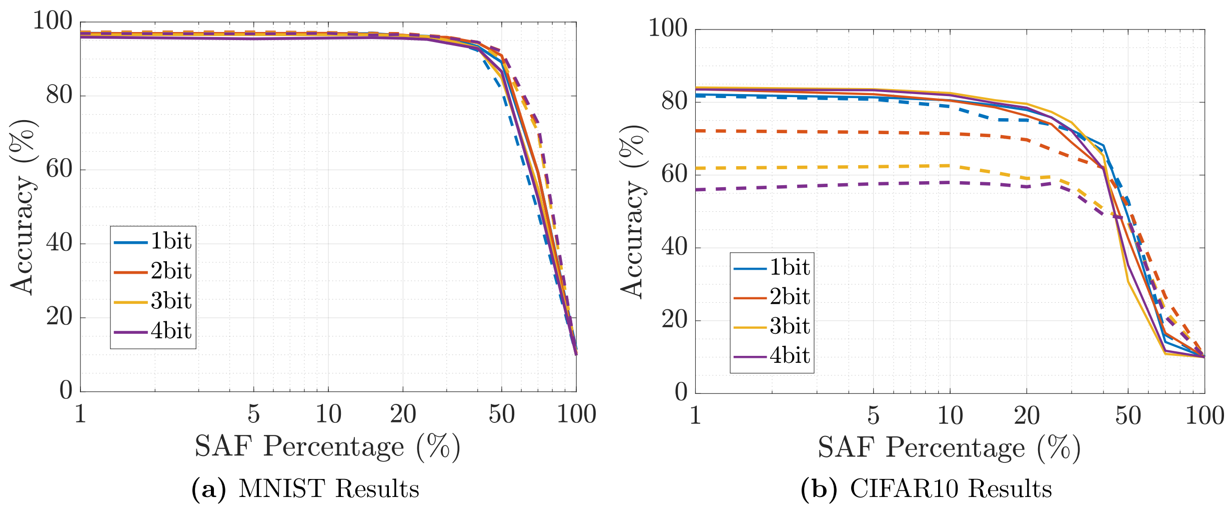
In this work, we consider three test scenarios; with 1Ω, 5Ω and 10Ω wire resistances to consider different technology nodes. For instance for 50 nm feature size, it is expected to have around 5Ω wire resistance [5, 25]. The results shown in Table 5.11 illustrate that without considering the sneak path problem in the training, the accuracy drops to around $10 \sim 12\%$ from the baseline test accuracies regardless of the number of bits.

After the retraining using the proposed techniques, the networks were able to reach close to the baseline accuracies. Tables 5.12 and 5.14 show the validation accuracy for MNIST

and CIFAR10 datasets for different wire resistance scenarios and different mappings. The higher the wire resistance the higher drop in the performance. Clearly mapping-II shows a better performance for 1-bit and 2-bit cases since the used resistance values are much higher than the one used for mapping-I and the severity of the sneak path problem is determined by the ratio between device's LRS and the wire resistance value. The smaller the ratio, the severe the sneak path problem. In general, training with multiple mask set achieves the best performance among the proposed solutions. On the other hand, increasing the number or bits does not improve the performance monotonically. The accuracy drops with increasing the number of bits to more than 2 bits, which is attributed to the overlap between states, as illustrated in Fig. 5.26. The drop in CIFAR10 test accuracy is much higher than MNIST test accuracy due to its high sensitivity to weight variations. Clearly, from these results, the proposed training method provides the best performance with 2-bit devices.

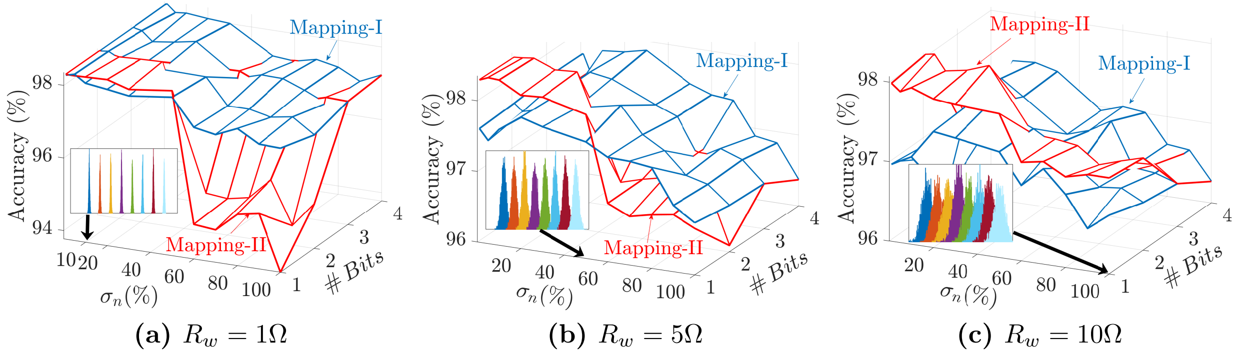
5.8.1 Stuck-At Fault Effect

Figure 5.29 shows the recognition accuracy with changing the SAF percentage for 10Ω wire resistance scenario. Clearly, the performance has no drop up to 50% and 20% SAF percentage for MNIST and CIFAR10 datasets, respectively thanks to the existence of zero state which mitigates reduces the effect of SAF devices. The results show that the SAF causes accuracy drop regardless the weight precision. The higher weight precision has a slight accuracy improvement after knee point. In conclusion, there is no need to retrain the network with the full knowledge of the SAF devices' locations, especially that the reported SAF percentages are less than 20% [79, 80].



(a) MNIST Results **(b)** CIFAR10 Results

Figure 5.29: Effect of changing the SAF percentage on the recognition accuracy using mapping-I. Solid line refers to multiple mask trained network results and dashed line shows stochastic mask trained network results.



(a) $R_w = 1\Omega$ **(b)** $R_w = 5\Omega$ **(c)** $R_w = 10\Omega$

Figure 5.30: Effect of changing the variability of each conductance's state on the recognition accuracy using mapping II. Dashed lines show the validation accuracy drop with pre-trained weights with multiple mask technique. Solid lines show the accuracy drop after retraining with additive noise.

5.8.2 Effect of Device Variability

In order to achieve low variation in each conductance's state i.e precise programming, error-correcting techniques (ECCs) such as write-verify technique are usually used. Using such ECCs would require multiple writes and read cycles which would increase the programming time of the entire crossbar array. In addition, a write-disturb problem occurs where writing some cells might disturb the written data in other cells [146]. With multiple writes to the same cell, a higher rate of the disturbed cell occurs. Thus, it is better to write once and take the variability into consideration during the training and validation. In order to consider these device's variabilities, we consider adding Gaussian noise to each conductance's state where we vary the the normalized standard deviation, σ_n, which is normalized to the difference between the states ($\Delta g = 6\mu S$).

Figure 5.30 shows the effect of changing the normalized standard deviation on the MNIST network. In this experiment, we used the trained model with stochastic mask and sampled from the measured data shown in Fig. 6.3b without performing any retraining to include new noise. Clearly, the performance slightly drop with increasing σ_n. The best accuracy results are obtained around $\sigma_n = 0.25$, since we used the parameters which are trained with measured values of the conductances, shown in Fig. 5.2. Although, the high overlapping between the conductance states at σ_n, the performance dropped around 1.5% at worst case for mapping-I. Approximately, the performance is the same regardless the number of bits for mapping-I. On the other hand, mapping-II is more sensitive to the variations. The performance is dropped around 4% for 1-bit and 2-bit cases with 1Ω wire resistance after $\sigma_n = 0.5$. This drop is caused by narrow spacing between the states in mapping-II. Clearly, training with higher wire resistance values, reduces the drop in the performance to 1.5%.

5.8.3 Effect of Limited Retention

The main factor that would affect the performance of the DNNs over time (i.e. aging) is the RRAMs' retention. Recent works show different retention values based on device structure and materials. These works also show that the device would drift over the time towards a very low conductance state, G_f, which is less than the formed low conductance state, G_{min}. In addition, the drift speed is a function of the temperature. The higher the temperature is, the higher drift occurs [147]. Due to the lack of aging model, we adopt the following retention model to be incorporated in the validation simulations to study the performance degradation with time. We would emphasize that the aging was taken into consideration in the training process to simulate practical scenarios. The conductance change versus time can be model as follows

$$G(t) = G_i - (G_i - G_f) \left(\frac{e^{v_d t_n} - 1}{v_d - 1} \right) \tag{5.24}$$

where G_i is the initial conductance state, v_d is the drift coefficient, and t_n normalized retention time which is normalized to the retention value of the device. We chose this normalized model to simulate different RRAMs' behaviors with different drift coefficients.

Figure 5.31 shows the effect of aging on the validation accuracy of MNIST dataset for two scenarios $v_d = 10$ and 0.1 with 25% variability in the normalized retention time to simulate different device conditions. Clearly, the network was able to achieve the baseline accuracy for more than the 50% of the retention time. Then, we start seeing performance degradation regardless of the number of bits. The accuracy degrades faster for a smaller drift coefficient.

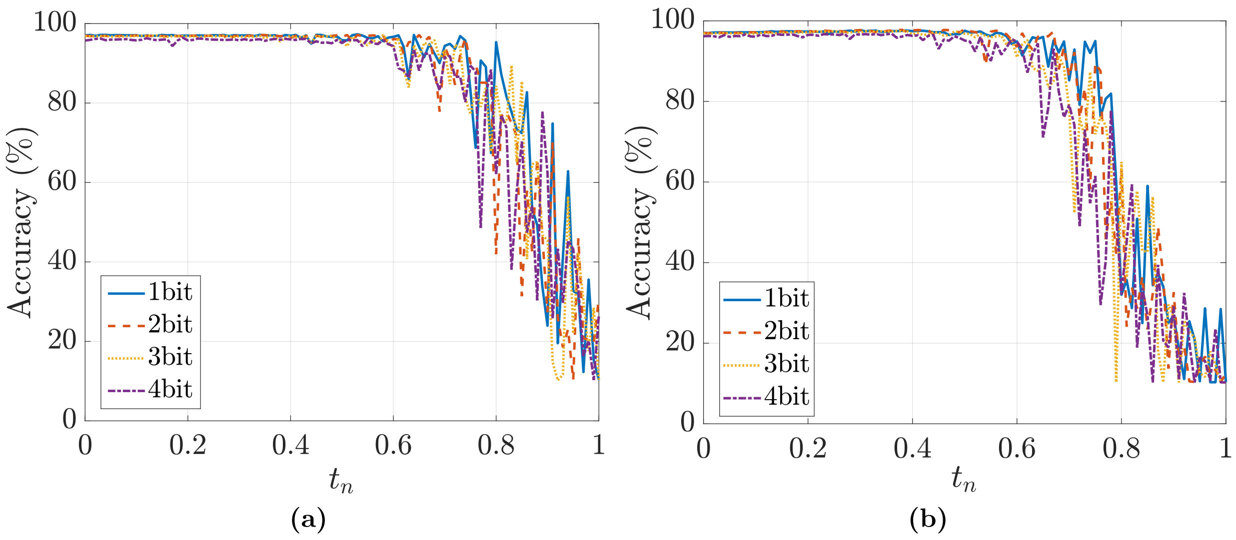


Figure 5.31: MNIST Recognition Accuracy against normalized retention time for a) $v_d = 10$, and b) $v_d = 0.1$.

5.8.4 Power and Area Results

The power dissipation during the inference consists of the power dissipation of RCAs and the power dissipation in the peripheral circuits. However, due to the resistive nature of RCAs, the power are mainly consumed inside the RCAs. Figure 5.32 shows the power dissipation of RCAs for processing one input image at $0.1V$. Clearly mapping-II consumes around 20% and 35% of the power consumed in mapping-I for 1-bit and 2-bit cases, respectively. It is worth to highlight that the power consumption of using the model trained with average mask is the same as the trained with multiple mask set. Using the hardware setup discussed in [134], the total power consumption per image is estimated to be around 0.9 W and 132 W for MNIST and CIFAR10 networks where RCAs consumes 65.8% and 69.65% of the total power while the rest is consumed in the sensing circuits and memory cells needed to store intermediate stages while pipelining. on the other hand the, the total area is estimated to be around $0.741nm^2$ and $0.112\mu m^2$ for MNIST and CIFAR10 networks, respectively, distributed as $\{17.68\%, 45.53\%, 36.78\%\}$ and $\{29.71\%, 38.88\%, 31.41\%\}$ for RCAs, peripheral circuits and storage cells, respectively, using 5nm technology node. According to aforementioned power and area results, this hardware is able to achieve $204Tops/W$ and $239Tops/W$

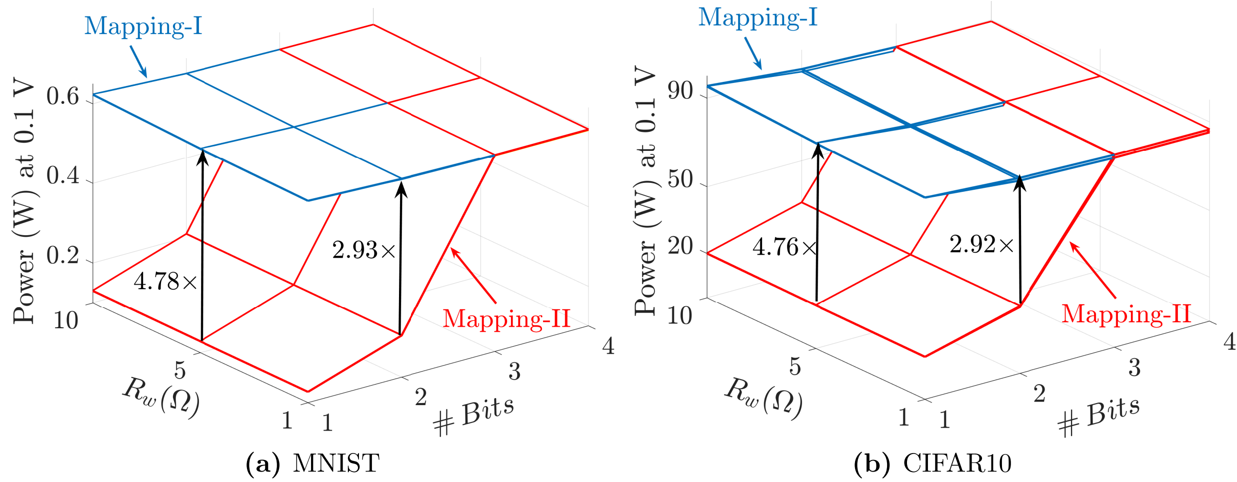


Figure 5.32: Static power dissipation in RCAs per image at $0.1V$ read voltage.

with $1.23KW/mm^2$ and $1.175KW/mm^2$ power density for MNIST and CIFAR10 networks, respectively, at 100MHz operating frequency.

5.8.5 Time Overhead Comparison

In order to show the efficiency of the proposed technique, we compare the training time per iteration with the mask versus the baseline training. Figure 5.25 shows the training time comparison. The baseline training is 1.38×, 1.026× and 1.013× faster than the training with the mask for MNIST, CIFAR10 and SVHN, respectively. The time overhead is negligible if compared to the gained improvement in the performance. And, it is much faster compared to running SPICE simulations for sneak path problem during the training, which takes around $20 \sim 30$ minutes per iteration for MNIST.

5.9 Driver and Neuronal Circuits Requirements

As previously discussed in section IV, we trained the the networks to have binary activation function, $\{-1, 1\}$, for efficient communication and buffering between the fully connected layers. Three nonidealities needs to considered while designing the periphery circuits:

- Driver output resistance: Each crossbar array is driven by a driver or buffer circuit. The output resistance of the driver circuit creates a voltage divider with the parallel RRAMs within the same driven row in addition to the sneak paths loading.
- Neuronal input resistance: After the current is summed within the crossbar array, a current sensing circuit is needed to sense the summed current from positive array and compare it with the summed current from negative array, and give a positive or negative output voltage. The input resistance of the sensing circuit create extra loading to the crossbar array.

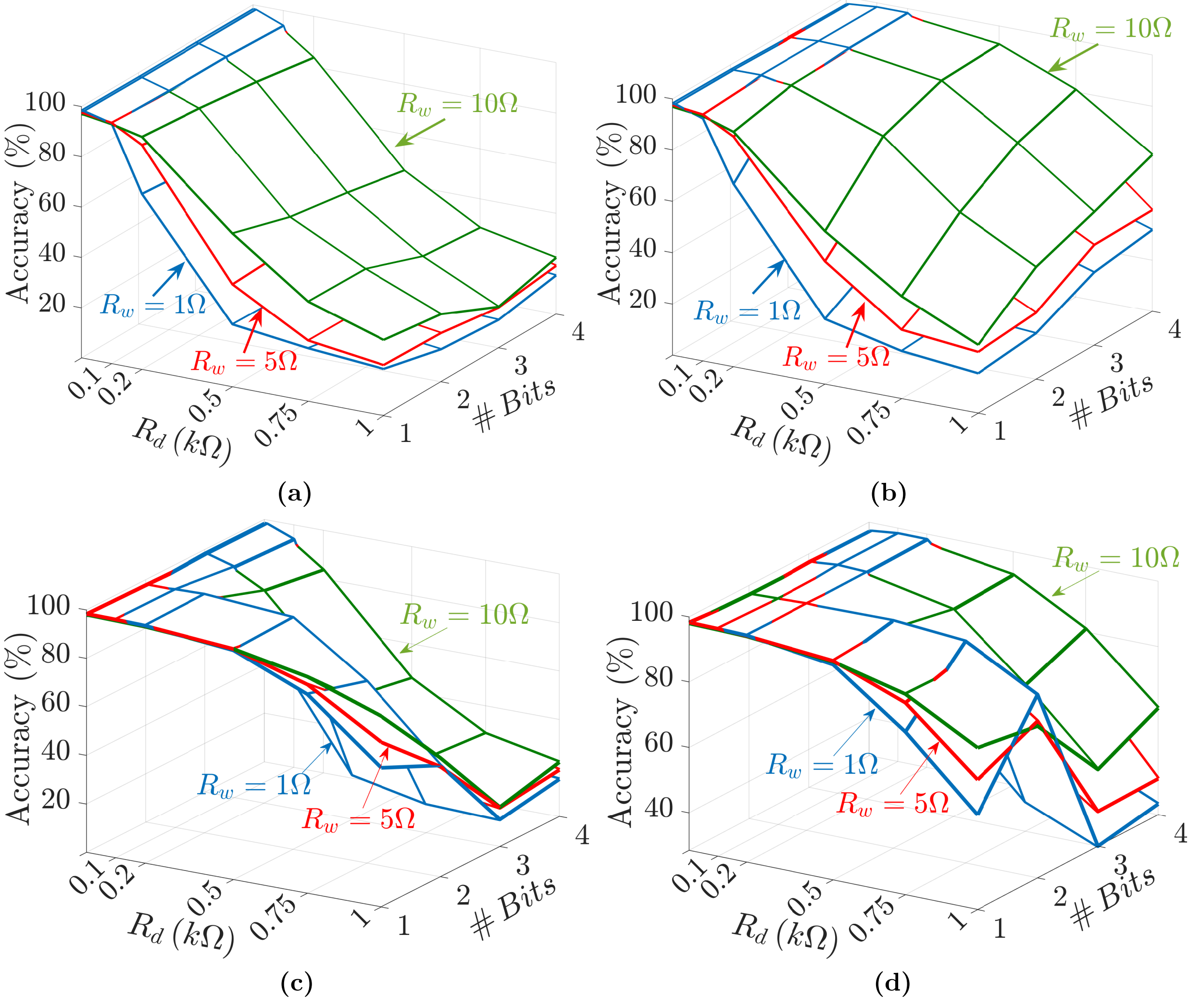


Figure 5.33: Effect of the driver resistance on the performance of MNIST recognition; (a) and (b) for Mapping-I and (c) and (d) for mapping-II with multiple mask training for (a) and (c) and with stochastic mask training for (b) and (d).

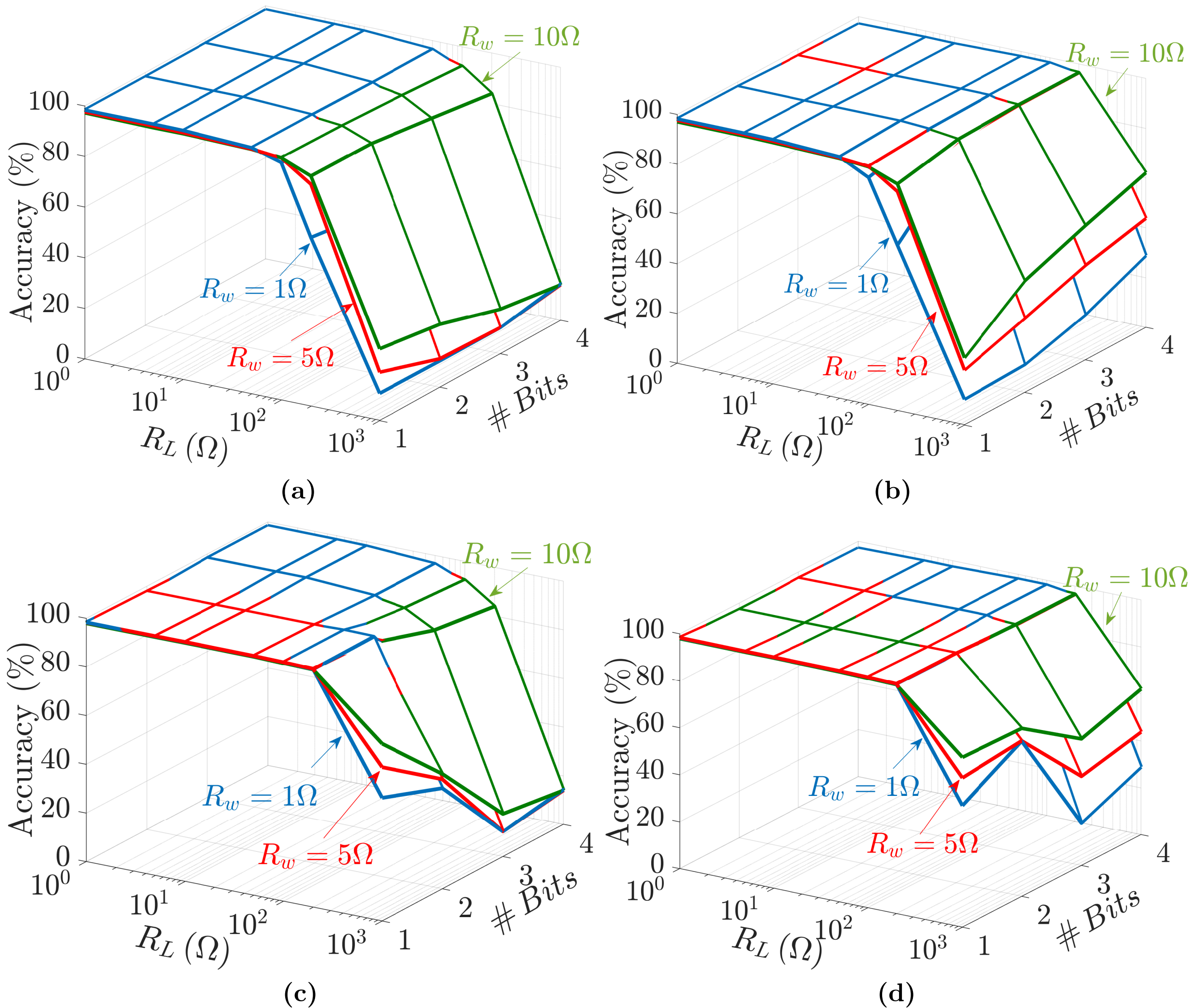


Figure 5.34: Effect of the load resistance (the input resistance of the sensing circuit) on the performance of MNIST recognition; (a) and (b) for Mapping-I and (c) and (d) for mapping-II with multiple mask training for (a) and (c) and with stochastic mask training for (b) and (d).

- Neuronal circuit variability: Due to PVT variations of the circuit, the comparison between current sensed from positive and negative RCAs is biased to one of them with random value.

These nonidealities disturb the MVM computation which affects the DNN performance. With well designed circuits, there is no need to consider them during training. Ideally, the driver circuit should have zero output resistance and the neuronal circuit should have zero input resistance and zero offset.

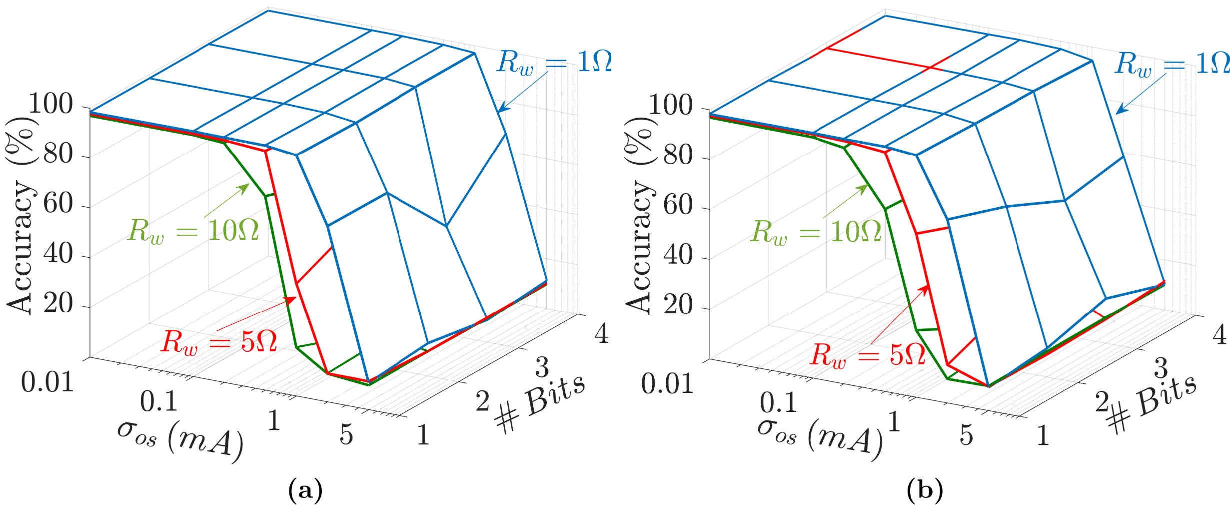


Figure 5.35: Effect of the neuronal offset current deviation on the performance of MNIST recognition for Mapping-I; (a)multiple mask training and (b) stochastic mask training.

In this section, we study the effect of these nonidealities on the MNIST network performance to find the maximum values that the network can tolerate without affecting the performance. It is worth to highlight that there is no retaining with peripheral circuits nonidealities is performed. Including them in the training will relax the design requirements. Although, we see it is more beneficial to consider the worst case.

Figure 5.33 shows performance degradation due to the output resistance of the driver circuit with changing the number of bits and wire resistance. The results shows that the trained network with multiple mask set can tolerate higher driver resistance especially with higher number of bits. In addition, mapping-II exhibits better behavior compared to mapping-I. Similarly, the effect of the load resistance is studied and shown in Fig. 5.34 for different wire resistance, number of bits and different training methods. The performance degradation due to increasing the load resistance is the same for any number of bits. Training with higher wire resistance value exhibits better performance against load and driver resistances. The effect of changing the standard deviation of the offset current on the recognition accuracy is depicted in Fig. 5.35. Per these results, the network can tolerate up to $0.1mA, 0.5mA$ and $1mA$ standard deviation of the offset current for $1\Omega, 5\Omega$ and 10Ω wire resistance, respectively, regardless the number of bits per device.

5.10 Conclusion

In this chapter, we showed the high performance degradation in BNNs due to the existence of crossbar array non-idealities especially the wire resistance. We presented a simple yet very effective technique to include the crossbar wire resistance in the training of deep neural networks. The proposed mask technique is fast and efficient training method to incorporate the wire resistance without any numerical or SPICE simulations with a negligible time overhead. In addition, the proposed technique can be easily integrated into any neural network framework. Different cases have been studied showing a significant improvement in the performance after retraining the different networks with the proposed mask method. Per our results, increasing the wire resistance is useful to decrease the power consumption as long as the same accuracy is maintained which is achievable using the proposed technique.

In this work, the mask is generated using SPICE simulations to capture the effect of the wire resistance. The same method can be used for fabricated crossbar arrays where the mask can be generated from real measurements of the fabricated crossbar arrays which will be more accurate and help to overcome other nonidealities such as RRAM's variability and stuck-at-fault devices. In addition, the proposed mask technique can be generalized and applied for multilevel RRAM devices to build quantized neural networks.

The chapter proposed software-level techniques to incorporate the sneak path problem (IR drop) in training the deep quantized neural networks. A comparison among the proposed methods is introduced for different sneak path scenarios. We studied the effect of other nonidealities such as device variability, stuck-at faults and aging. Our results shows that 2-bit device exhibits the best performance and training with multiple mask set. As a conclusion from our experiments, we recommend using a driver with output resistance to be less than 200Ω and input resistance of the sensing circuit should be less than 100Ω. In addition, the input referred current offset deviation should be less than $0.1mA$.

In this work, we considered only the effect of the wire resistance causing the sneak path problem in addition to other non-idealities from the driver and neuronal circuits, SAF etc. However, some non-idealities resulting from capacitive parasitics and routing of the partitioned crossbar arrays need more study. Such non-idealities require physical simulations to extract parasitic models to be considered in the software-based simulations. These non-idealites are out of the scope of this work and will be pursued in future works.

Chapter 6

Efficient Online Learning

The last decade witnessed significant progress in developing neuromorphic hardware that can solve large scale tasks much faster and with online learning capabilities unlike Von-Neumann based processor architectures [21, 148, 23, 149]. Von-Neumann architectures suffer from the memory-wall bottleneck, especially when performing neuromorphic tasks where massive data blocks are transferred from the memory to processing nodes [150]. Recently developed solid-state devices have shown non-volatile storage capabilities such as RRAM (i.e memristors), phase-change memory, and spin-transfer torque-RAM [151]. A major advantage of these devices is the ability to be assembled as a crossbar array that enables the Matrix-Vector Multiplication (MVM) operation to be completed in a single step. This is unlike other hardware solutions that require $N \times M$ steps, where N and M are the dimensions of the weight matrix.

Several RRAM devices demonstrating promising synaptic behaviors are characterized by nonlinear and asymmetric update dynamics, which is a major obstacle for large-scale deployment in neural networks [65], especially for learning tasks. Applying the vanilla back-propagation algorithms without taking into consideration the device non-idealities does not guarantee the convergence of the network. Thus, a closed-form model for the device non-linearity must be derived based on the device dynamics and added to the neural network training algorithm to guarantee convergence to the optimal point (minimal error).

Recently, different supervised learning techniques were proposed which are more energy and data-efficient such as random backpropagation, direct and indirect feedback alignment and surrogate gradient learning [152, 153, 154]. These techniques have not been utilized to enable more efficient hardware learning yet. The existing emerging hardware accelerators rely on stochastic gradient propagation [81, 61, 62, 155]. In this work, we discuss the local learning on memristive Spiking Neural Network for efficient write energy solutions.

6.1 Independent Component Analysis using RRAMs

The Internet of Things (IoT) market is growing exponentially and it is expected to have around 50 billion connected devices generating around 500 zettabytes of data per year by 2019 [14]. As a result, almost all IoT applications need a system to analyze patterns in this data, detect certain types of events and take decisions. Component analysis techniques [156] are promising candidates to perform these tasks especially with using local and efficient learning rules, that enable online learning, rather than the conventional techniques that require massive matrix operations. Additionally, by using hardware-based matrix-vector multiplication accelerators, a significant improvement in both performance and power can be achieved enabling handling the massive data generated from IoT [157].

Independent component analysis (ICA) is a very powerful tool to solve the cocktail party problem (blind source separation), feature extraction (sparse coding) and can be utilized in many applications such as de-noising images, Electroencephalograms (EEG) signals, and telecommunications [158].

As an illustrative example of cocktail party problem, it is required to recover signals that result from mixtures of sources that are statistically independent. An illustrative example can be given as follows; Figure 6.1a shows two observed signals. These signals are linear combinations (mixture) of two sources which are sinusoidal and square wave signals shown

in Fig. 6.1a. This problem can be mathematically modeled as

$$x_1 = a_{11}S_1 + a_{12}S_2$$
$$x_2 = a_{21}S_1 + a_{22}S_2 \tag{6.1}$$

where x_1 and x_2 are the observed signals, S_1 and S_2 are the original sources, and S_{11}, S_{12}, S_{21} and S_{22} are the mixing coefficients. In practice, both original sources and mixing coefficients are unknown. We only have the observed signals and it is required to recover the original sources. This problem is typically refereed to as the "cocktail party problem". In order to solve this problem, there are some constraints [158]; 1) the sources should be statistically independent (may have the same distribution) and (2) the sources are not Gaussian since sum of Gaussian variables is Gaussian as well. The solution key of the cocktail party problem is the statistical independence of the sources. Independent component analysis (ICA) can be applied to find the sources using different methods such as minimizing the mutual information between the components or maximizing non-Gaussianity between the components by using the higher-order statistics (such as kurtosis; the fourth standardized moment). Depending on the nature of the data, the appropriate ICA algorithm is chosen.

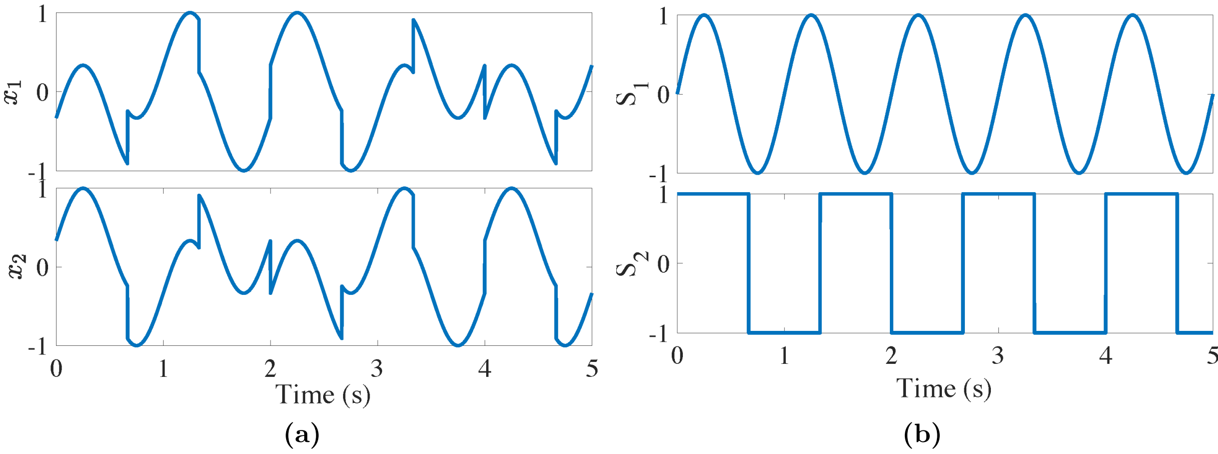


Figure 6.1: (a)Observed signals (mixtures of sources), and (b) original sources.

ICA consists of finding mutually independent and non-Gaussian hidden factors (components), $\mathbf{s}$, that form a set of signals or measurements, $\mathbf{x}$. This problem can be mathematically described for linearly mixed components as follows:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{6.2}$$

where A is the mixing matrix. Both $\mathbf{A}$ and $\mathbf{s}$ are unknowns. In order to find the independent components (sources), the problem can be formulated as $\mathbf{u} = \mathbf{w}\mathbf{x}$ where $\mathbf{x}$ is the mixed signals (inputs of ICA algorithm), $\mathbf{w}$ is the weight matrix (demixing matrix), $\mathbf{u}$ is the outputs of ICA algorithm (independent components). ICA's strength lies in utilizing the mutual statistical independence of components to separate the sources.

Resistive crossbar structures are considered a key enabler to hardware based neuromorphic acceleration due to their natural ability to do matrix-vector multiplication which is the basic operation for neural network acceleration (e.g. Multiply and Add). Crossbar arrays can perform matrix-vector multiplication in a single step as compared to m^2 steps for digital realizations, where m is the length of the vector. Furthermore, each RRAM occupies a very small area ($4F^2$), where F is the feature size, and operates as a nonvolatile continuous weight. In the digital realization, each weight is stored in at least a 32 bits register to have continuous weight which requires 38 transistors per bit. In addition, $m \times (m-1)$ multiply and accumulate (MAC) blocks are needed, increasing the power budget of the overall system. A comparative example of the network that was used for the De-mixing problem is given in the supplementary materials. RRAMs offer features such as high density, low power consumption, high endurance, high retention, high speed switching and 3D stack-ability in addition to ease of programming. Recently, RRAM-based neural network architectures have been deployed in brain-inspired computing applications[159] such as dimensionality reduction through PCA [160], sparse coding [161], reservoir computing [162] and image processing [81].

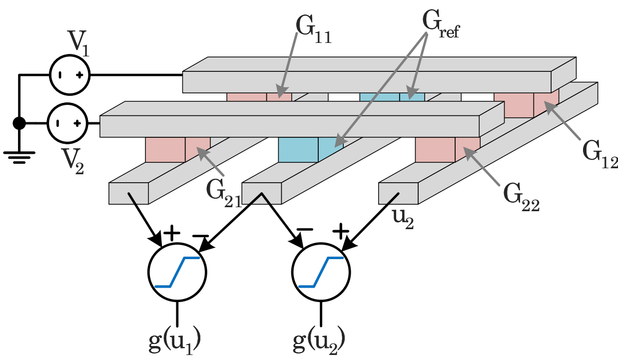


Figure 6.2: Illustration of used RRAM crossbar array to realize the ICA preceptron network.

Recently, Isomura and Toyoizumi have proposed an interesting biological plausible learning rule called *Error-Gated Hebbian Rule (EGHR)* [163] that enables local and efficient learning to find the independent components. But, the expensive part in this algorithm is the matrix-vector multiplication which can calculated using RRAMs-based crossbar array offering a low power and efficient solution. In this letter, a RRAM-based hardware realization for ICA is investigated using EGHR for online evaluation of the weights. We demonstrate that this learning rule is capable of local and efficient learning, even when taking into consideration the RRAM non-idealities, the asymmetric nonlinear conductance, and device variability.

6.1.1 Proposed RRAM's learning method

In this work, we consider the second weight realization technique, discussed in chapter V, and show that the algorithm converges despite the reduced dynamic range compared to the first realization.

Due to the importance of blind source separation problem, many algorithms and learning rules have been proposed to find the independent components such as minimizing the mutual information or maximum likelihood estimation [158] and Bell-Sejnowski or Amari [163], etc. Recently, Isomura and Toyoizumi have proposed an interesting biological plausible learning rule called *Error-Gated Hebbian Rule (EGHR)* inspired from the standard Hebbian rule [163].

In their work, the authors proved mathematically and numerically the efficiency of EGHR to achieve ICA. The EGHR learning rule can be written as

$$\dot{\mathbf{w}} = \eta\langle(E_o - E(\mathbf{u}))\, g(\mathbf{u})\mathbf{x}^T\rangle \tag{6.3}$$

where η is the learning rate, $\langle\cdot\rangle$ is the expectation over the ensemble (training samples), $g(u_i)x_j$ is the Hebbian learning rule, $g(u_i)$, x_j are the postsynaptic and presynaptic terms of the neuron, respectively, $(E_o - E(\mathbf{u}))$ is the global error signal which consists of E_o which is a constant, and $E(\mathbf{u})$ which is the surprise or reward that guides the learning. The cost function of EGHR is defined as $\mathcal{L} = c\frac{1}{2}\langle(E_0 - E(\mathbf{u}))^2\rangle$. It was proven mathematically and numerically that this learning rule is robust, stable and its equilibrium point is proportional to the inverse of the mixture matrix, *i.e.* the solution of ICA. However, there are some conditions that should be satisfied; 1) $g(u_i)$ is a monotonically increasing odd function, and 2) $E(\mathbf{u})$ is a convex function. In order to satisfy these conditions, $g(u_i)$ is chosen to be either a hypertangent function or hard hypertangent function and $E(\mathbf{u})$ is chosen as modulus function.

ICA assumes that the sources are linearly mixed using a mixture matrix $\mathbf{A}$. The final weight matrix, $\mathbf{w}$, should converge to $c\mathbf{A}^{-1}$ which is still a valid solution since c is a scaling factor. If the sources have known distributions, the optimal E_0 can be calculated as discussed in [163]. On the other hand, if there is no knowledge about source distributions, Isomura and Toyoizumi proved that for any $E_0 > 0$, there is a positive c that gives the equilibrium point of EGHR regardless of the input source nature (unknown distributions) since the relation between E_0 and c is a monotonically increasing function [163]. Thus, E_0 can be used as a tuning knob to the learning rule to achieve the solution.

As previously discussed, the weights are enclosed between $[-\frac{DR}{2}, \frac{DR}{2}]$, thus E_0 must be chosen to keep the weights in this range. With unknown sources, one can iterate over E_0 until a

solution is obtained. We start with an initial point (i.e. $E_o = 1.$) then keep decreasing it until the expectation of $E(\mathbf{u})$ becomes constant.

Different test cases with linear update for nonlinear devices were performed. We found that the algorithm does not converge to the independent components. The nonlinear asymmetric derived expressions are needed to make the RRAMs behave like the standard EGHR ICA weight update to guarantee convergence. Given below, is an example of the simulated cases without variations. Clearly, the weights converged to negative constant values within the RRAM range.

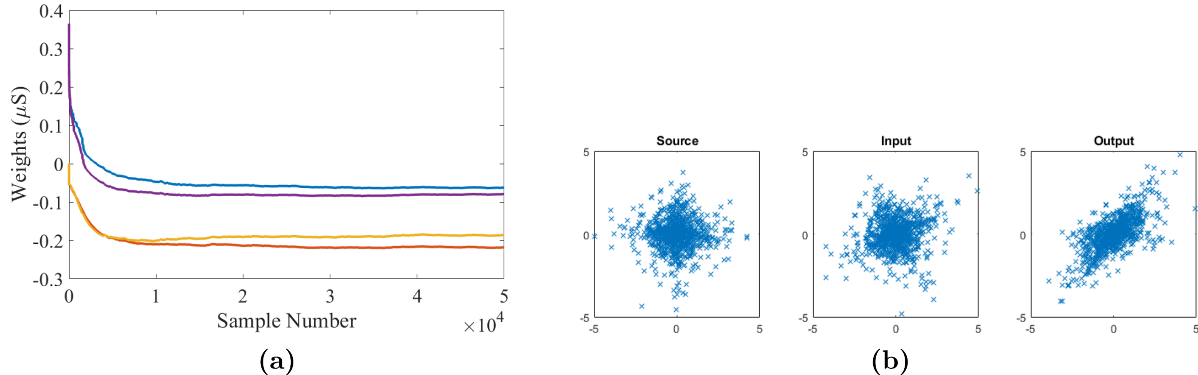


Figure 6.3: (a) Evolution of the weights, and (b) Visual results of the input and the output.

To have the resistive devices behave as EGHR to overcome the asymmetric nonlinearity, the change in each weight must be proportional to the change in the RRAM's conductance, $\Delta\mathbf{G} \propto \Delta\mathbf{w}$. To achieve this, the asymmetric nonlinear behavior of potentiation and depression should be included in developing the learning rule. We first calculate the change in the weights for both potentiation and depression cases taking into effect the asymmetric nonlinearity of the RRAM model. In general, the change in the LTP's conductance due to applying Δn is

$$\Delta G_{LTP} = G(n + \Delta n) - G(n) = (G_{max} - G(n)) \left(1 - e^{-\alpha_P \Delta n}\right) \tag{6.4}$$

where $G(n)$ is the previous conductance. Similarly, the change in the LTD conductance due

to applying Δn is $\Delta G_{LTD} = (G(n) - G_{min})(e^{-\alpha_D \Delta n} - 1)$. Clearly, the relation between the rate of change in conductance and Δn is an injective function. Thus, the number of pulses to cause ΔG_{LTP} and ΔG_{LTD} are

$$\Delta n_{LTP} = -\frac{1}{\alpha_P} \ln \left(1 - \frac{\Delta G_{LTP}}{G_{max} - G(n)}\right), \text{ and} \tag{6.5}$$

$$\Delta n_{LTD} = -\frac{1}{\alpha_D} \ln \left(\frac{\Delta G_{LTD}}{G(n) - G_{min}} + 1\right), \tag{6.6}$$

respectively. After learning, $\Delta \mathbf{G}$ goes to $\mathbf{0}$. As a result, $\Delta \mathbf{n}$ goes to zero as well. Equations (6.17) and (6.18) are nonlinear functions which are hardware expensive to implement. Thus, both can be linearized as $\ln(1 - x) \approx -x(1 + 0.5x) \approx -x$ and $\ln(1 + x) \approx x(1 - 0.5x) \approx x$ for $x \ll 1$. As previously discussed, we can replace $\Delta G_{i,j}$ by $\eta' \Delta w_{i,j}$, where η' is the scaled learning rate, and $\Delta w_{i,j}$ is given by EGHR. Thus, the final equations for potentiation and depression pulses can be written as follows:

$$\Delta n_{i,j}|_{LTP} \approx \frac{1}{\alpha_P} \left(\frac{\eta'(E_o - E(\mathbf{u}))g(u_j)x_i}{G_{max} - G_{i,j}(n)}\right), \text{ and} \tag{6.7}$$

$$\Delta n_{i,j}|_{LTD} = -\frac{1}{\alpha_D} \left(\frac{\eta'(E_o - E(\mathbf{u}))g(u_j)x_i}{G_{i,j}(n) - G_{min}}\right), \tag{6.8}$$

respectively.

By programming the RRAMs using the previous equations, the circuit behaves as required and compensates for the asymmetric nonlinearity of the devices. The proposed training algorithm is shown in Algorithm 1. We chose to initialize the weights with identity because we assume initially that the inputs are the independent components themselves and unmixed which means u=x, thus W=I. The inference part is represented in lines 5-7 however lines 8-9 represent the learning part. Line 5 in the algorithm is implemented using crossbar array

shown in Fig.6.2. Lines 6-7 can be implemented using either CMOS circuitry or off the shelf components such as operational amplifiers similar to those discussed in [61]. The training part, can be implemented either using analog or digital circuits.

Algorithm 6.1 Proposed Training Algorithm.

```
1: Set E_o = 1
2: while ⟨E(u)⟩ become constant do
3:     Initialize RRAMs' weights to Identity
4:     for x ∈ the training set do
5:         u = (G − G_r)x
6:         g = hardtanh(bu)
7:         E = Σ_i |g(u_i)|
8:         ΔW = η(E_o − E)gx^T
9:         ΔP=round(f(ΔW))
10:    end for
11:    Decrease E_o
12: end while
```

6.1.2 De-mixing Example and Results

As a test bench for the proposed technique, we consider two Laplacian random variables that are generated and mixed using a mixture matrix which is set to a rotation matrix $\mathbf{A} = (\cos\theta, -\sin\theta; \sin\theta, \cos\theta)$ with $\theta = \pi/6$. The circuit has been implemented as shown in Fig.6.2. The system weights are trained using 5×10^6 samples with 5×10^{-9} learning rate and hard *tanh* activation function (g) with $b = 2 \times 10^6\Omega$. On the other hand, RRAMs are programmed using $1\mu s$ pulses with $\pm 3V$ using the aforementioned conductance parameters. The variability of model parameters are considered with different Gaussian distribution for each device to consider device to device variability as well using the aforementioned parameters.

Figure 6.4 shows the results of the online learning of independent components of the mixed signals. Figure 6.4a shows the weights evolution during the training. The weights $\mathbf{W} = \mathbf{G} - Gr$ are initialized by the identity matrix (no knowledge about the mixture matrix) where

the conductance matrix is $G = [G_{max}, -G_r; G_r, G_{max}]$. After learning, the weight matrix is $W = [0.2034, 0.1133; -0.11, 0.17]\mu S$ and $WA = 0.233 \times [1, -0.0155; -0.043, 0.8733]\mu S$ which is approaching identity. Clearly, there are some oscillations in the weights around the final solution after 10^4 samples because of the continuous on line learning and the devices variations. Figure 6.4c shows the evolution of pulses for programming each weight and the evolution of the global surprise signal E is shown in Fig.6.4b. A visual representation of the signals before mixing, after mixing and after training is shown in Fig. 6.4d which depicts the similarity between the source and output signals.

Endurance and retention of RRAMs are very important measures especially for online learning case. It is required to guarantee that the algorithm converges after a number of update cycles much smaller than the endurance of the used device. The recent fabricated devices have good endurance and retention values typically around 10^8 and 4 years, respectively [68]. The total number of programming pulses for each weight are $\{2.96, 1.4, 0.64, 2.73\}$ million pulses which are less than the endurance. One way to decrease the number of programming pulses is by updating the weights using batch-based updates which will decrease the variations in the weights as well.

6.1.3 Power, area and speed comparison with CMOS realization

RRAM crossbar array offers performing the matrix vector multiplication in only one step compared to N^2 steps for other realizations. Thus, in terms of the speed, it highly accelerates the inference part N^2 times. The area of RRAM device in the crossbar is $4F^2$, where F is the feature size which is expected to be around $5nm$. Thus, the total area of the entire crossbar array (matrix vector multiplication engine) is around $4N^2F^2$. On the other hand, the transistor-based implementation requires registers, and MAC operation per cell which occupies a massive area in the chip. The registers, and MAC operation consumes a lot of power compared to one RRAM. Thus, crossbar based neuromorphic hardware has a

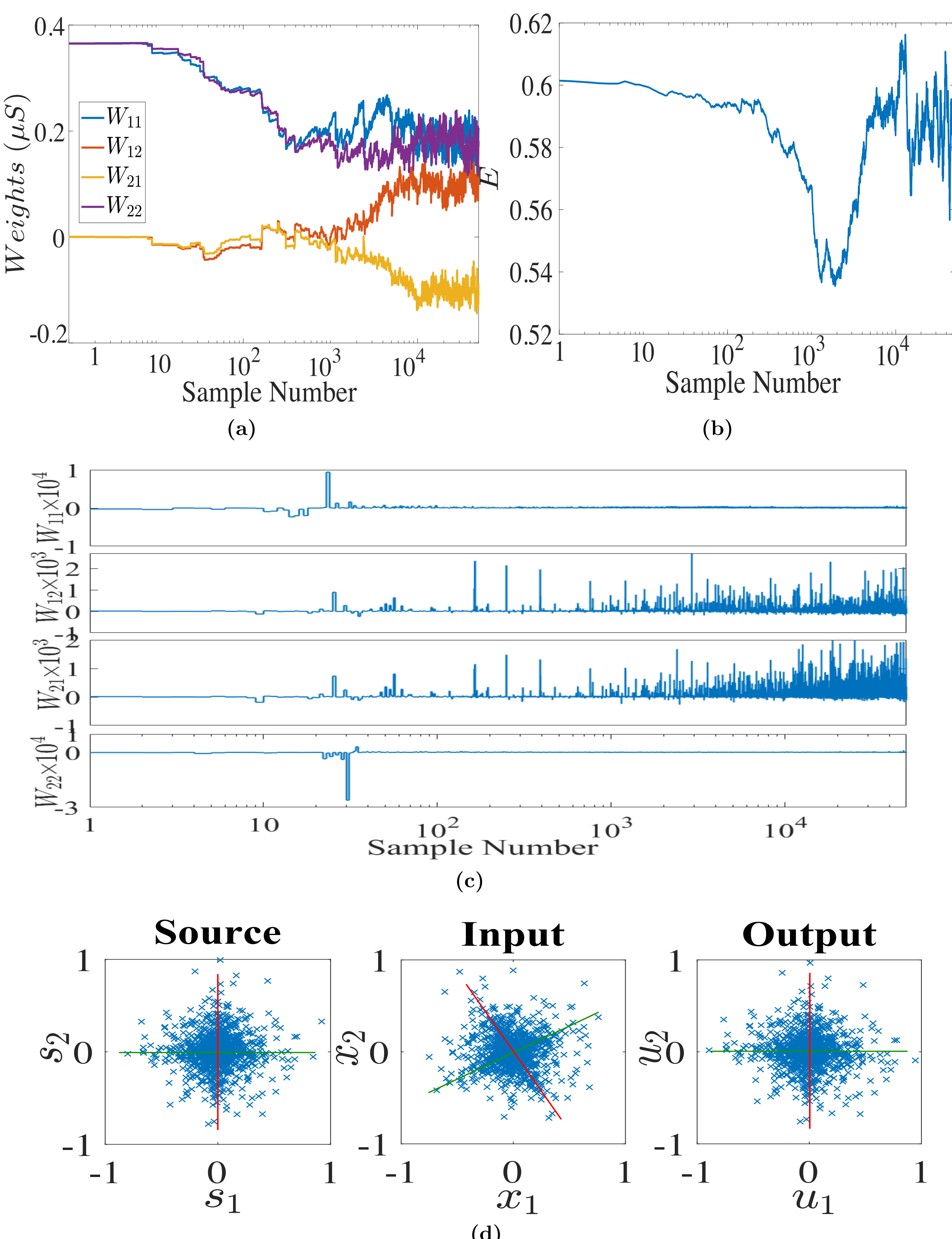


Figure 6.4: The online training results versus training time (a) Evolution of the weights, (b) surprise Energy function, (c) Required programming pulses for each weight, and (d) Visual results of the input and the output.

significant performance improvement. Consider the following numerical example for matrix-vector multiplication of two Laplacian sources. The transistor-based ICA is implemented using UMC 130nm (32-bit fixed point registers (16 bits for the integer part and 16 bits for the floating part)). The following tabulated results are post layout. To have fair comparison, these results are scaled to 5nm to be comparable to expected RRAM feature size ($F = 5nm$).

Table 6.1: Comparison between RRAM- and transistor-based ICA realization

	RRAM-based	Transistor-based	
		UMC 130nm	Expected $F = 5nm$
Area	$400nm^2$	$5333.76\mu m^2$	$7.89\mu m^2$
Power	$(0.12 - 2.8)\mu W$	$0.432mW$	$0.64\mu W$
Delay	$0.1ns$	$3.684ns$	$0.1416ns$

6.2 Deep Neural Network with Local Learning

6.2.1 Neural Network Model

The SNN model consists of spiking integrate-and-fire neurons without temperal dynamics to have a sparse and simple communications between the SNN layers to overcome the need for analog to digital and digital to analog converters. The discrete-time model can be formulated as

$$U_i^l[n] = \sum_j W_{ij}^l S_j^{l-1}[n] \tag{6.9a}$$

$$S_i^l[n] = \Theta(U_i^l[n]) \tag{6.9b}$$

where j and i are the indices of the input and the output signals, respectively, $U_i^l[n]$ is the membrane potential of the neuron i at layer l at time step n, W^l is the synaptic weight

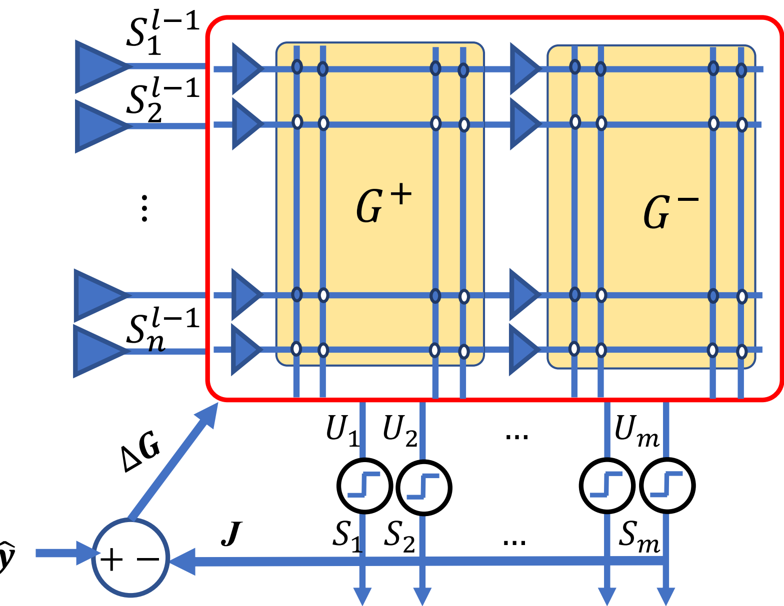


Figure 6.5: Single memristive spiking neural network layer with online local learning capabilities.

matrix between layer $l-1$ and l, and S_i^l is the binary output of this neuron i. The step function Θ is the spiking mechanism, *i.e.* $S_i^l[n] = 1$ for $U_i^l[n] > 0$. It is worth to highlight that in this SNN, the reset mechanism is omitted for simple implementation without loss in the performance [154]. Thus, it can be seen as spiking version of the artificial neural networks.

6.2.2 Local Learning with Crossbar Arrays

RRAMs are used to implement the synaptic weights. In practice, two RRAMs are needed to realize each weight to have positive and negative values. In this work, we use a balanced realization where two RRAMs are used for each weight such that $w = G^+ - G^-$. If the G^+ is greater/less than G^-, it represents positive/negative weight, respectively. Thus, the memristive SNN, shown in Fig. 6.5, can be written as

$$U_i^l[n] = \sum_j \left(G_{ij}^{+l} - G_{ij}^{-l}\right) S_j^{l-1}[n] \tag{6.10}$$

Assuming a local cost function $\mathcal{L}$ for each layer, the gradients with respect to each device's conductance are formulated as three factors

$$\frac{\partial \mathcal{L}}{\partial G_{ij}^{\pm l}} = \frac{\partial \mathcal{L}}{\partial S_i^l} \frac{\partial S_i^l}{\partial U_i^l} \frac{\partial U_i^l}{\partial G_{ij}^{\pm l}} \tag{6.11}$$

The term $\partial U_i^l / \partial G_{ij}^{\pm l}$ is equal to $\pm S_j^l[n]$ in the integrate and fire neuron. The middle factor, $\partial S_i^l / \partial U_i^l$ is the derivative of the step function Θ which is not differentiable. In practice, the step function is replaced by a piecewise linear function [154]. The derivative of the piecewise linear function is the box function $\partial S_i^l / \partial U_i^l = 1$ if $u_- < U_i < u_+$ and 0 otherwise. The term $\partial \mathcal{L} / \partial S_i^l$ describes the change in the spiking state affects in the loss is called the local error denoted as δ_i^l and computed by gradient backpropagation. The positive and negative conductances become

$$\Delta G_{ij}^{\pm l} = -\eta \frac{\partial \mathcal{L}}{\partial G_{ij}^{\pm l}} = \mp \eta \delta_i^l S_j^{l-1}, \text{ if } u_- < U_i < u_+, \tag{6.12}$$

where η is the learning rate.

In the local learning, the local losses are local classifiers (using same output labels) [164]. Local losses may outperform the regular backpropagation in some tasks [165]. The local classifier is a random linear transformation J that reduce the dimension of each layer output S^l to the number of classes. This comparison with class labels can be performed using either mean-squared error loss or cross-entropy or any other loss function. For example, mean-squared error loss is

$$\mathcal{L}^l = \frac{1}{C} \sum_{k=1}^{C} \mathcal{L}_k^l = \| \sum_{i=1}^{N^l} J_{ki}^l S_i^l - \hat{y}_k \|_2, \tag{6.13}$$

where C is the number of classes, N^l is the layer width and J_{ki}^l is an element in a random, fixed matrix, and $\hat{y}_k$ are one-hot encoded labels. Thus, $\delta_i^l = \sum_{k=1}^{C} J_{ik}^l err_k^l$ where $err_k^l =$

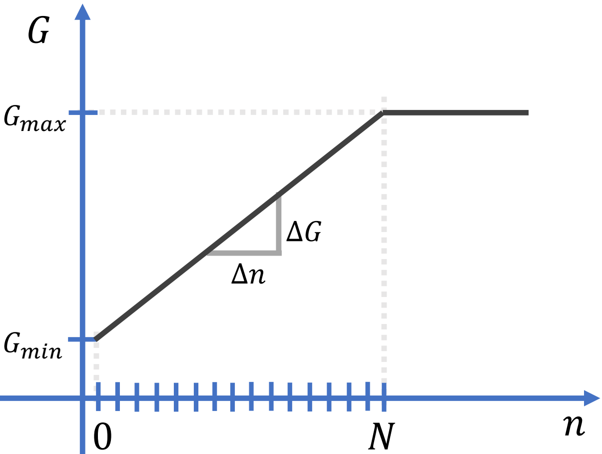


Figure 6.6: RRAM's conductance change relation with number of pulses .

$\sum_{i=1} J^l_{ki} S^l_i - \hat{y}_k$. For simplicity, it can be written in matrix form as $\boldsymbol{\delta}^l = \mathbf{J}^{l^T}(\mathbf{J}^l\mathbf{S}^l - \hat{\mathbf{y}})$.

The common practice to potentiate or depress the conductance value is through applying positive or negative voltage pulses, respectively. Ideally, the conductance change should have a linear relation with the update pulses, as follows:

$$G = G_{min} + \frac{G_{max} - G_{min}}{N} n, \quad G \in [G_{min}, G_{max}] \tag{6.14}$$

where G_{max} and G_{min} are the maximum and minimum achievable conductances, respectively, n is the pulse index, and N is the total number of pulses to fully potentiate or depress the device. According to this modeling equation, the required number of pulses to cause a certain change in conductance ΔG can be written as

$$\Delta n = \frac{N}{G_{max} - G_{min}} \Delta G \tag{6.15}$$

which should be rounded to the nearest integer. The previous equation can be rewritten in normalized form as $\Delta n = \frac{N}{1-K}\Delta w$ where $K = G_{min}/G_{max}$ and $\Delta w = \Delta G/G_{max}$.

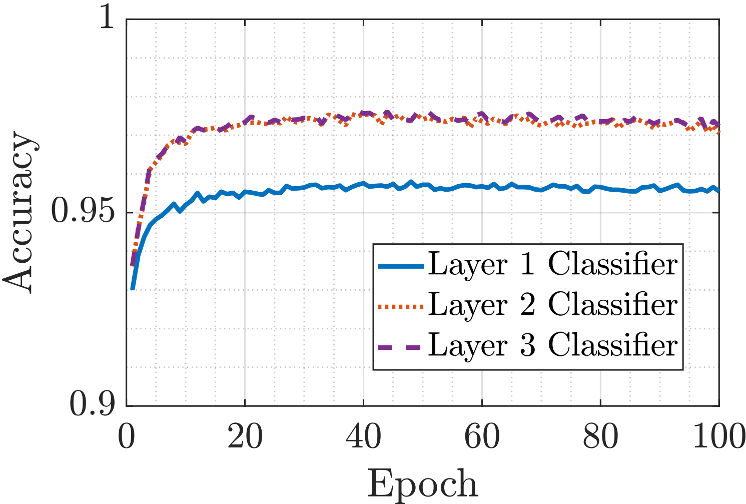


Figure 6.7: Ideal validation with limited weight range and with stochastic rounding at $K = 1/20$ and $N = 100$.

6.2.3 Experimental Setup

A three-layer fully connected spiking network is implemented using Pytorch with local learning using random local classifiers. The layer connections are as follows: 784 $\rightarrow$ 500 $\rightarrow$ 500 $\rightarrow$ 250. In this work, we used Adam optimizer with a cross-entropy loss function. The back-propagation calculation of each layer is calculated using autograd functionality [166].

Figure 6.7 shows the local test accuracy of each layer of a three-layer spiking neural network with limited weight value $w \in [K-1, 1-K]$ and with stochastically rounded updates. The stochastic rounding is common practice for limited numerical precision calculation where it is an unbiased rounding and has zero expected rounding error [167]. Without the stochastic rounding, the network fails to learn.

6.2.4 Online Training Under Asymmetric Nonlinear Updates

In section IV, we proposed a method to have the resistive devices behave exactly like the learning rule where the change in each weight must be proportional to the change in the

RRAM's conductance, $\Delta\mathbf{G} \propto \Delta\mathbf{w}$. To achieve this, the asymmetric nonlinear behavior of potentiation and depression are included in the learning rule. We first calculate the change in the weights for both potentiation and depression cases taking into effect the asymmetric nonlinearity of the RRAM model. In general, the change in the LTP's and LTD's conductance due to applying Δn is

$$\begin{aligned} \Delta G_{LTP} &= (G_{max} - G)\left(1 - e^{-\alpha_P \Delta n}\right), \text{ and} \\ \Delta G_{LTD} &= (G - G_{min})(e^{-\alpha_D \Delta n} - 1), \end{aligned} \tag{6.16}$$

respectively where G is the previous conductance. Clearly, the relation between the rate of change in conductance and Δn is an injective function. Thus, the number of pulses to cause ΔG_{LTP} and ΔG_{LTD} are

$$\Delta n_P = -\frac{1}{\alpha_P} \ln\left(1 - \frac{\Delta G_{LTP}}{G_{max} - G(n)}\right), \text{ and} \tag{6.17}$$

$$\Delta n_D = -\frac{1}{\alpha_D} \ln\left(\frac{\Delta G_{LTD}}{G(n) - G_{min}} + 1\right), \tag{6.18}$$

respectively. After learning, $\Delta\mathbf{G}$ goes to $\mathbf{0}$. As a result, $\Delta\mathbf{n}$ goes to zero as well.

The update equations ((6.17) and (6.18)) require the knowledge of the weight value, meaning a read operation is needed to calculate the required number of pulses to update. This nonlinear cancellation can be realized either in analog domain using logarithmic amplifier circuits or in digital domain using lookup tables. However, they are area and latency expensive.

One way to overcome the need for extra hardware for the nonlinearity cancellation is by linearizing the updated model as discussed in [121]. Both equations 6.17 and 6.18 can be linearity as $\ln(1-x) \approx -x(1+0.5x) \approx -x$ and $\ln(1+x) \approx x(1-0.5x) \approx x$ for $x \ll 1$. This

leads to that the potentiation and depression updates are given by

$$\Delta n_P = \frac{1}{\alpha_P}\frac{\Delta G_{LTP}}{G_{max}-G(n)},\ \Delta G_{LTP} \ll G_{max} - G(n),\ \text{and} \tag{6.19a}$$

$$\Delta n_D = \frac{1}{\alpha_D}\frac{|\Delta G_{LTD}|}{G(n)-G_{min}},\ |\Delta G_{LTD}| \ll G(n) - G_{min}, \tag{6.19b}$$

respectively. As previously discussed, it is needed to make $\mathbf{\Delta G} \propto \mathbf{\Delta w}$, these two conditions can be satisfied by using smaller learning rates. But, the weight read cycle is still needed even in this linearized model. The rounding of the integer number of update pulses does not guarantee the convergence to the optimal point since the rounding is done after nonlinearity cancellation and gradient calculations. The device dynamics should be included directly to the learning rule.

6.2.5 Stochastic Ternary Update Method

Another practical method to update the weight is that we use only directional updates where the update pulses are proportional or equal to $\Delta w = \Delta G/G_{max}$. Since ΔG is always less than G_{max}, then only positive or negative or zero update pulse is applied to each conductance which we can refer to as Ternary Update. This update method may cause slow learning. To have faster learning. scaling factors can be added to the update rule which can be generally written as

$$\Delta n_{P,D} = \gamma_{P,D} SR(\Delta w), \tag{6.20}$$

where $\gamma_{P,D}$ are scaling factors and SR is a stochastic rounding function. γ_P and γ_D can be equal or different based on the asymmetry in the potentiation and depression of the used device. This update model can be easily implemented without the need for any kind of

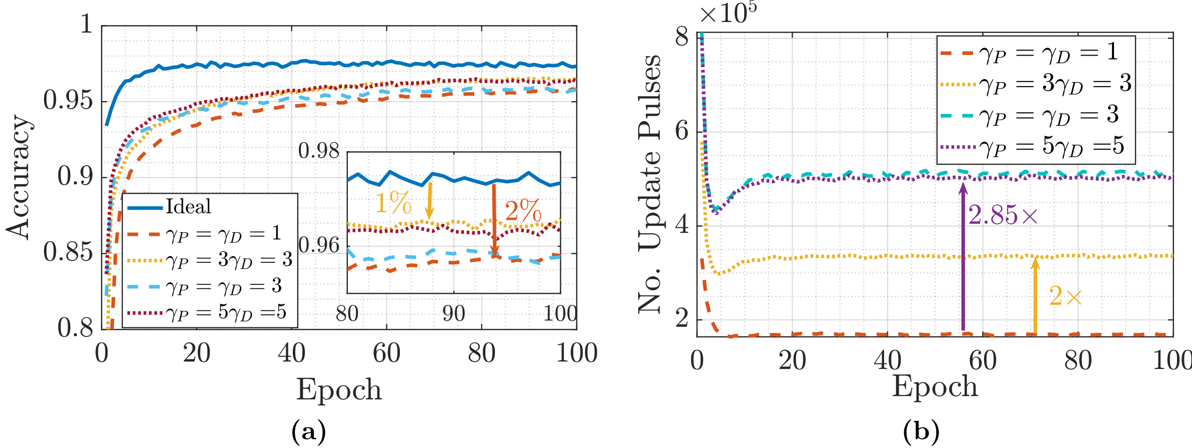


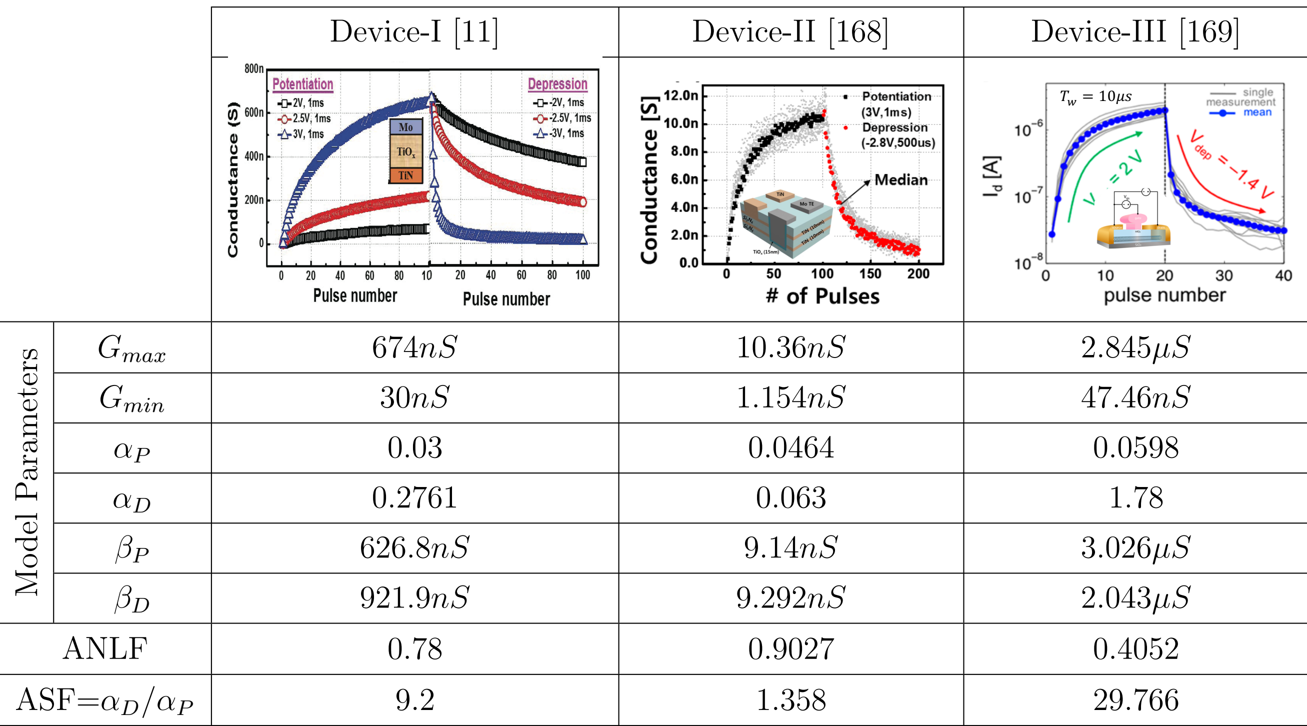
Figure 6.8: (a) Test accuracy of the outer classifier for different scaling factors of ternary update rule and (b) total number of positive and negative update pulses for each epoch.

nonlinearity cancellation (i.e. lookup tables) and read cycles.

6.2.6 Results and Comparsion

Figure 6.8a shows a comparison between the baseline performance and the proposed ternary update rule for different γ values. The results show that the network can achieve almost the baseline with a 2% drop for equal γ values. In order to accelerate the training, we considered higher γ values. It is clear that when γ increased $3\times$ or $5\times$ the training is speedup by the same ratio. We can see that we asymmetric γ values that the network is able to achieve an extra 1% in the accuracy compared to symmetric or equal γ values.

Figure 6.8b shows the number of update pulses performed in each epoch. At the beginning of the training, the number of update pulses is high and reduces during the progress of the training. The number of update pulses settles after 10 epochs where fine tuning of the network. Clearly, with increasing the scale factor, γ, the number of update pulses increases $2\times$ and $2.85\times$ for $\gamma_P + \gamma_P = 4$ and 6, respectively. Thus, the write energy is expected to increase with the same ratios. Clearly, the asymmetric updates is needed to achieve a better performance and faster learning.



		Device-I [11]	Device-II [168]	Device-III [169]
Model Parameters	G_{max}	$674nS$	$10.36nS$	$2.845\mu S$
	G_{min}	$30nS$	$1.154nS$	$47.46nS$
	α_P	0.03	0.0464	0.0598
	α_D	0.2761	0.063	1.78
	β_P	$626.8nS$	$9.14nS$	$3.026\mu S$
	β_D	$921.9nS$	$9.292nS$	$2.043\mu S$
ANLF		0.78	0.9027	0.4052
ASF=α_D/α_P		9.2	1.358	29.766

Table 6.2: Extracted model parameters of the three devices understudy. ANLF is the asymmeteric nonlinearity factor and ASF is the asymmetery factor,

Comparison with other devices

In order to study the effect of the asymmeteric nonlinearity on the performance, two other devices are modeled using the same model discussed before. The first device of those is TiO_x-based device which we refer to as Device II [168]. While the second device is aligned carbon nanotube (CNT) synaptic transistor which can be assembled in crossbar architecture to perform MVM [169]. We refer to this device as Device-III. Table 6.2 summarized the extracted model parameters of the three devices.

Figure 6.9(a) shows the MNIST test accuracy trained with the modeled devices with variations added to the parameters with sysmteric maginude update, $\gamma_P = \gamma_D = 1$. Clearly, Device-II has the slowest learning rate in contrast with Device-II that has the fastest learning rate. This happend because Device-II and -III have the highest and lowest ANL factors, respectively. As a conclusion, the higher ANL factor, the slower the learning. On the other

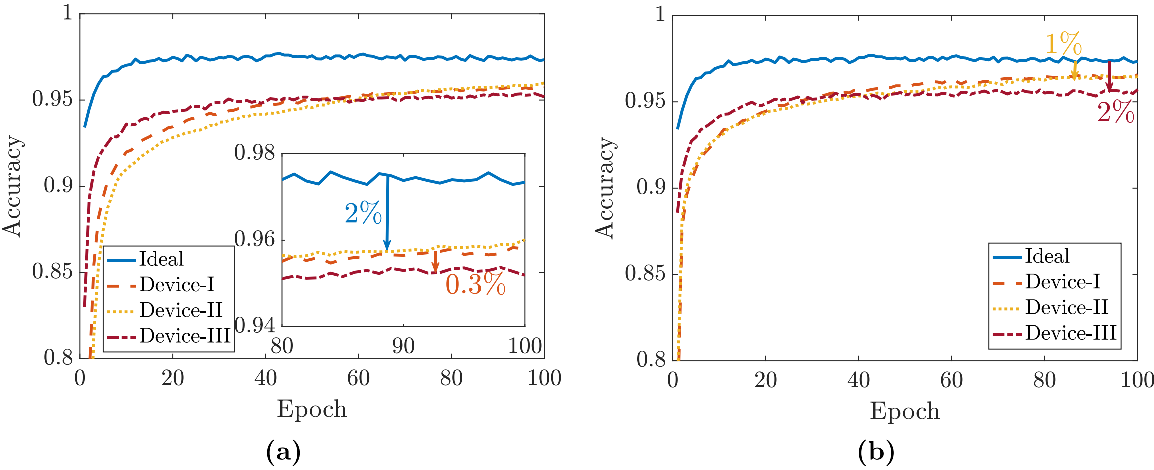


Figure 6.9: MNIST recognition accuracy for three different devices with a) $\gamma_P = \gamma_D = 1$ and b) $\gamma_P = 3\gamma_D = 3$.

hand, training with Device-II achieves as slightly higher train accuracy (extra 0.3%) compared to the other devices which happens since this device has less a asymmetry factor which is 1.3 compared to 9.2 and 29.76 in the other devices. While with $\gamma_P = 3\gamma_D = 3$, fig. 6.9(b) shows that the learning rate is improved by the double (i.e the same accuracy is achieved with half the number of epochs.) and the recognition accuracy is improved by 1% for device-I and Device-II. However, Device-III is improved by 0.3% only due to the high asymmetry in that device.

6.2.7 RRAM's Energy Update Model

The instantaneous power consumed while potentiating and depressing the conductance is $p_{P,D}(t) = G(t)V_{P,D}^2$, and the write energy is the time integral of the instantaneous power which can be written as

$$E_{w_{P,D}}(t) = \int_0^t G(\tau)V_{P,D}^2 d\tau \tag{6.21}$$

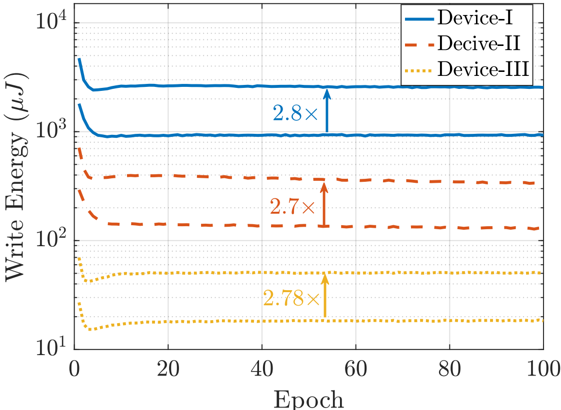


Figure 6.10: Total Write Energy for each epoch for $\gamma_P = \gamma_D = 1$ and $\gamma_P = 3\gamma_D = 3$

Since, the potentiation and depression are done through pulses as previously discussed then the write energy is given by

$$E_{w_{P,D}}(n) = \int_0^{\Delta n} G(n) V_{P,D}^2 T dn \tag{6.22}$$

The potentiation and depression update of the conductance ($G(n) = G_o + \Delta G(n)$) can be written as follows

$$G_{P,D}(n) = G_{max,min} \left(1 - e^{-\alpha_{P,D} n}\right) + G_o e^{-\alpha_{P,D} n} \tag{6.23}$$

Thus, the potentiation and depression write energy can be written as

$$E_{w_{P,D}} = V_{P,D}^2 T \left[G_{max,min} \Delta n + \left(\frac{G - G_{max,min}}{\alpha_{P,D}} \right) \left(1 - e^{-\alpha_{P,D} \Delta n}\right) \right] \tag{6.24}$$

Figure 6.10 shows the the total write energy while the training. Clearly, it decreases then saturates. Device III has the less the write energy while Device-I has the highest write energy around 700× of the Device-I. The total write energy is 2.7× higher with using 3× asymmetric weight pulses.

6.2.8 Limited Endurance

Endurance is a critical obstacle to RRAM deployment in neuromorphic hardware with online learning capabilities where the devices are frequently updated, and especially so during gradient-based learning [66, 68]. With limited endurance, it is necessary to complete the training before the devices degrade. In standard deep learning, weight updates are usually performed every batch. The bigger the batch size, the less number of updates. However, it would require a bigger memory to store the error signals and corresponding input for each input.

6.2.9 Conductance Degradation Model

In [151], the authors reported a conductance degradation in terms of reducing conductance ratio, G_{max}/G_{min}. Based on this behavior, a mathematical model can be developed to model the degradation in the conductance ratio as follows:

$$G_{max}(n_n) = G_{max} - (G_{max} - G_{min}) \left(\frac{e^{vn_n} - 1}{e^{v} - 1} \right) \tag{6.25}$$

where v and n_n are the drift coefficient and normalized programming pulses, normalized to the endurance value.

Figure 6.12 shows the test accuracy for different batch sizes. The smaller the batch size, the higher accuracy degradation while the training is proceeding. Besides, the final test accuracy is degraded extra 4% due to the endurance behavior. Larger batch sizes are required however it would require extra memory hardware to store the batch data for both inputs and local outputs. In order to overcome the endurance problem, more efficient training techniques are needed which we discuss in the next section.

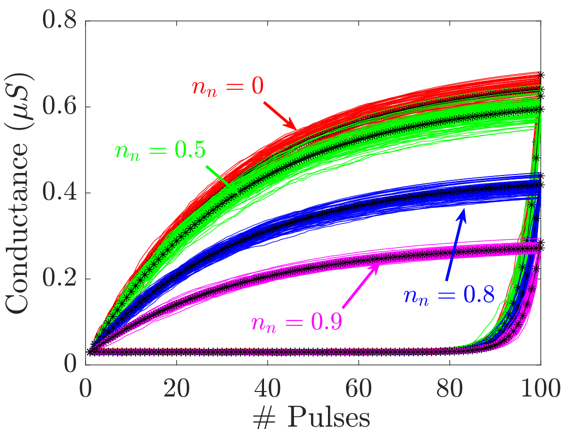


Figure 6.11: The potentiation and depression conductance degradation for different normalized programming pulses.

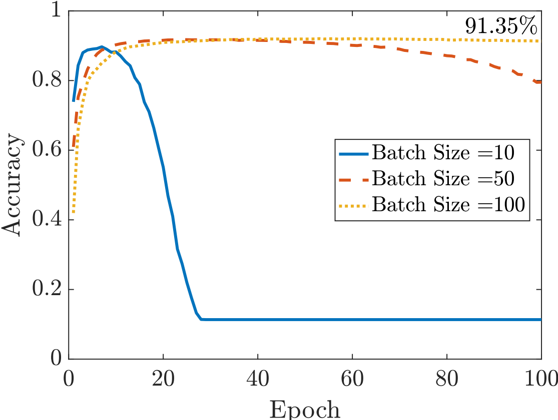


Figure 6.12: MNIST test accuracy for different batch sizes with 10^6 endurance value.

6.3 Error-triggered Learning

For most interesting cost functions, errors must be computed extrinsically and communicated to the neuron. To make this communication efficient, we encode errors using positive and negative events as follows:

$$E_i^l = \begin{cases} 1 & \text{if } err_i^l > \theta, \\ -1 & \text{if } err_i^l < -\theta, \\ 0 & \text{otherwise} \end{cases} \tag{6.26}$$

where $\theta \in \mathbb{R}$ is a constant or slowly varying error threshold. Using this encoding, the parameter update rule becomes:

$$\Delta W_{ij}^l = -\theta E_i^l P_j^l B(U_i^l) \tag{6.27}$$

where θ is called the stop-learning threshold (η was folded into θ). Thus, an update takes place on an error of magnitude θ and if $B(U_i^l) = 1$. The sign of the weight update is $-E_i^l$ and its magnitude θP_j^l. Provided that the layer-wide update magnitude can be modulated proportionally to θ, this learning rule implies two comparisons and an addition (subtraction).

6.3.1 Local Losses and Local Errors

Up to now, we have side-stepped the calculation of $err[n]_i^l$. If l is not the output layer, then computing this term requires solving a deep credit assignment problem. Gradient BP can solve this, but is not compatible with a physical implementation of the neural network [170]. Several approximations have emerged recently to solve this, such as feedback alignment [152, 153, 171], and local losses defined for each layer [164, 7, 165]. For classification, local

losses can be local classifiers (using output labels) [164], and supervised clustering, which perform on par and sometimes better than BP in classical ML benchmark tasks [165]. For all experiments used in this work, we use a layer-wise local classifier using a mean-squared error loss defined as $\mathcal{L}_i^l = ||\sum_{k=1}^{C}(J_{ik}^l S_k^l - \hat{y}_k)||_2$, where J_{ik}^l is a random, fixed matrix, $\hat{y}_k$ are one-hot encoded labels, and C is the number of classes. The gradients of $\mathcal{L}_i^l$ involve backpropagation within the time step n and thus requires the symmetric transpose, $J^{l,T}$. If this symmetric transpose is available, then $\mathcal{L}$ can be optimized directly. To account for the case where J^T is unavailable, for example in mixed signal systems, we train through feedback alignment using another random matrix H^l [152] whose elements are equal to $H_{ij}^l = J_{ij}^{l,T}\omega_{ij}^l$ with Gaussian distributed $\omega_{ij}^l \sim N(1, \frac{1}{2})$, where T indicates transpose. Weight updates are achieved through stochastic gradient descent (SGD). We note that an error can be computed with any loss function (*e.g.* mean-squared error or cross entropy) provided there is no temporal dependency, i.e. $\mathcal{L}[n]$ does not depend on variables in time step $n-1$. If such temporal dependencies exist, for example with Van Rossum distance [172], the learning rule becomes considerably more complex and (6.12). The matrices J^l and H^l can be very large, especially in the case of convolutional networks. One solution to the memory footprint of J^l is to generate these matrices on the fly using a random number generator [164]. Another solution is to define J^l as a sparse, binary matrix [170]. Using a binary matrix would further reduce the need for multiplications in the computation of err_i.

6.3.2 Large-scale Simulation Experiments

An important feature of the used learning rule is its scalability to multilayer networks with very small loss of performance compared to a standard deep neural network when using idealized dynamics. To demonstrate this, we simulate the learning dynamics for classification in large-scale, multilayer spiking networks. The GPU simulations focus on event-based datasets acquired using a neuromorphic sensor, namely the N-MNIST and DVS Gestures dataset for

Table 6.3: Recognition Error in Idealized Spiking Neural Network Simulations Averaged over 5 Runs.

	DVSGesture		N-MNIST	
$\langle E \rangle$	Error	Writes	Error	Writes
Cont.	3.82%	1M	2.3%	1.5M
50Hz	4.22%	50k	2.31%	75k
10Hz	6.25%	10k	2.71%	45k

demonstrating the learning model. Both datasets were pre-processed as in [7]. The N-MNIST network is fully connected (1000–1000–1000), while the DVS Gestures network is convolutional (64c7-128c7-128c7). For practical reasons, the neural networks were trained in minibatches of 72 (DVS Gestures) and 100 (N-MNIST). We note that the choice of using minibatches is advantageous when using GPUs to simulate the dynamics and is not specific to (6.12), and can also be used for gradient backpropagation through time and spike-timing dependent plastisity. The parameters of our model are similar to that of [7] except that the time constants were randomized. In our experiments, we used a proportional controller to adjust θ such as the average error spike rate $\langle E \rangle$ remains stable. The column writes indicates an upper bound on the number of weight writes. It is an approximate upper boundary, as the effect of $B(U)$ has not been taken into account. These results in Table 6.3 demonstrate a small loss in accuracy across the two tasks when updates are error-triggered. As comparison, published work on DVS Gestures with spiking neurons trained with backpropagation achieved 5.41% [173] and 6.36% [174] error rates and 1.3% [175].

6.4 Conclusion

The realization of ICA using RRAMs is introduced taking into consideration the asymmetric nonlinearity behavior of the devices and the variations. The closed form learning rule is introduced and is applied to de-mixing two Laplacian signals. The proposed algorithm showed good performance and converges to the independent components even with the existence of the device variability.

we proposed an effective method to train a memristive spiking neural network with local learning containing RRAMs with asymmetric nonlinearity update dynamics. Stochastic rounding is necessary to have a successful training. we showed with the experiments that the ternary update method was very efficient and almost achieve baseline accuracy. Other nonidealites such as IR drop (i.e sneak path problem), limited endurance will be investigated in the future work.

Chapter 7

Prospects on Memristive Neuromorphic Hardware

The implementation of learning dynamics as synaptic plasticity in neuromorphic hardware can lead to highly efficient, lifelong learning systems [21, 176, 177, 23, 149]. While gradient Backpropagation (BP) is the workhorse for training nearly all deep neural network architectures, it is incompatible with neuromorphic hardware because it is not spatially and temporally local [170]. Recent work addresses this problem using Surrogate Gradient (SG) learning [154]. SGs use a differentiable surrogate network to compute weight updates in a local fashion, and formulate the updates as three-factor synaptic plasticity rules. The SG approach reveals from first principles the mathematical nature of the three factors, and a learning dynamic that is continuous in time. While temporal continuity is a plausible property in the brain, while being able to perform a large number of weight updates (writes) which can be energetically expensive in hardware [176].

7.1 Synaptic Plasticity and Learning in SNN

As RRAM arrays provide a scalable physical substrate for implementing neural computations and plasticity, we now turn to the modeling of synaptic plasticity. Synaptic plasticity in the

brain is realized using some constraints as in RRAMs. One of these constraints is that information necessary for performing efficient weight updates must available at the physical location where the weights updates are computed and stored.

The brain is capable of learning and adapting at different time scales. Generally, learning processes operate in the hours to years range, which is thought to be implemented by synaptic plasticity and structural plasticity. Focusing on the former, a common synaptic plasticity process dictates that synaptic weights changes according to a modulated Hebbian-like process [178], which can be written in a functional form as:

$$\Delta W_{ij} = f(W_{ij}, S_i, S_j, M_i)$$

where M_i is some modulating function that is not yet specified. A common, biologically inspired model is fSTDP. The classical STDP rule modifies the synaptic strengths of connected pre- and post-synaptic neurons based on the spike history in the following way: if a post–synaptic neuron generates action potential within a time interval after the pre-synaptic neuron has fired multiple spikes then the synaptic strength between these two neurons becomes stronger (causal updates, long-term potentiation–LTP). Note that STDP does not use the modulation term. Formally:

$$\Delta W_{ij}^{STDP} \propto S_i(t)(\epsilon_{pre} * S_j(t)) - S_j(t)(\epsilon_{post} * S_i(t)) \tag{7.1}$$

where ϵ_{post} and ϵ_{pre} are two kernels, generally of first or second order (exponential or double exponential) filters as they relate to the neuron dynamics. The convolution terms $\epsilon * S(t) = \int \mathrm{d}s \epsilon(s) S(t-s)$ capture the trace of the spiking activity, and serve as key building blocks for synaptic plasticity. These terms are key for learning in SNN as they provide eligibility traces or memory of the neural activity history. These traces emerge from the gradients on the neural membrane potential dynamics [172].

STDP captures the change in the postsynaptic potential amplitude in an experimental setting [179] where the pair of neurons is elicited to fire at precise times. As such, it only captures a particular temporal aspect of the synaptic plasticity dynamics. Experimental work argues that STDP alone does not account for several observations in synaptic plasticity [180]. Theoretical work suggested that synapses require complex internal dynamics on different timescales to achieve extensive memory capacity [181]. Furthermore, error-driven learning rules derived from first principles are not directly compatible with pair-wise STDP [182]. These observations are not in contradiction with seminal work of [179], as considerable variation in LTP and LTD is indeed observed. Instead, [182] suggests that STDP is an incomplete description of synaptic plasticity.

On the flip-side, normative approaches derive synaptic plasticity dynamics from mathematical principles. While several normative approaches exist, in the following we focus on three-factor rules that are particularly well-suited for neuromorphic applications.

7.1.1 Gradient-based Learning in SNN and Three-Factor Rules

Three-factor rules can be viewed as extensions of Hebbian learning and STDP, and are derived from a normative approach [183]. The first two factors are functions of pre-synaptic activity and post-synaptic activity, and the third factor is a modulating function that is relevant to the learning task. Such rules have been shown to be compatible with a wide number of unsupervised, supervised, and reinforcement learning paradigms [183], and implementations can have scaling properties comparable to that of STDP [184].

Three-factor rules can be derived from gradient descent on the spiking neural network [182, 154]. Such rules are often "local" in the sense that all the information necessary for computing the gradient is available at the post-synaptic neuron [176]. Recent digital implementations of learning use three-factor rules, where the third factor is a modulation

term that depends on an internal synaptic state [21] or postsynaptic neuron state [184].

Three-factor rules are motivated by biology, where additional extrinsic factors that modulate the learning, for example, Dopamine, Acetylcholine, or Noradrenaline in reward-based learning [185], or GABA neuromodulator controlling STDP [186]. The three-factor learning rule can be written as follows:

$$\Delta W_{ij}^{3F} \propto f_{pre}(S_j(t)) f_{post}(S_i(t)) M_i \tag{7.2}$$

where f_{pre} and f_{post} correspond to functions over presynaptic and post synaptic variables, respectively, and M_i is the modulating term of postsynaptic neuron i. The modulating term is a task-dependent function, which can represent error, surprise, or reward.

The equivalence of SNN with artificial neural networks paired with synaptic plasticity derived from gradient descent suggests that the same approaches used for training artificial networks can be applied to SNN. In other words, the synaptic plasticity rule can be formulated in a way that it optimizes a task-relevant objective function [176]. A machine learning description of SNN training consists of three parts: The objective function, the (neural network) model and the optimization strategy. The objective, noted $\mathcal{L}(\mathbf{S}(\Omega), \mathbf{S_{data}})$ is a scalar function describing the performance of the task at hand (e.g. classification error, reconstruction error, free energy, etc.), where Ω are trainable parameters and $\mathbf{S}$,$\mathbf{S_{data}}$ are neural states (spikes, membrane potentials, etc.) and input spikes, respectively dictated by the SNN dynamics. If operating in a firing rate mode (where spike counts or mean firing rates are carriers of task-level information), $\mathbf{S}$, and $\mathbf{S_{data}}$ can be interpreted as firing rates instead. The optimization strategy consists of a parameter update derived from gradient descent on $\mathcal{L}$. If this update rule can be expressed in terms of variables that are local to the connection, then the learning rule will be called a synaptic plasticity rule.

Gradient-based approaches have been used in a wide range of work. For examples, the Tem-

potron is a learning rule using a membrane potential-based objective function to learn to discriminate between inputs on the basis of the spike train statistics [187]; [182] expresses the neuron model as a stochastic generative model and derive learning rules by maximizing the likelihood of a target spike train. Gradient-based approaches identify (non-unique) relationships between the STDP parameters and those of the neural and synaptic dynamics; and SpikeProp [188], a spike-based gradient backpropagation algorithm. Several other approaches that can collectively be described as surrogate gradient descent [154] rely on approximations of the gradient to perform SNN parameter updates [189, 174, 190, 172].

While the above models are computationally promising, there are important challenges in learning multilayer models on a physical substrate such as the brain: The physical substrate defines which variables are available to which processes and when. This is in stark contrast to von Neumann computers where learning processes have access to shared memory. One consequence of this limitation is the weight transport problem of gradient backpropagation, where the symmetric transpose of the weights is necessary to train deep networks. In many cases, however, the neurons and weight tables cannot be "reversed" in this fashion. Another less studied consequence is the temporal locality due to the continual dynamics of SNN: solving the credit assignment problem in recurrent neural networks requires some memory of the previous states and inputs, either in the form of buffers or eligibility traces [191]. Both of these problems, namely the weight transport problem and temporal credit assignment, must be solved in order to successfully implement memristor-based machine inference and learning. Below, we describe some promising approaches that overcome these two problems.

Feedback Alignment and Event-driven RBP One way to alleviate the weight transport problem is to replace the weights used in backpropagation with fixed, random ones [152]. Theoretical work suggests that the network adjusts its feed-forward weights such that they align with the (random) feedback weights, which is arguably equally good in commu-

nicating gradients. This approach is naturally extended to SNN to overcome the weight transport problem. Event-driven Random Back Propagation (eRBP) is one such rule that extends feedback alignment to meet the other constraints of learning in SNN, namely that weight updates are event-based (no separate forward and backward phases) and errors are maintained on a dedicated compartment of the neuron, rather than in a globally accessible buffer. Because it is local, it can be implemented as a presynaptic spike-driven plasticity rule modulated by top-down errors and gated by the state of the postsynaptic neuron and is simple to implement. The learning rule can be written as:

$$\begin{aligned} M_i &= \sum_k g_{ik} Error_k \\ \Delta W_{ij}^{eRBP} &\propto M_i Boxcar(U_i) S_j(t) \end{aligned} \tag{7.3}$$

where g_{ik} are fixed, random weights and $Boxcar$ is a symmetric function that is equal to 1 in the vicinity of $U_i = 0$, and zero otherwise. Here, M_i represents the state of the neural compartment that modulates the plasticity rule according to top-down errors. Its functionality is to maintain a trace of $Error_k = target_k - S_k$ when an input spike occurs. ERBP solves the nonlocality problems, leading to remarkably good performance on MNIST handwritten digit recognition tasks, achieving close to 2% error compared to 1.9% error using gradient backpropagation on the same network architecture.

Figure 7.1 shows a network that consists of feed-forward layers (784-200-200-10) for prediction and feedback layers for supervised training with labels (targets) $\mathcal{L}$. Full arrows indicate synaptic connections, thick full arrows indicate plastic synapses, and dashed arrows indicate synaptic plasticity modulation. Neurons in the network indicated by black circles were implemented as two-compartment LIF neurons. The top-layer errors are proportional to the difference between labels (L) and predictions (P) and is implemented using a pair of neurons coding for positive error (blue) and negative error (red). Each hidden neuron receives inputs from a random combination of the pair of error neurons. Output neurons receive inputs from

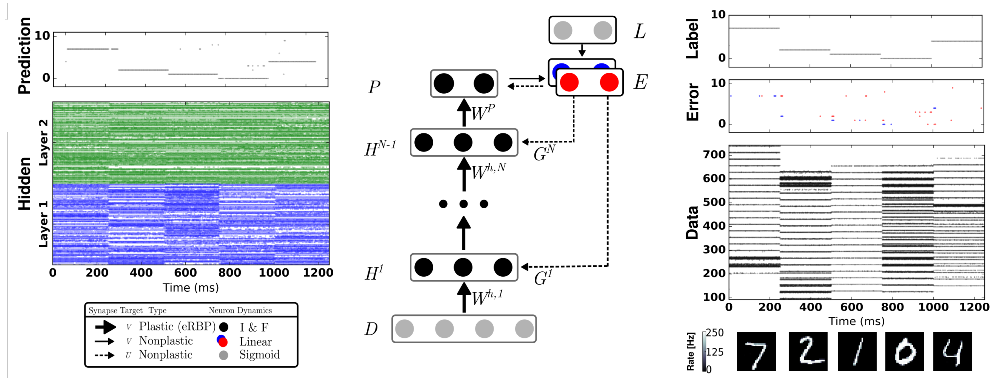


Figure 7.1: Network Architecture for Event-driven Random Backpropagation (eRBP). Reproduced from [6]
.

the pair of error neurons in a one-to-one fashion.

One limitation eRBP is related to the "loop duration", *i.e.* the duration necessary from the input onset to a stable response in the error neurons. A related problem is layerwise locking in deep neural networks: because errors are backpropagated from the top layers, hidden layers must wait until the prediction is made available [192]. This duration scales with the number of layers, limiting eRBP scalability for very deep networks. The loop duration problem is caused by the temporal dynamics of the spiking neuron, which are not taken into account in (7.3).

This problem can be partly overcome by maintaining traces of the input spiking activity, and a solution was reported in [172]. Their rule, called Superspike is derived from gradient descent over the LIF neuron dynamics, resulting in the following three-factor rule:

$$\Delta W_{ij}^{SS} \propto \alpha * (Error_i \rho'(U_i)(\epsilon_{pre} * S_j))) \tag{7.4}$$

where ρ describes the slope of the activation function ρ at the membrane potential U_i, and ϵ_{pre} here is the response of the post-synaptic neuron to a pre-synaptic spike (the impulse response function at U).

Interestingly, both (7.4) and (7.3) rules are reminiscent of STDP but include further terms that vary according to some external modulation, itself related to some task. Unsurprisingly, the three terms in (7.2) can be related to the classical Widrow-Hoff (Delta) rule, where the first term is the error, the second is the derivative of the output activation function, and the third term is the input.

The loop duration is only partly solved with (7.4), as α and ϵ introduce memory of the previous activity into the synapses. However, this is only an approximation, as the dynamics of every layer leading to the top layer must be taken into account with one additional temporal convolution per layer. As a result, (7.4) and (7.3) do not scale well with multiple layers.

Local Errors A more effective method to overcome the loop duration and the layerwise locking problem is to use synthetic gradients: gradients that can be computed locally, at every layer. Synthetic gradients were initially proposed to decouple one or more layers from the rest of the network to prevent layerwise locking [192]. Synthetic gradients usually involve an outer loop consisting of a full backpropagation through the network. While this provides convergence guarantees, a full backpropagation step cannot be done locally in spiking neural networks. Instead, relying on initialization of the local random classifier weights and forgoing the outer loop training yields good empirical results [164].

Using layerwise local classifiers [164], the gradients are computed locally using pseudo-targets (for classification, the labels themselves). To take the temporal dynamics of the neurons into account, the learning rule is similar to SuperSpike [172]. However, the gradients are computed locally through a fixed random projection of the network activities into a local classifier. Learning is achieved using a local rate-based cost function reminiscent of readout neurons in liquid state machines [193], but where the readout is performed over a fixed random combination of the neuron outputs. The readout does not have a temporal convolution

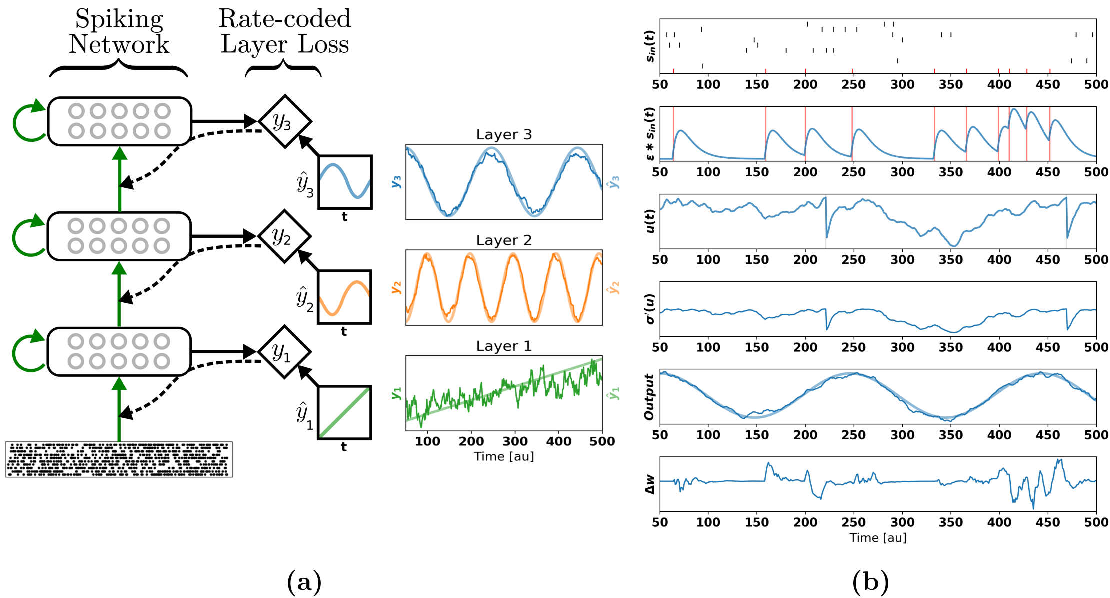


Figure 7.2: Deep continuous local learning example.Reproduced from [7].

term in the cost function, the absence of which enables linear scaling, and does not prevent learning precise temporal spike trains (7.2). The resulting learning dynamics are called DEep COntinuous Local LEarning (DECOLLE), and written:

$$\Delta W_{ij}^{DECOLLE} \propto (\sum_k g_{ik} Error_k)\rho'(U_i)(\epsilon_{pre} * S_j)). \tag{7.5}$$

Here, the error is computed with respect to a random linear combination of the neuron outputs: $Error_k = target_k - \sum_i g_{ik} S_i$. While SuperSpike scales at least quadratically with the number of neurons, learning with local errors scales linearly [7]. Linearity greatly improves the memory and computational cost of computing the weight updates and simplifies potential RRAM implementations (see 7.2.2).

In figure 7.2 (a), Each layer consists of spiking neurons with continuous dynamics. Each layer feeds into a local classifier through fixed, random connections (diamond-shaped, y). The classifier is trained to produce auxiliary targets $\hat{y}$. Errors in the local classifiers are propagated through the random connections to train the input weights, but no further (curvy,

dashed line). To simplify the learning rule and enable linear scaling of the computations, the cost function is formulated using a rate code. The states of the spiking neurons (membrane potential, synaptic states, refractory state) are carried forward in time. Consequently, even in the absence of recurrent connections, the neurons are stateful in the sense of recurrent neural networks. In figure 7.2 (b), Snapshot of the neural states illustrating the DECOLLE learning rule in the top layer. In this example, the network is trained to produce three time-varying pseudotargets $\hat{y}_1$, $\hat{y}_2$ and $\hat{y}_3$.

7.2 Stochastic Spiking Neural Networks

Up to now, we have considered fully deterministic neuron and synapse models. However, the writing of RRAMs values are stochastic. Additionally, analog VLSI neuron circuits have significant variability across neurons due to fabrication mismatch (fixed pattern noise) and behave stochastically due to noise intrinsic to the device operation. The variability at the device level can be taken in to account in SNN models and sometimes be exploited for improving learning performance and implementing probabilistic inference [194, 195]. Here, we list avenues for implementing online learning in memory devices that exploit the stochasticity in the neurons and synapses.

A stochastic model of the neurons can be expressed as:

$$P(S_i|\mathbf{s}) = \rho(U_i(t)) \tag{7.6}$$

where ρ_i is the stochastic intensity (the equivalent of the activation function in artificial neurons), and η and ϵ are kernels that reflect neural and synaptic dynamics, e.g. refractoriness, reset and postsynaptic potentials [178]. The stochastic intensity can be derived or estimated experimentally if the noiseless membrane potential ($U_i(t)$) can be measured at the times of the spike [196]. This type of stochastic neuron model drives numerous investigations

in theoretical neuroscience and forms the starting point for other types of adapting spiking neural networks capable of efficient communication [197].

7.2.1 Learning in Stochastic Spiking Neural Networks

Neural and synaptic unreliability can induce the necessary stochasticity without requiring a dedicated source of stochastic inputs, for example, the unreliable transmission of synaptic vesicles in biological neurons. This is a well-studied phenomenon [198, 199], and many studies suggested it as a major source of stochasticity in the brain [200, 201, 202, 203]. In the cortex, synaptic failures were argued to reduce energy consumption while maintaining the computational information transmitted by the post-synaptic neuron [204]. More recent work suggested synaptic sampling as a mechanism for representing uncertainty in the brain, and its role in synaptic plasticity and rewiring [205].

Strikingly, the simplest model of synaptic unreliability, a "*blank-out*" synapse, can improve the performance of spiking neural networks in practical machine learning tasks over existing solutions, while being extremely easy to implement in hardware [206], and often naturally occurring in emerging memory technologies [207, 208, 209].

One approach to learning with such neurons and synapses is Event-Driven Contrastive Divergence (ECD), using ideas borrowed from Contrastive Divergence in restricted Boltzmann machines [210]. The stochastic neural network produces samples from a probability distribution, and STDP carries out the weight updates according to the Contrastive Divergence rule in an online, asynchronous fashion. In terms of the three-factor rule above, ECD can be written:

$$\Delta W_{ij}^{ECD} = M_i(t)\Delta W_{ij}^{STDP} \tag{7.7}$$

where $M_i(t) = 1$ during the "data" phase and $M_i(t) = -1$ during the "reconstruction" phase.

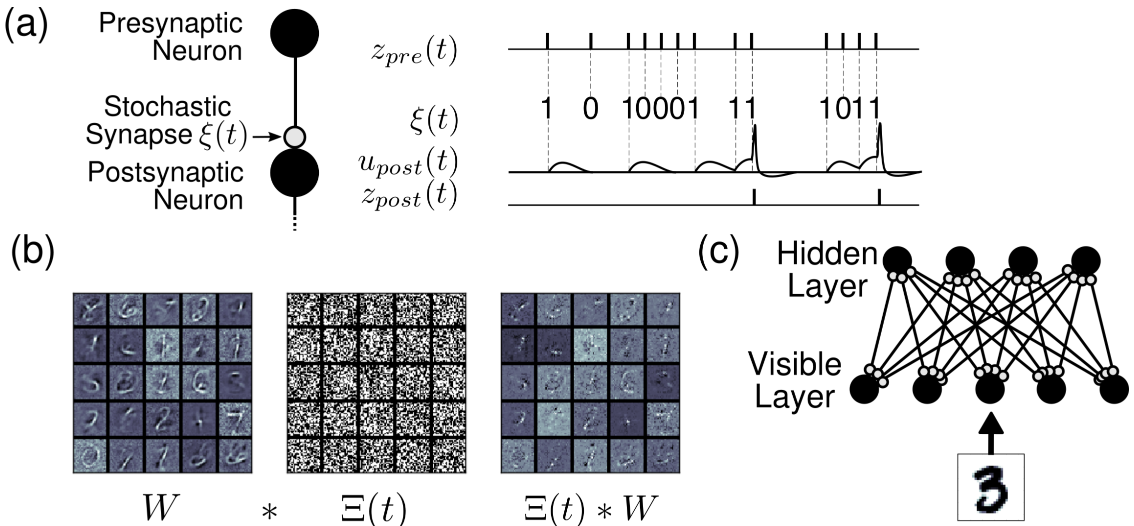


Figure 7.3: The Synaptic Sampling Machines (SSM). (a) At every occurrence of a pre-synaptic event, a pre-synaptic event is propagated to the post-synaptic neuron with probability p. (b) Synaptic stochasticity can be viewed as a continuous DropConnect method [8] where weights are masked by a binary matrix $\Theta(t)$, where $*$ denotes element-wise multiplication. (c) SSM Network architecture, consisting of a visible and a hidden layer. Reproduced from [9]

.

These neural networks can be viewed as a stochastic counterpart of Hopfield networks [211], but where stochasticity is caused by multiplicative noise at the connections (synapses) or at the nodes (neurons).

ECD requires symmetric weights, which is difficult to achieve due to the weight transport problem discussed above. A variation of ECD called random Contrastive Hebbian learning (rCHL) [212], replaces the transpose of the synaptic weights with fixed random matrices. This was performed similarly to Feedback Alignment (FDA) [152]. Contrastive Hebbian Learning (CHL) is similar to Contrastive Divergence, but it employs continuous nonlinear dynamics at the neuronal level. Like Contrastive Divergence, it does not rely on a special circuitry to compute gradients (but can be interpreted as the gradient of an objective function), allows information to flow in a coupled, synchronous way, and is grounded upon Hebb's learning rule. CHL uses feedback to transmit information from the output layer to hidden(s) layer(s), and in instances when the feedback gain is small (such as in the clamped

phase), has been demonstrated by Xie and Seung to be equivalent to Backpropagation [213]. Using this approach, the information necessary for learning propagates backward, though it is not transmitted through the same axons (as required in the symmetric case), but instead via separate pathways or neural populations.

Equilibrium Propagation (EP) describes another modification of CHL that generalizes the objective functions it can solve and improve on its theoretical groundings. In EP the neuron dynamics are derived from the energy function, however, EP requires symmetric weights. The energy function used in Equilibrium Propagation includes a mean-squared error term on the output units, allowing the output to be weakly clamped to the true outputs. (e.g. labels). The neuron model takes a form which is reminiscent of the LIF neuron. The recurrent dynamics in the network affect the other layers in the network, similarly to CHL. Both rCHL and EP were formulated for continuous (rate-based) neurons, although their implementation with spikes is straightforward following the same approach as Synaptic Sampling Machines (SSM).

Learning in stochastic neuron networks can also be performed using the surrogate gradient approach and three-factor rules. In this case, a simple approach is to use the stochastic intensity ρ as a drop-in replacement of the neural activation function for purposes of computing the weight updates [154]. In this case, stochasticity plays a regularization role similar to dropout in deep learning [153] and weight normalization [214] an online modification of batch normalization.

7.2.2 Three Factor Learning in Memristor Arrays

So far, we have discussed how to implement gradient-based learning as local synaptic plasticity rules in SNN. In many cases, gradient-based learning provides superior results compared to STDP and take the form of three-factor rules. These rules are biologically credible since

pre- and post-synaptic activities are available at the level of the neuron and neurotransmitters in the brain can carry the extrinsic factor. However, besides the LTP and LTD asymmetry problems already discussed, the implementation of the three-factors in memristor arrays come with certain challenges. The first challenge concerns the implementation of the synaptic traces. In certain simple cases, such as when the subthreshold neural dynamics are linear, only one trace for each neuron involved in the learning connections is required for learning [7], similar to the STDP case. Previous work has demonstrated STDP learning in RRAM and hence capture some form of a neural trace. The majority of these include additional CMOS circuitry in a "hybrid configuration" [215] to enable STDP. The simplest of these implementations consists of a 2T1R configuration that enables an update when both terminals are high (both spike). While this is sufficient for the case where $\alpha = \beta = 0$, an additional mechanism that filters the spike is necessary to recover STDP like curves when $\alpha > 0$ or $\beta > 0$. This can be achieved with a circuit that is similar to that of the synapse [216] or calcium variables [217]. For more complex neuron dynamics (such as non-linear neuron dynamics), then at least one trace *per synapse* is required, [218], and ideally, one trace per connection and per neuron [191].

Since the synaptic trace dynamics follow similar first order (RC) dynamics, the same circuits used to implement first-order synaptic dynamics can be used to implement synaptic traces [219] or dedicated calcium dynamics circuits [220]. However, scalability can become an issue: the same amount of memory for storing the weights is necessary for computing the traces, but the latter must be carried out continuously. One potential solution comes from recent work in using diffusive memristors to implement the leaky dynamics of integrate and fire neurons [221], which can be used for computing neural traces.

The second issue concerns the modulation. In many gradient-based three-factor rules, the modulation of the learning rule is specific to each neuron, not each synapse. This means that a similar approach to eRBP (7.3), where a separate neuron compartment is used for

maintaining the modulation factor can in principle be used in memristor arrays, *i.e.* the weight update can consist in the two factors $(\epsilon_{pre} * S_j))$, where $M_i\rho'(U_i)$.

Additionally, the variability in the conductance reading and writing can cause the learning to fail or slow down. Independent noise in the read or write is not a problem and can even help learning, as discussed in the stochastic SNN section. Fixed pattern noise, however, can be problematic as it translates into variable learning rates per weight and can impair learning.

Bibliography

[1] S. Furber, "Large-scale neuromorphic computing systems," *Journal of Neural Engineering*, vol. 13, no. 5, p. 051001, 2016.

[2] T. H. Intel Corporation, *Intels Pohoiki Beach*, Jul 2019. [Online]. Available: https://newsroom.intel.com/news/intels-pohoiki-beach-64-chip-neuromorphic-system-delivers-breakthrough-results-research-tests/#gs.s9bnly

[3] D. C. Daly, L. C. Fujino, and K. C. Smith, "Through the looking glass-the 2017 edition: Trends in solid-state circuits from isscc," *IEEE Solid-State Circuits Magazine*, vol. 9, no. 1, pp. 12–22, 2017.

[4] G. H. Kim, H. Ju, M. K. Yang, D. K. Lee, J. W. Choi, J. H. Jang, S. G. Lee, I. S. Cha, B. K. Park, J. H. Han *et al.*, "Four-bits-per-cell operation in an hfo2-based resistive switching device," *Small*, vol. 13, no. 40, p. 1701781, 2017.

[5] M. E. Fouda and et al., "Modeling and analysis of passive switching crossbar arrays," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 270–282, 2018.

[6] E. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," in *2017 IEEE International Symposium on Circuits and Systems*, May 2017.

[7] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity for deep continuous local learning," *arXiv preprint arXiv:1812.10766*, Nov 2018.

[8] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

[9] E. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, "Stochastic synapses enable efficient brain-inspired learning machines," *Frontiers in Neuroscience*, vol. 10, no. 241, 2016.

[10] S. Yu and P.-Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, 2016.

[11] J. Park and et al., "Tio x-based rram synapse with 64-levels of conductance and symmetric conductance change by adopting a hybrid pulse scheme for neuromorphic computing," *IEEE Electron Device Letters*, vol. 37, no. 12, pp. 1559–1562, 2016.

[12] M. Zidan, H. Omran, R. Naous, A. Sultan, H. Fahmy, W. Lu, and K. N. Salama, "Single-readout high-density memristor crossbar," *Scientific reports*, vol. 6, p. 18863, 2016.

[13] R. Naous et al., "Pilot assisted readout for passive memristor crossbars," *Microelectronics Journal*, vol. 54, pp. 48–58, 2016.

[14] "Worldwide internet of things forecast update, 2016-2020."

[15] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–36, 1990.

[16] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[17] E. Chicca, F. Stefanini, and G. Indiveri, "Neuromorphic electronic circuits for building autonomous cognitive systems," *Proceedings of IEEE*, 2013.

[18] J. Park, S. Ha, T. Yu, E. Neftci, and G. Cauwenberghs, "A 65k-neuron 73-mevents/s 22-pj/event asynchronous micro-pipelined integrate-and-fire array transceiver," in *Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, Nov 2014.

[19] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *International Symposium on Circuits and Systems, ISCAS 2010*. IEEE, 2010, pp. 1947–1950.

[20] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[21] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, P. Joshi, A. Lines, A. Wild, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. PP, no. 99, pp. 1–1, 2018.

[22] S. B. Furber, F. Galluppi, S. Temple, L. Plana *et al.*, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[23] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers in neuroscience*, vol. 9, 2015.

[24] M. Courbariaux, Y. Bengio, and J.-P. David, "Low precision arithmetic for deep learning," *arXiv preprint arXiv:1412.7024*, 2014.

[25] L. Wilson, "International technology roadmap for semiconductors (itrs)," *Semiconductor Industry Association*, 2013.

[26] H.-S. Wong et al, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[27] K. M. Kim, J. J. Yang, J. P. Strachan, E. M. Grafals, N. Ge, N. D. Melendez, Z. Li, and R. S. Williams, "Voltage divider effect for the improvement of variability and endurance of taox memristor," *Scientific reports*, vol. 6, 2016.

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

[29] P. Hou, J. Wang, X. Zhong, and Y. Wu, "A ferroelectric memristor based on the migration of oxygen vacancies," *RSC Advances*, 2016.

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

[31] A. G. Radwan and M. E. Fouda, *On the Mathematical Modeling of Memristor, Memcapacitor, and Meminductor.* Springer, 2015, vol. 26.

[32] 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.

[33] J. J. Yang, M.-X. Zhang, M. D. Pickett, F. Miao, J. P. Strachan, W.-D. Li, W. Yi, D. A. Ohlberg, B. J. Choi, W. Wu *et al.*, "Engineering nonlinearity into memristors for passive crossbar applications," *Applied Physics Letters*, vol. 100, no. 11, p. 113501, 2012.

[34] F. Miao, J. P. Strachan, J. J. Yang, M.-X. Zhang, I. Goldfarb, A. C. Torrezan, P. Eschbach, R. D. Kelley, G. Medeiros-Ribeiro, and R. S. Williams, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *Advanced Materials*, vol. 23, no. 47, pp. 5633–5640, 2011.

[35] D. M. Bromberg, D. H. Morris, L. Pileggi, and J.-G. Zhu, "Novel stt-mtj device enabling all-metallic logic circuits," *Magnetics, IEEE Transactions on*, vol. 48, no. 11, pp. 3215–3218, 2012.

[36] C. Chappert, A. Fert, and F. N. Van Dau, "The emergence of spin electronics in data storage," *Nature materials*, vol. 6, no. 11, pp. 813–823, 2007.

[37] A. Chanthbouala, A. Crassous, V. Garcia, K. Bouzehouane, S. Fusil, X. Moya, J. Allibe, B. Dlubak, J. Grollier, S. Xavier *et al.*, "Solid-state memories based on ferroelectric tunnel junctions," *Nature nanotechnology*, vol. 7, no. 2, pp. 101–104, 2012.

[38] J. S. Friedman and A. V. Sahakian, "Complementary magnetic tunnel junction logic," *Electron Devices, IEEE Transactions on*, vol. 61, no. 4, pp. 1207–1210, 2014.

[39] S. H. Jo, T. Kumar, S. Narayanan, W. D. Lu, and H. Nazarian, "3d-stackable crossbar resistive memory based on field assisted superlinear threshold (fast) selector," in *Electron Devices Meeting (IEDM), 2014 IEEE International*. IEEE, 2014, pp. 6–7.

[40] Y. Bai, H. Wu, K. Wang, R. Wu, L. Song, T. Li, J. Wang, Z. Yu, and H. Qian, "Stacked 3d rram array with graphene/cnt as edge electrodes," *Scientific reports*, vol. 5, 2015.

[41] H. Eggers, P. Lysaght, H. Dick, and G. McGregor, "Fast reconfigurable crossbar switching in fpgas," in *Field-Programmable Logic Smart Applications, New Paradigms and Compilers*. Springer, 1996, pp. 297–306.

[42] D. Bafumba-Lokilo, Y. Savaria, and J.-P. David, "Generic crossbar network on chip for fpga mpsocs," in *Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on*. IEEE, 2008, pp. 269–272.

[43] C.-L. Tsai, F. Xiong, E. Pop, and M. Shim, "Resistive random access memory enabled by carbon nanotube crossbar electrodes," *Acs Nano*, vol. 7, no. 6, pp. 5360–5366, 2013.

[44] G. Kim *et al.*, "Schottky diode with excellent performance for large integration density of crossbar resistive memory; may 24, 2012; 2012 american institute of physics," *Applied Physics Letters*, pp. 213 508–1.

[45] S. Kannan, J. Rajendran, R. Karri, and O. Sinanoglu, "Sneak-path testing of crossbar-based nonvolatile random access memories," *Nanotechnology, IEEE Transactions on*, vol. 12, no. 3, pp. 413–426, 2013.

[46] M. A. Zidan, A. M. Eltawil, F. Kurdahi, H. A. Fahmy, and K. N. Salama, "Memristor multiport readout: A closed-form solution for sneak paths," *Nanotechnology, IEEE Transactions on*, vol. 13, no. 2, pp. 274–282, 2014.

[47] 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.

[48] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 476–488.

[49] E. Linn, R. Rosezin, S. Tappertzhofen, R. Waser *et al.*, "Beyond von neumann? logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[50] R. Rosezin, E. Linn, C. Kügeler, R. Bruchhaus, and R. Waser, “Crossbar logic using bipolar and complementary resistive switches,” *Electron Device Letters, IEEE*, vol. 32, no. 6, pp. 710–712, 2011.

[51] A. Siemon, S. Menzel, R. Waser, and E. Linn, “A complementary resistive switch-based crossbar array adder,” *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 5, no. 1, pp. 64–74, 2015.

[52] P. J. Kuekes, G. S. Snider, and R. S. Williams, “Crossbar nanocomputers,” *Scientific American*, vol. 293, no. 5, pp. 72–80, 2005.

[53] D. B. Strukov and K. K. Likharev, “Reconfigurable nano-crossbar architectures,” *Nanoelectronics and Information Technology*, pp. 543–562, 2012.

[54] E. Zamanidoost, F. M. Bayat, D. Strukov, and I. Kataeva, “Manhattan rule training for memristive crossbar circuit pattern classifiers,” in *Intelligent Signal Processing (WISP), 2015 IEEE 9th International Symposium on*. IEEE, 2015, pp. 1–6.

[55] L. Zhang and Z. Chang, “A memristor-crossbar/cmos integrated network for pattern classification and recognition,” in *International Conference on Computer Information Systems and Industrial Applications*. Atlantis Press, 2015.

[56] C. Bennett et al, “Supervised learning with organic memristor devices and prospects for neural crossbar arrays,” in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH¿¿ 15)*. IEEE, 2015, pp. 181–186.

[57] M. Hu et al, “Memristor crossbar-based neuromorphic computing system: A case study,” *IEEE transactions on neural networks and learning systems*, vol. 25, no. 10, pp. 1864–1878, 2014.

[58] S. N. Truong and K.-S. Min, “New memristor-based crossbar array architecture with 50-% area reduction and 48-% power saving for matrix-vector multiplication of analog neuromorphic computing,” *Journal of semiconductor technology and science*, vol. 14, no. 3, pp. 356–363, 2014.

[59] P. Lugli, A. Mahmoud, G. Csaba, M. Algasinger, M. Stutzmann, and U. Rührmair, “Physical unclonable functions based on crossbar arrays for cryptographic applications,” *International journal of circuit theory and applications*, vol. 41, no. 6, pp. 619–633, 2013.

[60] U. Ruhrmair, C. Jaeger, M. Bator, M. Stutzmann, P. Lugli, and G. Csaba, “Applications of high-capacity crossbar memories in cryptography,” *Nanotechnology, IEEE Transactions on*, vol. 10, no. 3, pp. 489–498, 2011.

[61] M. Prezioso et al, “Training and operation of an integrated neuromorphic network based on metal-oxide memristors,” *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[62] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature Communications*, vol. 9, no. 1, p. 2385, 2018.

[63] C.-C. C. et al., "Mitigating asymmetric nonlinear weight update effects in hardware neural network based on analog resistive synapse," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2017.

[64] M. E. Fouda, J. Lee, A. M. Eltawil, and F. Kurdahi, "Overcoming crossbar nonidealities in binary neural networks through learning," in *2018 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH).* IEEE, 2018, pp. 1–3.

[65] S. Yu, "Neuro-inspired computing with emerging nonvolatile memorys," *Proceedings of the IEEE*, vol. 106, no. 2, pp. 260–285, 2018.

[66] M. Zhao, H. Wu, B. Gao, X. Sun, Y. Liu, P. Yao, Y. Xi, X. Li, Q. Zhang, K. Wang *et al.*, "Characterizing endurance degradation of incremental switching in analog rram for neuromorphic systems," in *2018 IEEE International Electron Devices Meeting (IEDM).* IEEE, 2018, pp. 20–2.

[67] B. Chen, Y. Lu, B. Gao, Y. Fu, F. Zhang, P. Huang, Y. Chen, L. Liu, X. Liu, J. Kang *et al.*, "Physical mechanisms of endurance degradation in tmo-rram," in *2011 International Electron Devices Meeting.* IEEE, 2011, pp. 12–3.

[68] C. Nail, G. Molas, P. Blaise, G. Piccolboni, B. Sklenard, C. Cagli, M. Bernard, A. Roule, M. Azzaz, E. Vianello *et al.*, "Understanding rram endurance, retention and window margin trade-off using experimental results and simulations," in *2016 IEEE International Electron Devices Meeting (IEDM).* IEEE, 2016, pp. 4–5.

[69] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[70] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "Rx-caffe: Framework for evaluating and training deep neural networks on resistive crossbars," *arXiv preprint arXiv:1809.00072*, 2018.

[71] M. Azzaz, E. Vianello, B. Sklenard, P. Blaise, A. Roule, C. Sabbione, S. Bernasconi, C. Charpin, C. Cagli, E. Jalaguier *et al.*, "Endurance/retention trade off in hfox and taox based rram," in *2016 IEEE 8th International Memory Workshop (IMW).* IEEE, 2016, pp. 1–4.

[72] M. E. Fouda, A. M. Eltawil, and F. Kurdahi, "Modeling and analysis of passive switching crossbar arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 270–282, 2018.

[73] ——, "On resistive memories: One step row readout technique and sensing circuitry," *arXiv preprint arXiv:1903.01512*, 2019.

[74] H. Kim, T. Kim, J. Kim, and J.-J. Kim, "Deep neural network optimized to resistive memory with nonlinear current-voltage characteristics," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 2, p. 15, 2018.

[75] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, 2008.

[76] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[77] J. Woo and S. Yu, "Resistive memory-based analog synapse: The pursuit for linear and symmetric weight update," *IEEE Nanotechnology Magazine*, vol. 12, no. 3, pp. 36–44, 2018.

[78] F. M. Puglisi, C. Wenger, and P. Pavan, "A novel program-verify algorithm for multi-bit operation in hfo 2 rram," *IEEE Electron Device Letters*, vol. 36, no. 10, pp. 1030–1032, 2015.

[79] C.-Y. Chen, H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu, and F. T. Chen, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 180–190, 2014.

[80] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at fault tolerance in rram computing systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 102–115, 2017.

[81] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electronics*, vol. 1, no. 1, p. 52, 2018.

[82] J. A. Davis and J. D. Meindl, "Compact distributed rlc interconnect models. i. single line transient, time delay, and overshoot expressions," *IEEE Transactions on Electron Devices*, vol. 47, no. 11, pp. 2068–2077, 2000.

[83] ——, "Compact distributed rlc interconnect models-part ii: Coupled line transient expressions and peak crosstalk in multilevel networks," *IEEE Transactions on Electron Devices*, vol. 47, no. 11, pp. 2078–2087, 2000.

[84] K. Agarwal, D. Sylvester, and D. Blaauw, "Modeling and analysis of crosstalk noise in coupled rlc interconnects," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 5, pp. 892–901, 2006.

[85] A. Chen, "A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics," *Electron Devices, IEEE Transactions on*, vol. 60, no. 4, pp. 1318–1326, 2013.

[86] A. Flocke, T. Noll, C. Kügeler, C. Nauenheim, and R. Waser, "A fundamental analysis of nano-crossbars with non-linear switching materials and its impact on tio2 as a resistive layer," in *Nanotechnology, 2008. NANO'08. 8th IEEE Conference on.* IEEE, 2008, pp. 319–322.

[87] J. F. Shackelford, Y.-H. Han, S. Kim, and S.-H. Kwon, *CRC materials science and engineering handbook*. CRC press, 2016.

[88] D. Josell, S. H. Brongersma, and Z. Tőkei, "Size-dependent resistivity in nanoscale interconnects," *Annual Review of Materials Research*, vol. 39, pp. 231–254, 2009.

[89] G. Hegde, R. C. Bowen, and M. S. Rodder, "Lower limits of line resistance in back end of line cu interconnects," *arXiv preprint arXiv:1601.06675*, 2016.

[90] G. Shomalnasab and L. Zhang, "New analytic model of coupling and substrate capacitance in nanometer technologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 7, pp. 1268–1280, 2015.

[91] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, no. 48, p. 485203, 2011.

[92] B. J. Choi, A. C. Torrezan, J. P. Strachan, P. Kotula, A. Lohn, M. J. Marinella, Z. Li, R. S. Williams, and J. J. Yang, "High-speed and low-energy nitride memristors," *Advanced Functional Materials*, vol. 26, no. 29, pp. 5290–5296, 2016.

[93] P. Narayanan, G. W. Burr, R. S. Shenoy, S. Stephens, K. Virwani, A. Padilla, B. N. Kurdi, and K. Gopalakrishnan, "Exploring the design space for crossbar arrays built with mixed-ionic-electronic-conduction (miec) access devices," *IEEE Journal of the Electron Devices Society*, vol. 3, no. 5, pp. 423–434, 2015.

[94] K. Ogata, *Modern control engineering.* Prentice Hall PTR, 2001.

[95] A. Chen, "Nonlinearity and asymmetry for device selection in cross-bar memory arrays," *IEEE Transactions on Electron Devices*, vol. 62, no. 9, pp. 2857–2864, 2015.

[96] M. Qureshi, W. Yi, G. Medeiros-Ribeiro, and R. Williams, "Ac sense technique for memristor crossbar," *Electronics letters*, vol. 48, no. 13, p. 1, 2012.

[97] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, no. 1, pp. 55–63, 1948.

[98] T.-M. Lin and C. A. Mead, "Signal delay in general rc networks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 3, no. 4, pp. 331–349, 1984.

[99] T.-y. Liu and et al., "A 130.7-mm^2 2-layer 32-gb reram memory device in 24-nm technology," *IEEE J. Solid-State Circuits*, pp. 140–153, 2014.

[100] A. Kawahara and et al., "An 8 mb multi-layered cross-point reram macro with 443 mb/s write throughput," *IEEE J.Solid-State Circuits*, 2013.

[101] H. Manem, G. S. Rose, X. He, and W. Wang, "Design considerations for variation tolerant multilevel cmos/nano memristor memory," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI.* ACM, 2010, pp. 287–292.

[102] J. Liang and H.-S. P. Wong, "Cross-point memory array without cell selectorsdevice characteristics and data storage pattern dependencies," *IEEE Trans. Electron Devices*, vol. 57, no. 10, pp. 2531–2538, 2010.

[103] M. Fouda, A. Eltawil, and F. Kurdahi, "On one step row readout technique of selectorless resistive arrays," in *Cir. and Sys., Int. Midwest Symp.* IEEE, 2017.

[104] D. Ielmini, "Resistive switching memories based on metal oxides: mechanisms, reliability and scaling," *Semiconductor Science and Technology*, vol. 31, no. 6, p. 063002, 2016.

[105] A. Sedra, G. W. Roberts, and F. Gohh, "The current conveyor: history, progress and new results," in *IEE proceedings*, vol. 137, no. 2, 1990, pp. 78–87.

[106] H. O. Elwan and A. M. Soliman, "Cmos differential current conveyors and applications for analog vlsi," *Analog Integr. Circuits Signal Process.*, vol. 11, no. 1, pp. 35–45, 1996.

[107] Y. Huang, H. Schleifer, and D. Killat, "Design and analysis of novel dynamic latched comparator with reduced kickback noise for high-speed adcs," in *Cir. Theory and Des., European Conf.* IEEE, 2013, pp. 1–4.

[108] D. Balobas and N. Konofaos, "Design of low-power high-performance 2–4 and 4–16 mixed-logic line decoders," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 2, pp. 176–180, 2017.

[109] F. F. Khan and A. Ye, "Measuring the accuracy of minimum width transistor area in estimating fpga layout area," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on.* IEEE, 2015, pp. 223–226.

[110] Z. Chen, C. Schoeny, and L. Dolecek, "Coding assisted adaptive thresholding for sneak-path mitigation in resistive memories," in *IEEE Information Theory Workshop (ITW).* IEEE, 2018, pp. 1–5.

[111] Y. Ben-Hur and Y. Cassuto, "Detection and coding schemes for sneak-path interference in resistive memory arrays," *IEEE Transactions on Communications*, 2019.

[112] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[113] M. Alsan and E. Telatar, "A simple proof of polarization and polarization for non-stationary memoryless channels," *IEEE Trans. Inf. Theory*, vol. 62, no. 9, pp. 4873–4878, 2016.

[114] E. Arikan, "Systematic polar coding," *IEEE communications letters*, vol. 15, no. 8, pp. 860–862, 2011.

[115] K. Niu, K. Chen, and J.-R. Lin, "Beyond turbo codes: Rate-compatible punctured polar codes," in *IEEE Int. Conf. on Communications (ICC)*. IEEE, 2013, pp. 3423–3427.

[116] A. Chen, "Analysis of partial bias schemes for the writing of crossbar memory arrays," *IEEE Transactions on Electron Devices*, vol. 62, no. 9, pp. 2845–2849, 2015.

[117] R. Micheloni and L. Crippa, "3d stacked nand flash memories," in *3D Flash Memories*. Springer, 2016, pp. 63–83.

[118] S. Anthony, "Hp and sandisk join forces to finally bring memristor like tech to market," Oct 2015.

[119] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[120] R. Gabrys and L. Dolecek, "Coding for the binary asymmetric channel," in *Int. Conf. Comp., Net. and Comm. (ICNC)*, Jan 2012, pp. 461–465.

[121] M. Fouda, E. Neftci, A. M. Eltawil, and F. Kurdahi, "Independent component analysis using rrams," *IEEE Transactions on Nanotechnology*, 2018.

[122] Y. Jeong, M. A. Zidan, and W. D. Lu, "Parasitic effect analysis in memristor-array-based neuromorphic systems," *IEEE Transactions on Nanotechnology*, vol. 17, no. 1, pp. 184–193, 2018.

[123] F. M. Bayat, M. Prezioso, B. Chakrabarti, H. Nili, I. Kataeva, and D. Strukov, "Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits," *Nature communications*, vol. 9, no. 1, p. 2331, 2018.

[124] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 367–379, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/3007787.3001177

[125] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 uj/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE Journal of Solid-State Circuits*, no. 1, pp. 158–172, Jan. 2019.

[126] H. Sim and J. Lee, "Cost-effective stochastic MAC circuits for deep neural networks," *Neural Networks*, vol. 117, pp. 152–162, Sep. 2019.

[127] S. Yu, Z. Li, P. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 mb rram macro chip for classification and online training," in *2016 IEEE International Electron Devices Meeting (IEDM)*, Dec 2016, pp. 16.2.1–16.2.4.

[128] X. Sun, X. Peng, P.-Y. Chen, R. Liu, J.-s. Seo, and S. Yu, "Fully parallel rram synaptic array for implementing binary neural network with (+ 1,- 1) weights and (+ 1, 0) neurons," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 2018, pp. 574–579.

[129] X. Sun, S. Yin, X. Peng, R. Liu, J. Seo, and S. Yu, "Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1423–1428.

[130] L. Ni, Z. Liu, H. Yu, and R. V. Joshi, "An energy-efficient digital reram-crossbar-based cnn with bitwise parallelism," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 3, pp. 37–46, Dec 2017.

[131] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on rram," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 782–787.

[132] E. Giacomin, T. Greenberg-Toledo, S. Kvatinsky, and P.-E. Gaillardon, "A robust digital rram-based convolutional block for low-power image processing and learning applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 2, pp. 643–654, 2019.

[133] A. Levisse, B. Giraud, J.-P. Noel, M. Moreau, and J.-M. Portal, "Sneakpath compensation circuit for programming and read operations in rram-based crosspoint architectures," in *2015 15th Non-Volatile Memory Technology Symposium (NVMTS)*. IEEE, 2015, pp. 1–4.

[134] M. E. Fouda, S. Lee, J. Lee, A. M. Eltawil, and F. Kurdahi, "Mask technique for fast and efficient training of binary resistive crossbar arrays," *IEEE Transactions on Nanotechnology*, pp. 1–1, 2019.

[135] C.-C. Chang, P.-C. Chen, T. Chou, I.-T. Wang, B. Hudec, C.-C. Chang, C.-M. Tsai, T.-S. Chang, and T.-H. Hou, "Mitigating asymmetric nonlinear weight update effects in hardware neural network based on analog resistive synapse," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 116–124, 2018.

[136] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[137] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang *et al.*, "Memristor-based analog computation and neural

network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[138] M. E. Fouda, A. Eltawil, and F. Kurdahi, "Activated current sensing circuit for resistive neuromorphic networks," in *2019 IEEE 17th International New Circuits and Systems Conference (NEWCAS)*. IEEE, 2019, pp. 1–4.

[139] H. Abunahla and B. Mohammad, *Memristor technology: synthesis and modeling for sensing and security applications*. Springer, 2018.

[140] I. Chakraborty, D. Roy, and K. Roy, "Technology aware training in memristive neuromorphic systems for nonideal synaptic crossbars," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 335–344, 2018.

[141] M. Lanza and et al., "Recommended methods to study resistive switching devices," *Advanced Electronic Materials*, p. 1800143, 2018.

[142] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Curran Associates Inc., 2016, pp. 4114–4122.

[143] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[144] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, p. 18, 2010.

[145] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, vol. 55, 2014.

[146] M. E. Fouda, A. M. Eltawil, and F. Kurdahi, "Minimal disturbed bits in writing resistive crossbar memories," in *2018 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2018, pp. 1–3.

[147] B. Traoré, P. Blaise, E. Vianello, H. Grampeix, S. Jeannot, L. Perniola, B. De Salvo, and Y. Nishi, "On the origin of low-resistance state retention failure in hfo 2-based rram and impact of doping/alloying," *IEEE Transactions on Electron Devices*, vol. 62, no. 12, pp. 4029–4036, 2015.

[148] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[149] S. Friedmann, J. Schemmel, A. Grübl, A. Hartel, M. Hock, and K. Meier, "Demonstrating hybrid learning in a flexible neuromorphic hardware system," *IEEE transactions on biomedical circuits and systems*, vol. 11, no. 1, pp. 128–142, 2017.

[150] G. Cauwenberghs, "Reverse engineering the cognitive brain," *Proceedings of the national academy of sciences*, vol. 110, no. 39, pp. 15 512–15 513, 2013.

[151] Q. Xia and J. J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature materials*, vol. 18, no. 4, pp. 309–323, 2019.

[152] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Communications*, vol. 7, 2016.

[153] E. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random backpropagation: Enabling neuromorphic deep learning machines," *Frontiers in Neuroscience*, vol. 11, p. 324, Jun 2017.

[154] E. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *Signal Processing Magazine, IEEE*, Dec 2019, (accepted).

[155] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, p. 60, 2018.

[156] D. H. Hoang and H. D. Nguyen, "A pca-based method for iot network traffic anomaly detection," in *Advanced Communication Technology (ICACT), 2018 20th International Conference on.* IEEE, 2018, pp. 381–386.

[157] S. Joshi, C. Kim, S. Ha, Y. M. Chi, and G. Cauwenberghs, "21.7 2pj/mac 14b 8× 8 linear transform mixed-signal spatial filter in 65nm cmos with 84db interference suppression," in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International.* IEEE, 2017, pp. 364–365.

[158] A. e. a. Hyvärinen, *Independent component analysis.* John Wiley & Sons, 2004, vol. 46.

[159] A. M. Hassan, C. Yang, C. Liu, H. H. Li, and Y. Chen, "Hybrid spiking-based multi-layered self-learning neuromorphic system based on memristor crossbar arrays," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2017, pp. 776–781.

[160] S. e. a. Choi, "Experimental demonstration of feature extraction and dimensionality reduction using memristor networks," *Nano letters*, vol. 17, no. 5, pp. 3113–3118, 2017.

[161] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, and W. D. Lu, "Sparse coding with memristor networks," *Nature nanotechnology*, vol. 12, no. 8, p. 784, 2017.

[162] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee, and W. D. Lu, "Reservoir computing using dynamic memristors for temporal information processing," *Nature communications*, vol. 8, no. 1, p. 2204, 2017.

[163] T. Isomura and T. Toyoizumi, "A local learning rule for independent component analysis," *Scientific reports*, vol. 6, p. 28073, 2016.

[164] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *arXiv preprint arXiv:1711.06756*, 2017.

[165] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," *arXiv preprint arXiv:1901.06656*, 2019.

[166] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[167] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[168] M. Kwak, J. Park, J. Woo, and H. Hwang, "Implementation of convolutional kernel function using 3-d tio x resistive switching devices for image processing," *IEEE Transactions on Electron Devices*, vol. 65, no. 10, pp. 4716–4718, 2018.

[169] I. Sanchez Esqueda, X. Yan, C. Rutherglen, A. Kane, T. Cain, P. Marsh, Q. Liu, K. Galatsis, H. Wang, and C. Zhou, "Aligned carbon nanotube synaptic transistors for large-scale neuromorphic computing," *ACS nano*, vol. 12, no. 7, pp. 7352–7361, 2018.

[170] P. Baldi, P. Sadowski, and Z. Lu, "Learning in the machine: The symmetries of the deep learning channel," *Neural Networks*, vol. 95, pp. 110–133, 2017.

[171] A. Nø kland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 1037–1045.

[172] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multi-layer spiking neural networks," *arXiv preprint arXiv:1705.11146*, 2017.

[173] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[174] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," *arXiv preprint arXiv:1810.08646*, 2018.

[175] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, 2016.

[176] E. O. Neftci, "Data and power efficient intelligence with neuromorphic learning machines," *iScience*, vol. 5, pp. 52–68, jul 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2589004218300865

[177] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, “A 0.086-mm ^{2}12.7-pJ/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS,” *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 1, pp. 145–158, 2018.

[178] W. Gerstner and W. Kistler, *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge University Press, 2002.

[179] G.-Q. Bi and M.-M. Poo, “Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type,” *J. Neurosci.*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[180] H. Z. Shouval, S. S.-H. Wang, and G. M. Wittenberg, “Spike timing dependent plasticity: a consequence of more fundamental learning rules,” *Frontiers in Computational Neuroscience*, vol. 4, p. 19, 2010.

[181] S. Lahiri and S. Ganguli, “A memory frontier for complex synapses,” in *Advances in Neural Information Processing Systems 26*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., 2013, pp. 1034–1042.

[182] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, “Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning,” *Neural computation*, vol. 18, no. 6, pp. 1318–1348, 2006.

[183] R. Urbanczik and W. Senn, “Learning by the dendritic prediction of somatic spiking,” *Neuron*, vol. 81, no. 3, pp. 521–528, 2014.

[184] G. Detorakis, S. Sheik, C. Augustine, S. Paul, B. Pedroni, N. Dutt, J. Krichmar, G. Cauwenberghs, and E. Neftci, “Neural and synaptic array transceiver: A brain-inspired computing framework for embedded learning,” *Frontiers in Neuroscience*, vol. 12, p. 583, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00583

[185] W. Schultz, “Getting formal with dopamine and reward,” *Neuron*, vol. 36, no. 2, pp. 241–263, 2002.

[186] V. Paille, E. Fino, K. Du, T. Morera-Herreras, S. Perez, J. H. Kotaleski, and L. Venance, “Gabaergic circuits control spike-timing-dependent plasticity,” *Journal of Neuroscience*, vol. 33, no. 22, pp. 9353–9363, 2013. [Online]. Available: http://www.jneurosci.org/content/33/22/9353

[187] R. Gütig and H. Sompolinsky, “The tempotron: a neuron that learns spike timing–based decisions,” *Nature Neuroscience*, vol. 9, pp. 420–428, 2006.

[188] S. M. Bohte, J. N. Kok, and J. A. La Poutré, “Spikeprop: backpropagation for networks of spiking neurons.” in *ESANN*, 2000, pp. 419–424.

[189] D. Huh and T. J. Sejnowski, “Gradient descent for spiking neural networks,” *arXiv preprint arXiv:1706.04698*, 2017.

[190] N. Anwani and B. Rajendran, "Normad-normalized approximate descent based supervised learning rule for spiking neurons," in *Neural Networks (IJCNN), 2015 International Joint Conference on.* IEEE, 2015, pp. 1–8.

[191] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[192] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," *arXiv preprint arXiv:1608.05343*, 2016.

[193] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[194] D. Querlioz, O. Bichler, A. F. Vincent, and C. Gamrat, "Bioinspired programming of memory devices for implementing an inference engine," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1398–1416, 2015.

[195] R. Naous, M. AlShedivat, E. Neftci, G. Cauwenberghs, and K. N. Salama, "Memristor-based neural networks: Synaptic versus neuronal stochasticity," *Aip Advances*, vol. 6, no. 11, p. 111304, 2016.

[196] R. Jolivet, A. Rauch, H.-R. Lüscher, and W. Gerstner, "Predicting spike timing of neocortical pyramidal neurons by simple threshold models," *Journal of computational neuroscience*, vol. 21, no. 1, pp. 35–49, 2006.

[197] D. Zambrano and S. M. Bohte, "Fast and efficient asynchronous neural computation with adapting spiking neural networks," *arXiv preprint arXiv:1609.02053*, 2016.

[198] B. Katz, *Nerve, muscle, and synapse.* McGraw-Hill New York, 1966.

[199] T. Branco and K. Staras, "The probability of neurotransmitter release: variability and feedback control at single synapses," *Nature Reviews Neuroscience*, vol. 10, no. 5, pp. 373–383, 2009.

[200] A. A. Faisal, L. P. Selen, and D. M. Wolpert, "Noise in the nervous system," *Nature Reviews Neuroscience*, vol. 9, no. 4, pp. 292–303, 2008.

[201] L. Abbott and W. Regehr, "Synaptic computation," *Nature*, vol. 431, pp. 796–803, October 2004.

[202] Y. Yarom and J. Hounsgaard, "Voltage fluctuations in neurons: signal or noise?" *Physiological Reviews*, vol. 91, no. 3, pp. 917–929, 2011.

[203] R. Moreno-Bote, "Poisson-like spiking in circuits with probabilistic synapses," *PLoS computational biology*, vol. 10, no. 7, p. e1003522, 2014.

[204] W. B. Levy and R. A. Baxter, "Energy-efficient neuronal computation via quantal synaptic failures," *The Journal of Neuroscience*, vol. 22, no. 11, pp. 4746–4755, 2002.

[205] D. Kappel, S. Habenschuss, R. Legenstein, and W. Maass, "Network plasticity as bayesian inference," *arXiv preprint arXiv:1504.05143*, 2015.

[206] D. Goldberg, G. Cauwenberghs, and A. Andreou, "Probabilistic synaptic weighting in a reconfigurable network of VLSI integrate-and-fire neurons," *Neural Networks*, vol. 14, no. 6–7, pp. 781–793, Sep 2001.

[207] S. Saïghi, C. G. Mayr, T. Serrano-Gotarredona, H. Schmidt, G. Lecerf, J. Tomas, J. Grollier, S. Boyn, A. F. Vincent, D. Querlioz *et al.*, "Plasticity in memristive devices for spiking neural networks," *Frontiers in neuroscience*, vol. 9, p. 51, 2015.

[208] M. Al-Shedivat, R. Naous, E. Neftci, G. Cauwenberghs, and K. Salama, "Inherently stochastic spiking neurons for probabilistic neural computation," in *IEEE EMBS Conference on Neural Engineering*, Apr 2015.

[209] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H.-S. P. Wong, "Stochastic learning in oxide binary synaptic device for neuromorphic computing," *Frontiers in neuroscience*, vol. 7, 2013.

[210] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[211] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[212] G. Detorakis, T. Bartley, and E. Neftci, "Contrastive hebbian learning with random feedback weights," *Neural Networks*, 2018, (accepted). [Online]. Available: https://arxiv.org/abs/1806.07406

[213] X. Xie and H. S. Seung, "Equivalence of backpropagation and contrastive hebbian learning in a layered network," *Neural computation*, vol. 15, no. 2, pp. 441–454, 2003.

[214] E. Neftci, "Stochastic synapses as resource for efficient deep learning machines," in *Electron Devices Meeting (IEDM), 2017 IEEE International*. IEEE, Dec 2017, pp. 11–1.

[215] D. Ielmini, "Brain-inspired computing with resistive switching memory (rram): Devices, synapses and neural networks," *Microelectronic Engineering*, vol. 190, pp. 44–53, 2018.

[216] C. Bartolozzi and G. Indiveri, "A silicon synapse implements multiple neural computational primitives," The Neuromorphic Engineer, pp. 1–3, 2008.

[217] S. Mitra, S. Fusi, and G. Indiveri, "A VLSI spike-driven dynamic synapse which learns only when necessary," in *International Symposium on Circuits and Systems, (ISCAS), 2006*. IEEE, May 2006, pp. 2777–2780.

[218] G. Bellec, F. Scherr, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets," *arXiv preprint arXiv:1901.09049*, 2019.

[219] C. Bartolozzi and G. Indiveri, "Silicon synaptic homeostasis," in *Brain Inspired Cognitive Systems, BICS 2006*, October 2006, pp. 1–4.

[220] F. L. M. Huayaney, S. Nease, and E. Chicca, "Learning in silicon beyond STDP: A neuromorphic implementation of multi-factor synaptic plasticity with calcium-based dynamics," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2189–2199, dec 2016.

[221] Z. Wang, S. Joshi, S. Savelev, W. Song, R. Midya, Y. Li, M. Rao, P. Yan, S. Asapu, Y. Zhuo *et al.*, "Fully memristive neural networks for pattern classification with unsupervised learning," *Nature Electronics*, vol. 1, no. 2, p. 137, 2018.

Appendix A

KCL Formulation of the crossbar

Equations (Ch3.4) and (Ch3.5) describe KCL continuity equations of the crossbar array. These equations at the boundary lines ($i = 1$ or $i = m$ or $j = 1$, or $j = n$) are different than the nodes inside the array. By rewriting (Ch3.44) and (Ch3.45) in voltage form and rearranging them, the nodal voltages are given by (A.1) and (A.10) for internal nodes where $2 \leq j \leq n-1$ for WLs and $2 \leq i \leq m-1$ for BLs. Where $R_{m,i,j}$ is the switching device at cell number (i, j). The boundary nodes, which are connected to the applied voltages, are given by (A.2-A.9) for WLs and (A.11- A.18) for BLs. It worth to note that $R_{s_{WL}}$ and $R_{s_{BL}}$ contains the value of access resistors and half line resistance. Now, there are $2mn$ equations in $2mn$ unknowns, and $2(m+n)$ applied signals.

$$0 = \left[\frac{1}{R_{m,i,j}} + \frac{2}{R_{WL}}\right] V_{WL,i,j} - \frac{V_{BL,i,j}}{R_{m,i,j}} - \frac{V_{WL,i,j-1}}{R_{WL}} - \frac{V_{WL,i,j+1}}{R_{WL}} + \left(C_m + C_{Ws} + 2C_{Wc}\right) \dot{V}_{WL,i,j} - C_m \dot{V}_{BL,i,j} - C_{Wc} \dot{V}_{WL,i+1,j} - C_{Wc} \dot{V}_{WL,i-1,j} \tag{A.1}$$

$$0 = \left[\frac{1}{R_{m,1,j}} + \frac{2}{R_{WL}}\right] V_{WL,1,j} - \frac{V_{BL,1,j}}{R_{m,1,j}} - \frac{V_{WL,1,j-1}}{R_{WL}} - \frac{V_{WL,1,j+1}}{R_{WL}} + \left(C_m + C_{Wc} + C_{Wsb}\right) \dot{V}_{WL,1,j} - C_m \dot{V}_{BL,1,j} - C_{Wc} \dot{V}_{WL,2,j} \tag{A.2}$$

$$\frac{V_{app_{WL1},i}}{R_{s_{WL1},i}}=\left[\frac{1}{R_{s_{WL1},i}}+\frac{1}{R_{m,i,1}}+\frac{1}{R_{WL}}\right]V_{WL,i,1}-\frac{V_{BL,i,1}}{R_{m,i,1}}-\frac{V_{WL,i,2}}{R_{WL}}+$$
$$(C_m + C_{Ws}+2C_{Wc})\,\dot{V}_{WL,i,1}-C_m\dot{V}_{BL,i,1}-C_{Wc}\dot{V}_{WL,i+1,1}-C_{Wc}\dot{V}_{WL,i-1,1} \tag{A.3}$$

$$\frac{V_{app_{WL2},i}}{R_{s_{WL2},i}}=\left[\frac{1}{R_{s_{WL2},i}}+\frac{1}{R_{m,i,n}}+\frac{1}{R_{WL}}\right]V_{WL,i,n}-\frac{V_{BL,i,n}}{R_{m,i,n}}-\frac{V_{WL,i,n-1}}{R_{WL}}+$$
$$(C_m + C_{Ws}+2C_{Wc})\,\dot{V}_{WL,i,n}-C_m\dot{V}_{BL,i,n}-C_{Wc}\dot{V}_{WL,i-1,n}-C_{Wc}\dot{V}_{WL,i+1,n} \tag{A.4}$$

$$\frac{V_{app_{WL1},1}}{R_{s_{WL1},1}}=\left[\frac{1}{R_{s_{WL1},1}}+\frac{1}{R_{m,1,1}}+\frac{1}{R_{WL}}\right]V_{WL,1,1}-\frac{V_{BL,1,1}}{R_{m,1,1}}-\frac{V_{WL,1,2}}{R_{WL}}+$$
$$(C_m+C_{Wc}+C_{Wsb})\,\dot{V}_{WL,1,1}-C_m\dot{V}_{BL,1,1} \tag{A.5}$$

$$\frac{V_{app_{WL2},1}}{R_{s_{WL2},1}}=\left[\frac{1}{R_{s_{WL2},1}}+\frac{1}{R_{m,1,n}}+\frac{1}{R_{WL}}\right]V_{WL,1,n}-\frac{V_{BL,1,n}}{R_{m,1,n}}-$$
$$\frac{V_{WL,1,n-1}}{R_{WL}}+(C_m+C_{Wc}+C_{Wsb})\,\dot{V}_{WL,1,n}-C_m\dot{V}_{BL,1,n}-C_{Wc}\dot{V}_{WL,2,n} \tag{A.6}$$

$$\frac{V_{app_{WL1},m}}{R_{s_{WL1},m}}=\left[\frac{1}{R_{s_{WL1},m}}+\frac{1}{R_{m,m,1}}+\frac{1}{R_{WL}}\right]V_{WL,m,1}-\frac{V_{BL,m,1}}{R_{m,m,1}}-$$
$$\frac{V_{WL,m,2}}{R_{WL}}+(C_m+C_{Wsb}+C_{Wc})\,\dot{V}_{WL,m,1}-C_m\dot{V}_{BL,m,1}-C_{Wc}\dot{V}_{WL,m-1,1} \tag{A.7}$$

$$\frac{V_{app_{WL2},m}}{R_{s_{WL2},m}}=\left[\frac{1}{R_{s_{WL2},m}}+\frac{1}{R_{m,m,n}}+\frac{1}{R_{WL}}\right]V_{WL,m,n}-\frac{V_{BL,m,n}}{R_{m,m,n}}-\frac{V_{WL,m,n-1}}{R_{WL}}+$$
$$(C_m+C_{Wsb}+C_{Wc})\,\dot{V}_{WL,m,n}-C_m\dot{V}_{BL,m,n}-C_{Wc}\dot{V}_{WL,m-1,n} \tag{A.8}$$

$$0 = \left[\frac{1}{R_{m,m,j}} + \frac{2}{R_{WL}}\right] V_{WL,m,j} - \frac{V_{BL,m,j}}{R_{m,m,j}} - \frac{V_{WL,m,j-1}}{R_{WL}} - \frac{V_{WL,m,j+1}}{R_{WL}} +$$

$$(C_m + C_{Wsb} + C_{Wc}) \dot{V}_{WL,m,j} - C_m \dot{V}_{BL,m,j} - C_{Wc} \dot{V}_{WL,m-1,j} \quad \text{(A.9)}$$

$$0 = \left[\frac{1}{R_{m,i,j}} + \frac{2}{R_{BL}}\right] V_{BL,i,j} - \frac{V_{WL,i,j}}{R_{m,i,j}} - \frac{V_{BL,i-1,j}}{R_{BL}} - \frac{V_{BL,i+1,j}}{R_{BL}} + (C_m + C_{Bs} + 2C_{Bc}) \dot{V}_{BL,i,j} -$$

$$C_m \dot{V}_{WL,i,j} - C_{Bc} \dot{V}_{BL,i,j+1} - C_{Bc} \dot{V}_{BL,i,j-1} \quad \text{(A.10)}$$

$$\frac{V_{app_{BL1},j}}{R_{s_{BL1},j}} = \left[\frac{1}{R_{s_{BL1},j}} + \frac{1}{R_{m,1,j}} + \frac{1}{R_{BL}}\right] V_{BL,1,j} - \frac{V_{WL,1,j}}{R_{m,1,j}} - \frac{V_{BL,2,j}}{R_{BL}} +$$

$$(C_m + C_{Bs} + 2C_{Bc}) \dot{V}_{BL,1,j} - C_m \dot{V}_{WL,1,j} - C_{Bc} \dot{V}_{BL,1,j+1} - C_{Bc} \dot{V}_{BL,1,j-1} \quad \text{(A.11)}$$

$$0 = \left[\frac{1}{R_{m,i,1}} + \frac{2}{R_{BL}}\right] V_{BL,i,1} - \frac{V_{WL,i,1}}{R_{m,i,1}} - \frac{V_{BL,i-1,1}}{R_{BL}} - \frac{V_{BL,i+1,1}}{R_{BL}} +$$

$$(C_m + C_{Bc} + C_{Bsb}) \dot{V}_{BL,i,1} - C_m \dot{V}_{WL,i,1} - C_{Bc} \dot{V}_{WL,i,2} \quad \text{(A.12)}$$

$$\frac{V_{app_{BL2},j}}{R_{s_{BL2},j}} = \left[\frac{1}{R_{s_{BL2},j}} + \frac{1}{R_{m,m,j}} + \frac{1}{R_{BL}}\right] V_{BL,m,j} - \frac{V_{WL,m,j}}{R_{m,m,j}} - \frac{V_{BL,m-1,j}}{R_{BL}} +$$

$$(C_m + C_{Bs} + 2C_{Bc}) \dot{V}_{BL,m,j} - C_m \dot{V}_{WL,m,j} - C_{Bc} \dot{V}_{BL,m,j+1} - C_{Bc} \dot{V}_{BL,m,j-1} \quad \text{(A.13)}$$

$$0 = \left[\frac{1}{R_{m,i,n}} + \frac{2}{R_{BL}}\right] V_{BL,i,n} - \frac{V_{WL,i,n}}{R_{m,i,n}} - \frac{V_{BL,i-1,n}}{R_{BL}} - \frac{V_{BL,i+1,n}}{R_{BL}} +$$

$$(C_m + C_{bc} + C_{Bsb}) \dot{V}_{BL,i,n} - C_m \dot{V}_{WL,i,n} - C_{Bc} \dot{V}_{BL,i,n-1} \quad \text{(A.14)}$$

$$\frac{V_{app_{BL1},1}}{R_{s_{BL1},1}}=\left[\frac{1}{R_{s_{BL1},1}}+\frac{1}{R_{m,1,1}}+\frac{1}{R_{BL}}\right]V_{BL,1,1}-\frac{V_{WL,1,1}}{R_{m,1,1}}-\frac{V_{BL,2,1}}{R_{BL}}+$$
$$(C_m+C_{Bc}+C_{Bsb})\dot{V}_{BL,1,1}-C_m\dot{V}_{WL,1,1}-C_{Bc}\dot{V}_{BL,1,2} \quad \text{(A.15)}$$

$$\frac{V_{app_{BL1},n}}{R_{s_{BL1},n}}=\left[\frac{1}{R_{s_{BL1},n}}+\frac{1}{R_{m,1,n}}+\frac{1}{R_{BL}}\right]V_{BL,1,n}-\frac{V_{WL,1,n}}{R_{m,1,n}}-\frac{V_{BL,2,n}}{R_{BL}}+$$
$$(C_m+C_{Bc}+C_{Bsb})\dot{V}_{BL,1,n}-C_m\dot{V}_{WL,1,n}-C_{Bc}\dot{V}_{BL,1,n-1} \quad \text{(A.16)}$$

$$\frac{V_{app_{BL2},1}}{R_{s_{BL2},1}}=\left[\frac{1}{R_{s_{BL2},1}}+\frac{1}{R_{m,m,1}}+\frac{1}{R_{BL}}\right]V_{BL,m,1}-\frac{V_{WL,m,1}}{R_{m,m,1}}-\frac{V_{BL,m-1,1}}{R_{BL}}+$$
$$(C_m+C_{Bc}+C_{Bsb})\dot{V}_{BL,m,1}-C_m\dot{V}_{WL,m,1}-C_{Bc}\dot{V}_{WL,m,2} \quad \text{(A.17)}$$

$$\frac{V_{app_{BL2},m}}{R_{s_{BL2},n}}=\left[\frac{1}{R_{s_{BL2},n}}+\frac{1}{R_{m,m,n}}+\frac{1}{R_{BL}}\right]V_{BL,m,n}-\frac{V_{WL,m,n}}{R_{m,m,n}}-\frac{V_{BL,m-1,n}}{R_{BL}}+$$
$$(C_m+C_{Bc}+C_{Bsb})\dot{V}_{BL,m,n}-C_m\dot{V}_{WL,m,n}-C_{Bc}\dot{V}_{BL,m,n-1} \quad \text{(A.18)}$$

A.0.1 Linear Switching Modeling Equations Derivation

Consequently, These equations can be arranged in matrix form as follows:

$$\mathcal{M}\mathbf{V}+\mathcal{N}\dot{\mathbf{V}}=Gu \quad \text{(A.19)}$$

where $\mathcal{M}$ and $\mathcal{N}$ are $2mn\times 2mn$ matrix, $\dot{\mathbf{V}}$ is $2mn\times 1$ vector, G is $2mn\times 2(m+n)$ matrix. And, $\dot{\mathbf{V}}$ is the time derivative of $\mathbf{V}$.

Equation (A.19) can be further described by introducing a number of intermediate coefficients. $\mathcal{M}$, $\mathcal{N}$ and G represent the coefficients of the nodal voltages and are segmented as

$$\mathcal{A}_i = \begin{bmatrix} \frac{1}{R_{S_{WL1}}(i)} + \frac{1}{R_{m,i,1}} + \frac{1}{R_{WL}} & -\frac{1}{R_{WL}} & 0 & \cdots & 0 & \\ -\frac{1}{R_{WL}} & \frac{1}{R_{m,i,2}} + \frac{2}{R_{WL}} & -\frac{1}{R_{WL}} & \cdots & 0 & \\ \vdots & \vdots & \vdots & \ddots & -\frac{1}{R_{WL}} & \\ 0 & \cdots & 0 & \cdots & -\frac{1}{R_{WL}} & \frac{1}{R_{S_{WL2}}(i)} + \frac{1}{R_{m,i,n}} + \frac{1}{R_{WL}} \end{bmatrix} \tag{A.21a}$$

$$\mathcal{B}_{\mathbf{i}}=Diag\left(\frac{-1}{R_m\left(i,1\right)},\ \frac{-1}{R_m\left(i,2\right)},\ \ldots,\ \frac{-1}{R_m\left(i,n\right)}\right) \tag{A.21b}$$

---

follows

$$\mathcal{M}=\begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{bmatrix},\ \mathcal{N}=\begin{bmatrix} \mathcal{P} & \mathcal{Q} \\ \mathcal{R} & \mathcal{S} \end{bmatrix},\ andG=\begin{bmatrix} G_{\mathbf{WL1}}G_{\mathbf{WL2}} & \mathbf{0} \\ \mathbf{0} & G_{\mathbf{BL1}}G_{\mathbf{BL2}} \end{bmatrix} \tag{A.20}$$

where $\mathcal{A},\mathcal{B},\ \mathcal{C},\ \mathcal{D},\ \mathcal{P},\ \mathcal{Q},\ \mathcal{R}$ and $\mathcal{S}$ are all $mn \times mn$ matrices. Moreover, $G_{\mathbf{WL1}}$ and $G_{\mathbf{WL2}}$ are $nm \times m$ and $G_{\mathbf{B}L\mathbf{1}}$ and $G_{\mathbf{B}L\mathbf{2}}$is $nm \times n$. The model equations can be organized so that $\mathcal{C} = \mathcal{B}$ and $\mathcal{R} = \mathcal{Q}$.

$\mathcal{A}$ and $\mathcal{B}$ are diagonal matrices where $\mathcal{A} = Diag\left(\mathcal{A}_1,\ \mathcal{A}_2,\ \ldots,\ \mathcal{A}_m\right), \mathcal{C} = \mathcal{B} = Diag(\mathcal{B}_{\mathbf{1}},\ \mathcal{B}_{\mathbf{2}},\ \ldots,\ \mathcal{B}_{\mathbf{m}})$ where $\mathcal{A}_i$ and $\mathcal{B}_i$ are $n \times n$ matrix for $1 \leq i \leq m$ and are given by (A.21). Moreover, $\mathcal{D} = \left[\mathcal{D}_1\mathcal{D}_2\ \ldots\mathcal{D}_n\right]^T$ where $\mathcal{C}_j$ and $\mathcal{D}_j$ are $m \times\ mn$ matrix, for $1 \leq j \leq n$ and are given by(A.22a). In addition, $\mathcal{R} = \mathcal{Q} = -C_m\mathbf{I}_{mn\times mn}.\mathcal{P}$ and $\mathcal{S}$ are given by (A.22b) and (A.22c), respectively. Finally, $G_{WL1},\ G_{WL2},\ G_{BL1}$ and G_{BL2} are given by (A.23).

$$\mathcal{D}=\begin{cases}\mathcal{D}\left(j,j\right)=\frac{1}{R_{s_BL1}(j)}+\frac{1}{R_{m,i,j}}+\frac{1}{R_{BL}} & \\ \mathcal{D}((m-1)n+j,(m-1)n+j)=\frac{1}{R_{s_BL2}(j)}+\frac{1}{R_{m,i,j}}+\frac{1}{R_{BL}} & \\ \mathcal{D}\left((i-1)n+j,(m-1)n+j\right)=\frac{1}{R_{m,i,j}}+\frac{2}{R_{BL}} & 2\leq i\leq m-1 \\ \mathcal{D}\left((i-1)n+j,(i-2)n+j\right)=\frac{-1}{R_{BL}} & 2\leq i\leq m \\ \mathcal{D}\left((i-1)n+j,in+j\right)=\frac{-1}{R_{BL}} & 1\leq i\leq m-1\end{cases} \tag{A.22a}$$

$$\mathcal{P}=\begin{cases}\mathcal{P}\left((i-1)n+j,(i-1)n+j\right)=C_{WC}+C_{Wsb}+C_m & i=1,m \\ \mathcal{P}\left((i-1)n+j,(i-1)n+j\right)=2C_{WC}+C_{Ws}+C_m & 2\leq i\leq m-1 \\ \mathcal{P}\left((i-1)n+j,in+j\right)=-C_{WC} & 1\leq i\leq m-1 \quad \forall\, 1\leq j\leq n \\ \mathcal{P}\left((i-1)n+j,(i-2)n+j\right)=-C_{WC} & 2\leq i\leq m\end{cases} \tag{A.22b}$$

$$\mathcal{S}=\begin{cases}\mathcal{S}\left((i-1)m+j,(i-1)m+j\right)=C_{BC}+C_{Bsb}+C_m & j=1,n \\ \mathcal{S}\left((i-1)m+j,(i-1)m+j\right)=2C_{BC}+C_{Bs}+C_m & 2\leq j\leq n-1 \\ \mathcal{S}\left((i-1)m+j,(i-1)m+j\pm 1\right)=-C_{BC} & 2\leq j\leq n-1 \quad \forall\, 1\leq i\leq m \\ \mathcal{S}\left((i-1)m+j,(i-1)m+2\right)=-C_{BC} & \\ \mathcal{S}\left((i-1)m+j,(i-1)m+n-1\right)=-C_{BC} & \end{cases} \tag{A.22c}$$

$$G_{WL1}=\begin{bmatrix}\mathbf{g_{WL1}}_1\\ \vdots \\ \mathbf{g_{WL1m}}\end{bmatrix},\mathbf{g_{WL1i}}(1,i)=\frac{1}{R_{S_{WL1}}(i)},\ 1\leq i\leq m \tag{A.23a}$$

$$G_{WL2}=\begin{bmatrix}\mathbf{g_{WL1}}_2\\ \vdots \\ \mathbf{g_{WL2m}}\end{bmatrix},\mathbf{g_{WL2i}}(n,i)=\frac{1}{R_{S_{WL2}}(i)},1\leq i\leq m \tag{A.23b}$$

$$G_{BL1}(j,j)=\frac{1}{R_{S_{BL1}}(j)} \qquad 1\leq j\leq n \tag{A.23c}$$

$$G_{BL2}((2m-1)n+j,2m+n+j)=\frac{1}{R_{S_{BL2}}(j)},\ 1\leq j\leq n \tag{A.23d}$$

A.0.2 Nonlinear Switching Modeling Equations Derivation

The modeling equations are the same proposed in the linear case except that the switching devices are function of the voltage across them. This affects only the coefficient matrix of nodal voltages,$\mathcal{M}$. Consider a simple case of nonlinearity and similarly it can be generalized to general case. Assume that the switching device transconductance is $G_{m,i,j} = G_{m_o,i,j} + G_{m_1,i,j} \times V_{m,i,j}$ where $V_{m,i,j} = V_{WL,i,j} - V_{BL,i,j}$. Thus, for inner node of a word line, the resistive part will change to (A.24). As clear that a new quadratic term of the difference between voltages of word line and bit line, respectively and another $R_{m,i,j}$ is replaced by $1/a_{i,j}$. This system can be written in a matrix form as (A.25) where $\mathcal{M}_{\mathbf{a}}$ is linear coefficient matrix and is the same as coefficient matrix,$\mathcal{M}$, in the linear case with replacing $R_{m,i,j}$ by $1/G_{m_o,i,j}$. $\mathcal{M}_{\mathbf{b}}$ nonlinear coefficients matrix and is the same as $\mathcal{M}$ with replacing $R_{m,i,j}$ by $1/G_{m_1,i,j}$ and $R_{WL} = R_{BL} = R_{S_{WL}} = R_{S_{BL}} = 0$. $\mathbf{T}$ is a transformation matrix to rotate nodal voltages vector, $\mathbf{V}$, and is given as (A.26).

$$\begin{aligned} &\left[G_{m_o,i,j} + \frac{2}{R_{WL}}\right] V_{WL,i,j} - G_{m_o,i,j} V_{BL,i,j} + \\ &G_{m_1,i,j}(V_{WL,i,j} - V_{BL,i,j})^2 - \frac{V_{WL,i,j-1}}{R_{WL}} - \frac{V_{WL,i,j+1}}{R_{WL}} \end{aligned} \tag{A.24}$$

$$\mathcal{N}\dot{\mathbf{V}} = -\mathcal{M}_{\mathbf{a}}\mathbf{V} - \mathcal{M}_{b_1}(\mathbf{V} - \mathbf{T}\mathbf{V})^2 + G\mathbf{u} \tag{A.25}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{0}_{mn} & \mathbf{I}_{mn} \\ \mathbf{I}_{mn} & \mathbf{0}_{mn} \end{bmatrix} \tag{A.26}$$

Similarly, the higher nonlinear coefficients can be obtained creating the matrix form of generalized system of equations given in (17).